Reprinted from JOURNAL OF COMPUTER AND SYSTEM SCIENCES
Vol. 6, No. 3, June 1972
Printed in Belgium

Real-Time Language Recognition by One-Dimensional Cellular Automata*

ALVY RAY SMITH III

New York University, Bronx, New York 10453

Received July 7, 1971

Pattern recognition by parallel devices is investigated by studying the formal language recognition capabilities of one-dimensional cellular automata. The precise relationships of cellular automata to iterative automata and to Turing machines are established: In both cases, cellular automata are inherently faster. The relationship of context-free languages to the languages recognized in real time by bounded cellular automata is detailed. In particular, nondeterministic bounded cellular automata can recognize the context-free languages in real time. The deterministic case remains open, but many partial results are derived. Finally, closure properties and cellular automata transformation lemmas are presented.

1. INTRODUCTION

The cellular automaton model, defined in the following section, has been used by several theorists [1, 2, 6, 7, 19] for proving the existence of nontrivial self-reproducing computing machines. All these proofs employ the step-by-step simulation of a universal Turing machine, the most serial computer model, by a cellular automaton, a highly parallel computer model. A primary purpose of this paper is the exploitation of the inherent, but neglected, parallelism of cellular automata. This purpose is effected by treating the cellular automaton as a pattern recognizer. Specifically, attention is focused on one-dimensional pattern recognition, with formal languages as the classes of patterns recognized. Speed of recognition is emphasized throughout to make it clear that a cellular automaton is not just another recognition device for well-known languages but is, in fact, a fast recognition device.

A formal definition of a cellular automaton is presented in the next section. Intuitively, however, a cellular automaton is an array of identical finite-state Moore machines, called cells, which are uniformly interconnected. At time $t = 0$, it receives

* The research reported herein was presented at the I.E.E.E. Eleventh Annual Symposium on Switching and Automata Theory, Santa Monica, California, October 1970. The research was supported by the National Science Foundation under Grant GK-5406.

a spatial pattern of inputs, called an initial configuration. The temporal sequence of configurations of states of the array generated autonomously after $t = 0$ is, in general, the object of interest. Each configuration in a sequence is the image of a function, the global transition function, of the preceding configuration, with only one such function associated with a given cellular automaton. Thus cellular automata are a subclass of the tessellation automata [25], each of which may have a set of global transition functions.

To make the observation of configurations generated by a cellular automaton effective, various finiteness conditions are imposed. For example, an initial configuration may be required to have only a finite number of nonquiescent cells, where a quiescent cell is a cell in a specially designated state, the quiescent state. Here the condition will be that an array contain only a fixed number of cells from $t = 0$ onwards which may be nonquiescent. In one dimension, these bounded cellular automata are the bilateral iterative networks of Hennie [13] with a unit delay between each two adjacent cells. The paper of Kasami and Fujii [15], written in the terminology of iterative networks, is closely related to the work here.

Another closely related reference is Cole [9], which presents an alternate parallel computer model, called an iterative automaton. An iterative automaton is also a uniform array of identical cells but is not autonomous, receiving a temporal input pattern at one specially designated input–output cell. The temporal sequence of outputs generated by this one cell is the object of interest. Bounded cellular automata will be shown to be inherently faster than iterative automata.

2. Definitions and Basic Lemmas

The abbreviation n — D is used to mean n-dimensional. The 1 — D case will be the usual case and should be assumed unless otherwise indicated.

A *finite-state machine* (FSM) is a 5-tuple (X, Y, Q, f, g) with X, Y, and Q all finite, nonempty sets called inputs, outputs, and states, respectively. The next-state function is $f: Q \times X \to 2^Q$ and $g: Q \to Y$ is the output function. A *cell* is an FSM (Q^2, Q, Q, δ, i_Q), denoted (Q, δ), where i_Q is the identity on set Q. If the range of f for a given FSM be a set composed entirely of singletons, then the FSM is said to be *deterministic*, else *nondeterministic*. The prefix D shall be used throughout to denote the deterministic special case. Thus, for example, a DFSM is a deterministic FSM. Furthermore, singletons will be denoted by their single element—e.g., $\{b\}$ will be abbreviated b.

Let $C = (Q, \delta)$ be a given cell. Then a *cellular space*, or *cellular automaton*, Z is an assignment of one copy of C to each integer point on the real line such that the input to cell i (i.e., the copy of C at point $i \in I$, the integers) is the output pair from cells $i - 1$ and $i + 1$, for all i. If C is nondeterministic (deterministic), then Z is nondeterministic (deterministic). C is said to be *the cell of* Z. Cells $i - 1$, i, and $i + 1$ form the *neighborhood* of cell i. Cells $i - 1$ and $i + 1$ are the *neighbors* of cell i. It has been

shown that there is no loss of generality in assuming the 3-cell neighborhood adopted here [9, 21]. Denote Z by the triple (Q, δ, q_0), where Q is called the *state set* of Z, δ is called the *local transition function* of Z, and q_0 is the *quiescent state* of Z with the property $\delta(q_0, q_0, q_0) = q_0$. It is convenient to rearrange the domain of δ so that $\delta(x, y, z)$ is always the state of cell i at time $t + 1$ if cells $i - 1$, i, and $i + 1$ are in states x, y, and z, respectively, at time t.

All cells in a cellular space Z are assumed to change state simultaneously. Thus a *global transition function* Δ can be defined for Z as the simultaneous application of δ at all cells in Z. Formally, let a *configuration* $\hat{c}$ be an assignment of states from Q to each cell in a cellular space—i.e., $\hat{c}: I \to Q$. Then, for χ the set of all configurations in Z, $\Delta: \chi \to \chi$ is defined by $[\Delta(\hat{c})](i) = \delta(\hat{c}(i - 1), \hat{c}(i), \hat{c}(i + 1))$. A cellular space is operated as follows: An arbitrary configuration is assumed at time $t = 0$, an *initial configuration*, and the sequence of configurations which occurs autonomously by repeated applications of Δ is observed. This is in contrast to the operation of an iterative automaton defined next.

An *iterative automaton* [9] is a cellular space with cell 0, the *distinguished cell*, augmented by an extra input, the *external input*, and an extra output, the *external output*. That is, the distinguished cell is an FSM $(Q^2 \times X', Q \times Y', Q, \delta', \beta')$ with $\beta'(q) = (q, y)$. For convenience, let β_2' give the second element of β'— i.e., $\beta_2'(q) = y$. All other cells are as defined for a cellular automaton. An iterative automaton is nondeterministic if either its cell or its distinguished cell is nondeterministic, else it is deterministic. An iterative automaton is operated as follows: All cells, including the distinguished cell are assumed to be quiescent (i.e., in state q_0) at $t = 0$. A temporal sequence of inputs from set X' is applied to the external input of the distinguished cell and the sequence of outputs from set Y' are observed at the external output of the distinguished cell.

Both cellular and iterative automata may be employed as language acceptors. The following definitions explain this mode of operation for each device.

DEFINITION 1. A *bounded cellular space* (BCS), denoted by the 4-tuple (X, Q, δ, b), is a cellular space (Q, δ, q_0) restricted as follows:

(1) $b \in Q$ is a specially designated *boundary state*;

(2) $X \subset Q_b = Q - b$ is the *initial alphabet*;

(3) $\delta(q, q'', q') = b$ if and only if $q'' = b$, for arbitrary q and q' in Q;

(4) two and only two cells, the *boundary cells*, are in state b at time $t = 0$.

Thus, although a cellular space is of infinite length, a bounded cellular space may be considered finite because the states of the cells between and including the boundary cells are independent of the states of all other cells in the cellular space. This is a consequence of the definition of the boundary state b, which does not permit the creation or destruction of a boundary cell after $t = 0$, and of the 3-cell neighborhood.

Therefore, a *retina* for a BCS, the cells between but not including the two boundary cells, is fixed for all time after $t = 0$. All cells other than the boundary cells and the retina may be assumed quiescent. The retina is the pattern processing portion of a BCS, where a *pattern* is a configuration restricted to the retina. In general, for a finite, nonempty set W, W^* denotes the set of all finite strings formed from concatenations of elements of W and includes the empty string e. Hence a pattern is an element of $Q_b{}^*$.

DEFINITION 2. The *pattern transition function* for a BCS $Z = (X, Q, \delta, b)$ is the function $F: Q_b{}^* \to (2^{Q_b})^*$ such that

$$F(q_1q_2 \cdots q_n) = \delta(b, q_1, q_2)\,\delta(q_1, q_2, q_3) \cdots \delta(q_{n-2}, q_{n-1}, q_n)\,\delta(q_{n-1}, q_n, b)$$

and $F(e) = e$, for n the number of cells in a retina of Z. Let F^t denote t successive applications of F to a pattern.

DEFINITION 3. Let $R: (2^{Q_b})^* \to 2^Q$ be the *extraction function* which extracts the present states of the rightmost cell, the *accept cell*, in a retina: $R(Q_1Q_2 \cdots Q_n) = Q_n$ and $R(e) = b$.

DEFINITION 4. A BCS $Z = (X, Q, \delta, b)$ is said to *accept the language* $L \subseteq X^*$ *(on* A*)* if, for arbitrary $x \in L$, there is a time t such that $R(F^t(x)) \cap A \neq \phi$, where $A \subset Q$ is a set of *accept states* disjoint from X. A BCS used in this manner will be denoted by the 5-tuple (X, Q, δ, b, A) and called a *BCS acceptor*. Z is said to *recognize* L if it accepts L on A_1 and accepts $L' = X^* - L$ on A_2, where $A_1 \cap A_2 = \phi$. If Z recognizes L, then it *rejects* L'. Such a Z is called a *BCS recognizer*.

A 1 — D language-accepting device is said to *accept* (*recognize*) a language L *within time* $T(n)$ if, for any x of length n, it can determine that $x \in L$ (that $x \in L$ or $x \in L'$) within $T(n)$ steps, where $T: N \to N$ is a total *time function* on the positive integers. $T(n) = n$ is called *real time*; $T(n) = cn$, c a constant, is called *linear time*.

DEFINITION 5. L is a *BCS language* if there is a BCS acceptor $Z = (X, Q, \delta, b, A)$ such that $L = L(Z) = \{x \in X^* \mid (\exists t)[R(F^t(x)) \cap A \neq \phi]\}$. Similarly, L is a *BCS predicate* if it is recognized by some BCS recognizer. A *linear-time* BCS language (predicate) is a BCS language (predicate) which is accepted (recognized) within $T(n) = cn$. If $c = 1$, then the language (predicate) is said to be *real-time*.

Thus a string is accepted by a BCS if, when embedded between two boundary cells in some BCS acceptor at $t = 0$, action of the pattern transition function causes the accept cell to pass eventually into a set of states including an accept state.

DEFINITION 6. Consider an iterative automaton V with a set $A \subset Y'$ of accept states and external input alphabet $X' = X \cup 0$, where $0 \notin X$. Then V is said to accept $x_1x_2 \cdots x_n \in X^*$ if there is a time $t \geqslant n$ such that $\beta_2'(\delta'(q_1, (q_2, x_t), q_3)) \cap A \neq \phi$ and the external input string has been $x_1x_2 \cdots x_n \cdots x_t$, $x_i = 0$ for $n + 1 \leqslant i \leqslant t$.

Here δ' is generalized to subsets of 2^Q and its arguments are rearranged in accordance with the previous treatment of δ for BCS. A language $L \subseteq X^*$ is accepted by V if V accepts all strings $x \in L$. Such a V is called an *iterative acceptor*, and if $t = n$, then it is called a *real-time iterative acceptor*. The set of all strings on X accepted by iterative acceptor V is denoted $L(V)$ and said to be the language accepted by V.

DEFINITION 7. A *bounded iterative automaton* V is an iterative automaton restricted as follows:

(1) $b \in Q$ is a boundary state:
(2) $\delta(q, q'', q') = b$ if and only if $q'' = b$, for arbitrary q and q' in Q;
(3) at $t = 0$, two and only two cells, cell i and cell j, are boundary cells (i.e., in state b) with $i < 0$ and $j > 0$.

If $-i = j = n + 1$ for each string $x_1x_2 \cdots x_n$ in $L(V)$, then V is called a *real-space iterative acceptor*.

Hence a real-space iterative acceptor has the same memory size as a real-time iterative acceptor. Since the addition of boundary cells at positions $n + 1$ and $-(n + 1)$ in a real-time iterative acceptor will not alter its computation, a real-time iterative acceptor is taken to be a real-space iterative acceptor.

A third class of machines, of secondary importance to this paper, are the multitape nondeterministic Turing machines described informally below. A formal definition appears, for example, in [5]. An *m-tape on-line Turing machine* (TM) T operates as follows: At $t = 0$, the finite-state control head of T is in a given initial state and scanning the leftmost nonblank symbol of the input tape, which contains a string w from given alphabet Σ. All other tapes, the m working tapes, are blank. At each step, T nondeterministically changes the scanned symbols on its working tapes, moves one square right, left, or not at all on each working tape independently, moves one square right on its input tape, and changes the state of its control head. The language accepted by T is the set of all input tapes $w \in \Sigma^*$ for which T may halt eventually in some predesignated final state with all working tapes blank. The languages accepted in real time by m-tape on-line TM are the *quasirealtime languages* [5].

Two classes of machines are said to be *equivalent* if they accept precisely the same class of languages. A general knowledge of formal language theory, associated automata, and terminology is assumed [1, 14].

The following lemma is a basic cellular automata theory result and will be used frequently in the sequel. Here and throughout the paper, a^n will denote a string of n copies of the symbol a.

THE FIRING-SQUAD LEMMA [24]. *There is a* DBCS $Z = (Q, \delta, b)$ *with specially designated states* $¢, \$ \in Q$ *and quiescent state* 0 *such that* $F^t(0^{n-1}¢) = \$^n$ *for* $t = 2n - 2$ *and* $[F^t(0^{n-1}¢)](i) \neq \$$ *for* $0 \leqslant t < 2n - 2$ *and for all* $i \in I$ *in the retina.*

In the terminology of the firing-squad literature, this lemma says that a general, at the right end of a line of soldiers initially at rest (0), issues a command (¢) to fire. After an elapsed time of precisely twice the number of soldiers, all soldiers and the general fire ($) simultaneously and none fires before this time. The general could also be at the left end of the line. In fact, it has been shown [18, 23] that he can be anywhere in the line with no loss in time.

It is known that $2n - 2$ is the minimum possible time to fire for a firing squad. Hence the firing squad transformation is accomplished in linear time. The next lemma can be used to speed up linear time to almost real time. It is the speed-up lemma for BCS [4]. Closely related are the speed-up theorems of cellular automata [21] and iterative acceptors [9]. All these theorems are based on the obvious fact that if the information originally held in several cells is packed into one cell, then the cell can process that information more quickly because the time required for accessing it has been reduced. The speed-up is not a strict increase in speed by a constant factor because on the order of n time units are required to perform the initial packing.

THE SPEED-UP LEMMA. *Let k be an arbitrary positive integer. For an arbitrary* DBCS *acceptor $Z = (X, Q, \delta, b, A)$ with $|Q| = r$, there are a constant c and a* DBCS *acceptor $Z' = (X, Q', \delta, b, A)$ with $|Q'| = cr^k$ such that, if Z accepts language L within time $T(n)$, then Z' accepts L within time $(T(n)/k) + n$.*

In particular, if Z accepts L within linear time, then there is a Z' which accepts L within $T(n) = (1 + \epsilon)n$, $\epsilon > 0$, or almost real time. Unfortunately the cost of a speed-up by a factor of k, in terms of the size of the state set, increases exponentially with k. Hence this paper concentrates on exactly real-time results with only occasional resort to the speed-up lemma.

3. LANGUAGE-RECOGNITION RESULTS

LEMMA 3.1. *The class of* BCS *languages is equivalent to the class of context-sensitive languages* (CSL).

COROLLARY 3.1.1 [15]. *The class of* DBCS *languages is equivalent to the class of* DCSL.

The lemma follows from a straightforward simulation of a linear-bounded automaton (LBA), as defined in [16] for example, by a BCS, and vice versa. There are two ways to simulate an LBA by a BCS. The first, easily employed here, is to embed the LBA tape in the BCS with the boundary cells as endmarkers. Then the movement of the head along the stationary tape is simulated by "movement" of a simulating state from cell to cell in the BCS. A nondeterministic move is accomplished in two steps: (1) a

nondeterministic choice is made, and (2) the result of (1) is deterministically simulated. The other simulation method [8] assumes the head is stationary and "moves" the tape beneath it. This latter technique will be used in the proof of Theorem 3.8. The details of the simulations for the proof of Lemma 3.1 are left to the reader. It is well known that the CSL (DCSL) are precisely the languages accepted by LBA (DLBA).

Hence pattern recognition by cellular automata reduces to problems in the theory of context-sensitive languages. The same is true, in a sense which is made precise below, for iterative acceptors.

The next proof is the first of several in this paper which utilize a *space-time diagram* as a heuristic aid. The device is not original, having been used successfully, for example, by Waksman [24] and Fischer [10]. It consists of a 1 — D cellular machine arrayed across a page (the space dimension) and lines directed down the page (the time dimension), proceeding away from the cellular array at angles of at least 45°. See Fig. 1. These lines represent the flow of state information in the array. The units in

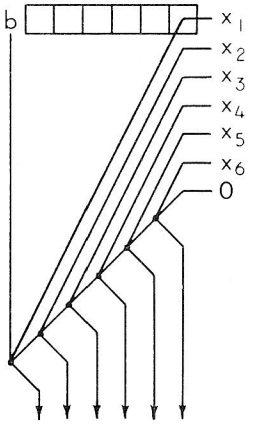

FIG. 1. Space–time diagram for Lemma 3.2.

each dimension are such that a 45° line represents a flow of information at the rate of one cell per time step, *unit speed.*

An informal programming language is helpful in discussion of space–time diagrams. A *q-cell* is a cell in state $q \in Q$ at $t = 0$. A *q-pulse propagating right* (*left*) *at* $(1/k)$ *unit speed from a q-cell* is represented by a line of slope $-k$ (k), $k > 0$ an integer, originating from each q-cell. It carries the information that the q-cell was in state q at $t = 0$ to other cells in the array, and compiles, for "don't care" states (—), into $\delta(-, q, -) = q^{(1)}$, $\delta(-, q^{(1)}, -) = q^{(2)}, \ldots, \delta(-, q^{(k-2)}, -) = q^{(k-1)}$, $\delta(q^{(k-1)}, -, -) = q$. The last entry is replaced by $\delta(-, -, q^{(k-1)}) = q$ for a left-propagating pulse. In either case, the cell is said to have *sent* the pulse. Two pulses are said to *collide* if their lines, in a space–time

diagram, intersect. For, say, a p-pulse propagating right at unit speed and a q-pulse propagating left at unit speed, this notion compiles into (1) $\delta((p, -), -, (-, q)) = (p, q)$, for a p-cell an even distance from a q-cell, or (2) $\delta((p, -), (q, -), -) = (p, q)$ for an odd distance, where each state is given two coordinates, or *channels*, for distinguishing pulses. A q-pulse is *annihilated* if it ceases to propagate after a collision. A *q-cell (q-pulse) maintaining its position* is represented by a line proceeding down the page—i.e., with infinite slope. A *pq-boundary* is a p-cell, with a q-cell as right neighbor, maintaining its position. A *right (left) propagating q-pulse is reflected by a p-cell at $(1/k)$ unit speed* if a q-pulse is annihilated by collision with a p-cell and a q-pulse is sent left (right) at $(1/k)$ unit speed from the p-cell at the time of collision.

LEMMA 3.2. *The class of languages accepted by (deterministic)* BCS *is accepted by (deterministic) real-space iterative acceptors.*

Proof. A BCS can be simulated—in its language-accepting mode—by an initially quiescent iterative acceptor as follows. Let q_0 be the quiescent state of the iterative acceptor. An initial pattern $x_1x_2 \cdots x_n$ for the cellular space is input temporally into the distinguished cell of the simulating iterative acceptor, leftmost symbol x_1 first. Each symbol is shifted left until the entire input pattern is arrayed spatially in the cells of the iterative acceptor as indicated in Fig. 1. As each x_i enters the distinguished cell, it is sent left at 1/2 unit speed as an x_i-pulse. The first 0 input to the distinguished cell is sent left as a 0-pulse propagating at unit speed. As each x_i-pulse collides with the 0-pulse, it is reflected one cell to the right, where it creates an x_i-cell which maintains position. The collision of the 0-pulse with the left b-cell causes the creation of a command-to-fire signal ¢ as in the firing squad lemma. This initiates a firing squad consisting of the n x_i-cells. When the firing squad fires (\$), all x_i-cells simultaneously begin to function exactly like the cells of the simulated BCS. If the BCS would accept the initial pattern, then its accept cell would enter an accept state. But by the input procedure above, the simulated accept cell is the distinguished cell of the iterative acceptor. Q.E.D.

LEMMA 3.3. *For an arbitrary (deterministic) iterative automaton, there is a (deterministic) cellular space which simulates it in real time.*

Proof. The input string $x_1x_2 \cdots x_n$ to the simulated iterative automaton is embedded one symbol per cell in the simulating cellular space so that the nonquiescent part of the initial configuration is $@x_n \cdots x_2x_1@$, where @ is a special marker state. The right marker state also serves to designate one cell in the cellular space as the simulated distinguished cell of the iterative automaton. Then the input string is shifted one symbol at a time, at unit speed, into the right @-cell. Besides being capable of this shifting operation, each cell is also able to simulate a cell in the simulated iterative automaton. That is, the states of each cell consist of three channels, the first for

shifting the input string, the second for simulating the iterative automaton, and the third for holding the output string as it is shifted left out of the cell simulating the the distinguished cell at unit speed. Q.E.D.

THEOREM 3.4. *The class of (deterministic)* BCS *is equivalent to the class of (deterministic) real-space iterative acceptors.*

Proof. The converse of Lemma 3.2 is required, but only slight modification of the proof of Lemma 3.3 is necessary for this goal. Clearly, Lemma 3.3 is true for iterative acceptors as a special case of iterative automata. Specifically, the left and right @-cells in the proof become the boundary cells. The simulating cellular space is supplied with three channels. One is used for shifting the input into the right boundary cell, the second simulates the n cells to the left of the distinguished cell, and the third simulates the n cells to the right of the distinguished cell. Q.E.D.

COROLLARY 3.4.1. *If an arbitrary (deterministic) iterative acceptor accepts language L within time $T(n)$, then there is a* BCS (DBCS) *which accepts L within time $T(n)$.*

The next theorem states that the converse of Corollary 3.4.1 is false and makes explicit the difference in computing speeds between cellular spaces and iterative acceptors. However, a lemma must first be proved. Let a *palindrome* in X^* be a word of the form ww^R, where $w \in X^*$ and w^R is the string w written in reverse order. The set of palindromes has been shown [9] to be a real-time iterative acceptor language and is hence a real-time DBCS language by Corollary 3.4.1. Lemma 3.5 below says more and uses a different and very simple proof technique (see also [11]).

LEMMA 3.5. $L_1 = \{ww^R \mid w \in X^*, \mid w \mid \geqslant 1\}$ *is a real-time* DBCS *language; so are* $L_2 = L_1X^*$ *and* $L_3 = X^*L_1$.

Proof. Consider the space–time diagram of Fig. 2. Each cell sends its state to the left and right at unit speed. The center cell of the array is determined by the collision of the two boundary pulses sent by the boundary cells. Each cell acts as if it were the center cell (the right cell of the two center cells if n is even) of a palindrome. Should a p-pulse and a q-pulse, $p \neq q$, ever collide at cell i then that cell goes into a special state, say #, signifying that cell i cannot be the center cell of a palindrome, and it remains in that state. The b-pulse sent by the right boundary cell acts as a "collection" pulse, which reflects from the center cell of the pattern to the right boundary at unit speed. Suppose it is required that this pulse send an accept state as a pulse to the right if and only if it finds a non-# center cell at the center of the given pattern. Then Z_1 has been designed such that $L_1 = L(Z_1)$ is accepted within real time. But suppose the collection pulse is required to send an accept state to the right if and only if it finds at least one non-# cell before, or as, it collides with the center cell of the pattern. Then

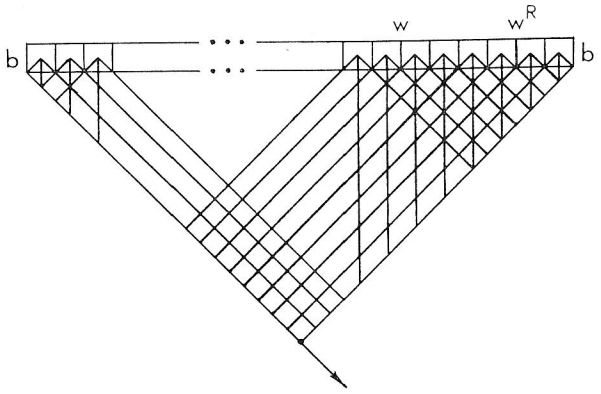

FIG. 2. Palindrome recognizer.

real-time acceptor Z_3 has been designed such that $L_3 = L(Z_3)$. If the collection pulse were instead the other boundary pulse, and if it acted just as does the collection pulse of Z_3, then Z_2, the real-time acceptor of L_2, has been designed. Q.E.D.

THEOREM 3.6. *There is a context-free language, not accepted within real time by any deterministic iterative acceptor, which is a real-time* DBCS *language. (Hence* DBCS *are inherently faster than deterministic real-space iterative acceptors.)*

Proof. Consider the language X^*L_1'', where $L_1'' = \{wvw^R \mid w \in X^*, \mid wvw^R \mid \geqslant 3\}$. Cole [9] has proved that no deterministic real-time iterative acceptor (of any dimension) can accept X^*L_1''. A simple finite-state machine added to each cell in Z_3 of Lemma 3.5 yields Z_3'' which accepts X^*L_1'' in real time. Q.E.D.

Remark. Although the result stated in Theorem 3.6 is never explicitly mentioned, Kasami and Fujii [15] essentially prove it based on an equivalence-class counting argument. In fact, they essentially prove the stronger result: The linear deterministic context-free language

$$L = \{du_m\, du_{m-1} \cdots du_1 c v_1\, dv_2 d \cdots v_m d \mid u_i{}^R = v_i w_i\,,\ u_i\,,\ v_i\,,\ w_i \in X^*\} \subset (X \cup \{c, d\})^*$$

is a real-time DBCS language but not a real-time iterative acceptor language.

THEOREM 3.7. *There is a context-free language, not recognized within time $T(n) = cn^2$ by any one-tape off-line deterministic Turing machine, which is a real-time* DBCS *language. (Hence* DBCS *are inherently faster than single-tape Turing machines.)*

Proof. A one-tape, one-head deterministic Turing machine M cannot recognize $L = L_1 \cup \{wdw^R \mid w \in X^*, d \in X\}$ within square time, where L_1 is as in Lemma 3.5. That is, for any such M which recognizes L, there is a constant c such that M does not recognize L within time $T(n) = cn^2$[1, 3]. But L is a real-time DBCS language by simple modification of the scheme used for recognizing L_1 in Lemma 3.5. Q.E.D.

These results suggest the following interesting, and as yet unsolved, problem: Are the context-free languages a subset of the real-time DBCS languages? It is not difficult to show that they are a subset of the real-time BCS languages.

THEOREM 3.8. *The quasirealtime languages are real-time* BCS *languages.*

Proof. The quasirealtime languages are precisely the languages accepted in real time by nondeterministic on-line multitape Turing machines. Cole [8] has shown how to simulate a deterministic off-line multitape Turing machine with a deterministic iterative acceptor in real time. A technique very similar to his is described briefly below. It is easily adapted to the on-line nondeterministic case. Then the theorem follows from Corollary 3.4.1.

A deterministic iterative acceptor V simulates a one-tape off-line deterministic Turing machine M as follows: The distinguished cell simulates, at all times, the head of M, the scanned square on the tape of M, and the two squares on either side of the scanned square. All other cells simulate two consecutive squares on the tape. Thus an instantaneous description of M such as $\ldots\ x_{-6}x_{-5}x_{-4}x_{-3}x_{-2}x_{-1}qx_0x_1x_2x_3x_4x_5x_6\ \ldots$ is represented in V by the following configuration at time t (semicolons separate the contents of distinct cells and parentheses enclose contents of the distinguished cell):

$$\ldots;\ x_{-6}x_{-5};\ x_{-4}x_{-3};\ (x_{-2}x_{-1}x_0',\ q'x_1,\ @x_2);\ x_3x_4;\ x_5x_6;\ \ldots,$$

where q' and x_0' are the new state and new symbol, respectively. Special state coordinate @ propagates right at unit speed causing each simulated tape symbol to move left one position. Simultaneously, the excess symbol x_0' causes a pulse to propagate left at unit speed which forces each simulated tape symbol to move left one position. Thus the configurations at times $t + 2$ and $t + 3$, assuming that no further state changes by M are simulated, are as shown below:

$$\ldots;\ x_{-6}x_{-5};\ x_{-4}x_{-3}x_{-2};\ (x_{-1}x_0',\ q'x_1,\ x_2x_3);\ @x_4;\ x_5x_6;\ \ldots,$$

$$\ldots;\ x_{-6}x_{-5}x_{-4};\ x_{-3}x_{-2};\ (x_{-1}x_0',\ q'x_1,\ x_2x_3);\ x_4x_5;\ @x_6;\ \ldots\ .$$

Since the changes indicated above propagate away from the distinguished cell at unit speed, it is not necessary for the distinguished cell to wait for the simulated tape to be completely shifted before continuing its simulation of M. In fact, it is clear that V can simulate M in real time, using an analogous technique for a left move as used in the demonstration above for a right move.

Since the simulated head cell remains fixed and the simulated tape moves beneath it the technique can also be used for a multitape machine. Furthermore, since a nondeterministic move by M can affect only one cell in a simulation by V, the distinguished cell, then the generalization of the technique to the nondeterministic case is immediate with no loss in the speed of simulation, or real time. However, the technique requires the input be "packed," two symbols per cell (except for the distinguished cell which requires five symbols). Unfortunately, this packing requires time on the order of n [4]. For the case of an on-line TM, it is now shown that packing can be carried out while the simulation occurs and, hence, the entire simulation requires only real time.

But the read-only input tape of the simulated on-line machine need not be packed since it simply moves left at unit speed into the distinguished cell. Since the working tapes are initially blank, the simulated working tapes may be assumed to be already packed, two blanks per cell, at $t = 0$. Q.E.D.

COROLLARY 3.8.1. *The following are quasirealtime and hence real-time* BCS *languages* [5]: (1) *the context-free languages*, (2) *the real-time definable languages*, (3) *the nondeterministic real-time storage languages*, (4) *the real-time counter languages, and* (5) *the languages generated by linear-time grammars.*

In contrast, there is the following result.

THEOREM 3.9. *There is a context-free language, not recognized within real time by any multitape Turing machine, which is a real-time* DBCS *language.*

Proof. Hartmanis and Stearns have shown [12] that the language

$$L = \{yxdy'x^R \mid x \in \{0, 1\}^*; y, y' \in e \cup \{0, 1, d\}^*d\}$$

cannot be recognized by a multitape Turing machine in real time. A DBCS Z is now illustrated which accepts L in real time. Consider Fig. 3.

Each cell sends a q_i-pulse containing its initial state q_i right and left at unit speed. Should two d-pulses collide at cell i, then cell i begins to act as if it were the center cell of a palindrome as in the proof of Lemma 3.5. Should a d-pulse propagate to or through cell i, then cell i ceases to check for palindromes. Should a d-pulse (or b-pulse) collide with the right b-pulse at cell i while cell i is checking for palindromes, then the b-pulse checks for a non-# at cell i (i.e., for existence of a palindrome). One such non-# before or at the collision of the left and right b-pulses is sufficient for propagation of an accept state to the right.

The form xdx^R is a special case of L which Z must accept. For this case, have each d-cell also act as a center cell of a palindrome. Should a d-pulse ever propagate to or through such a cell, then it ceases checking for palindromes. However, should a

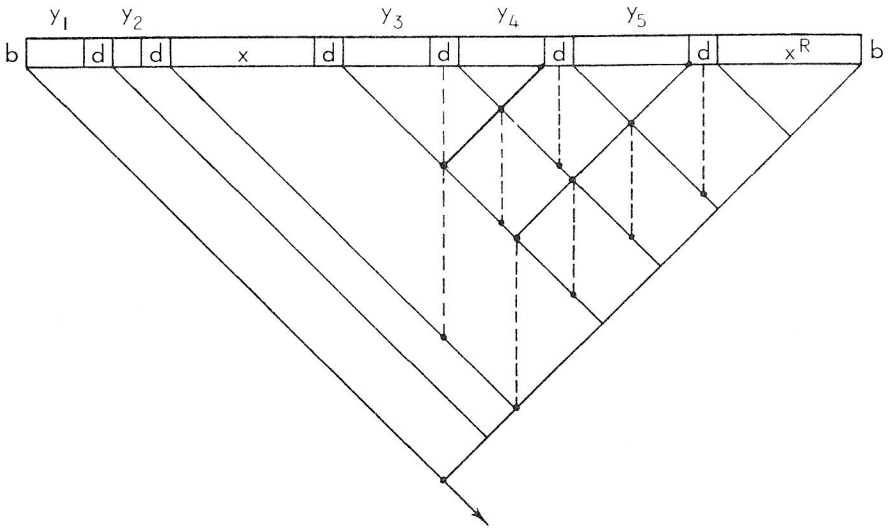

FIG. 3. Multitape language recognizer.

d-pulse (or *b*-pulse) and the right *b*-pulse ever collide at such a cell while it is checking for palindromes, then the *b*-pulse checks for a non-# just as above. Q.E.D.

Kasami and Fujii [15] have shown that the context-free languages are a proper subclass of the DBCS languages. Their real-time results are summarized in the theorem below (see also [20], where the languages accepted by deterministic pushdown automata without *e*-moves are called *e-free deterministic* context-free languages.

THEOREM 3.10. (1) *The linear context-free languages are real-time* DBCS *languages.* (2) *The e-free deterministic context-free languages are real-time* DBCS *languages.*

A corollary to case (1) is immediate from Lemma 3.3 and the speed-up lemma for iterative acceptors [9].

COROLLARY 3.10.1. *Any linear context-free language can be accepted within* $T(n) = (1 + \epsilon)n$, $\epsilon > 0$, *by a deterministic real-space iterative acceptor.*

Kosaraju [17] derived the closely related results: Deterministic iterative acceptors accept the context-free languages within $T(n) = (1 + \epsilon)n^2$; deterministic iterative acceptors, generalized to two dimensions, accept the context-free languages within $T(n) = (1 + \epsilon)n$.

In comparison to case (2) of Theorem 3.10, there is the result below.

THEOREM 3.11. *There are inherently ambiguous context-free languages which are real-time* DBCS *languages. Hence there are nondeterministic context-free languages which are real-time* DBCS *languages.*

Proof. $L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$ is an inherently ambiguous context-free language. L is also a real-time DBCS language as is shown by constructing real-time DBCS acceptor Z such that $L = L(Z)$. Consider Fig. 4 which illustrates a DBCS Z_1 for

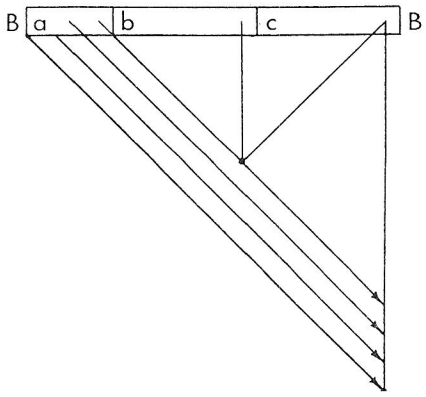

FIG. 4. Recognizer for $\{a^i b^i c^j\}$.

accepting $\{a^i b^j c^j\}$. To avoid confusion, let B be the boundary state here. The Ba, ab, bc, and cB boundaries are specially marked and maintain position. The cB-boundary sends a pulse left at unit speed checking for all c's. Each ab-boundary sends a pulse right at unit speed checking for all b's. Should an ab-boundary pulse collide with the cB-boundary pulse at a bc-boundary, having seen only b's and c's respectively, then form $b^j c^j B$ is guaranteed. The ab-boundary pulse carries this information to the accept cell. Meanwhile, each cell initially in state a sends a pulse right at unit speed. The collision of the ab-boundary pulse with the accept cell causes the accept cell to begin checking for the arrivals of only a-pulses until the left boundary pulse arrives. Should this condition occur then Z_1 accepts.

Z_2 is constructed to accept $\{a^i b^i c^k\}$ by the scheme illustrated in Fig. 5, where the dashed lines are ignored in this proof. All boundaries maintain position as in Fig. 4. Each bc-boundary sends a pulse left at unit speed checking for all b's. The Ba-boundary, if it exists, sends a pulse right at unit speed checking for all a's. Once the Ba-boundary and bc-boundary pulses collide at an ab-boundary, having encountered only a's and b's respectively, the Ba-boundary pulse continues right checking for all b's, a collision with a bc-boundary, and finally all c's until collision with the right boundary. Should this condition occur, then Z_2 accepts.

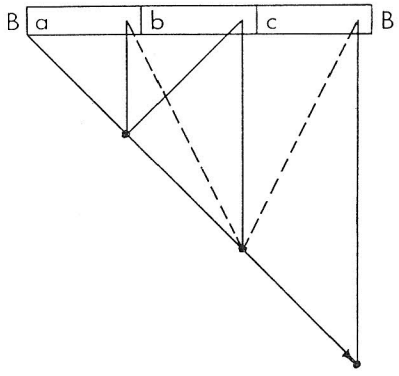

FIG. 5. 1-D French-flag recognizer.

In Section 4, the real-time DBCS languages are shown to be closed under union. Hence Z is guaranteed such that $L(Z) = L(Z_1) \cup L(Z_2)$. Q.E.D.

LEMMA 3.12. $L_4 = \{ww \mid w \in X^*\}$ *is a real-time* DBCS *language.*

Proof. Cole [9] has demonstrated an iterative acceptor which accepts L_4 in real time. The lemma follows from Corollary 3.4.1. Q.E.D.

LEMMA 3.13. $L_5 = \{a^m b^m c^m \mid m \geqslant 1\}$ *is a real-time* DBCS *language.*

Proof. Reconsider Fig. 5, but with the dashed lines. Pulses are propagated as described in the proof of Theorem 3.11. In addition, each *ab*-boundary sends a pulse to the right at 1/2 unit speed checking for all *b*'s, and each *cB*-boundary sends a pulse left at 1/2 unit speed checking for all *c*'s. Should the *cB*-boundary pulse, if it exists, collide with an *ab*-boundary pulse at a *bc*-boundary, then the form $b^j c^j B$ is guaranteed. Furthermore, should it collide with the *Ba*-boundary pulse, which has determined the existence of the form $Ba^i b^i$, at the same *bc*-boundary, then the form $Ba^m b^m c^m B$ is guaranteed and an accept pulse is sent right at unit speed. Q.E.D.

LEMMA 3.14. $L_6 = \{a^m \mid m$ *is prime*$\}$ *is a real-time* DBCS *language.*

Proof. Let $s_1 s_2 \cdots s_i \cdots$ be the characteristic sequence of the primes—i.e., $s_i = 1$ if i is prime and $s_i = 0$ otherwise. Fischer [10] has shown that this sequence can be generated in real time by a deterministic iterative automaton, say V. Design Z to accept L_6 as follows: Let Z simulate V, as in Corollary 3.4.1, with the accept cell simulating the distinguished cell of V. Simultaneously, have the left boundary cell send a pulse to the right at unit speed checking for all *a*'s. If the simulated distinguished cell of V generates a 1 just as the *b*-pulse arrives from the left, then Z accepts. Q.E.D.

THEOREM 3.15. *There exist real-time* DBCS *languages which are not context-free. (Hence the intersection of the real-time* DBCS *languages and the context-free languages is a proper and nonempty subset of the former.)*

Proof. L_1, L_2, and L_3 of Lemma 3.5 are context-free, but L_4 of Lemma 3.12, L_5 of Lemma 3.13, and L_6 of Lemma 3.14 are not context-free. Q.E.D.

The following special class of real-time DBCS context-free languages has become a useful tool for studying two-dimensional pattern recognition [22].

THEOREM 3.16. *A Dyck language is a real-time* DBCS *language.*

Proof. Let D be the Dyck language on alphabet $X = \{a_1, a_2, ..., a_m, b_1, b_2, ..., b_m\}$. That is, if each a_i, $1 \leqslant i \leqslant m$, is one type of left parenthesis and if each b_i is the right parenthesis corresponding to a_i, then D is the set of all well-formed strings of parentheses in X. Consider Fig. 6 for designing DBCS Z to accept D in real time. Each

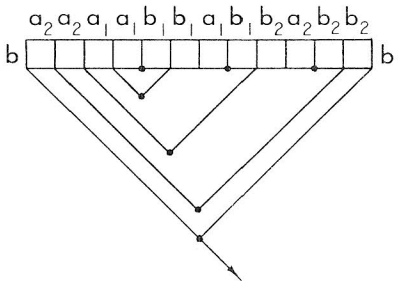

FIG. 6. Dyck language recognizer.

cell in Z is given two channels—i.e., each state is an ordered pair. Each left (right) parenthesis a_i (b_i) is propagated to the right (left) at unit speed via the left (right) channel. Should a left channel ever contain an a_i while the corresponding right channel contains b_i, then the left and right parentheses "cancel out."

The boundary cells also propagate as b-pulses to the right and left at unit speed. Should a left channel ever contain b while the corresponding right channel contains a b_i, then the right parenthesis has not been cancelled by a left parenthesis and never will be. Hence a reject signal is created which propagates right at unit speed. Similarly, should a right channel contain b while the corresponding left channel contains an a_i, then a reject signal is created and propagated right at unit speed. Should two boundary pulses collide without having previously created a reject signal, then an accept state is propagated right at unit speed. Q.E.D.

The next section treats the closure properties of real-time DBCS languages. In particular, they are shown to be closed under intersection. Hence, by Theorems 3.10(1)

and 3.16, the intersection of any regular set and Dyck language is a real-time DBCS language. Closure under homomorphism, however, is an open problem.

4. Closure Properties and Transformation Lemmas

THEOREM 4.1. *The class of real-time* DBCS *languages is closed under intersection and complementation, hence under union and set difference.*

Proof. These closures are easily derived by techniques similar to those of Cole [9]. For example, if L_i and L_j are real-time DBCS languages, then DBCS Z is designed to accept $L_i \cup L_j$ in real time as follows: Each cell has two channels. One is used for accepting L_i and the other for L_j. An OR-gate in the accept cell causes Z to accept the union in real time if and only if at least one of the channels has accepted in real time. The elapse of real time is clocked by a pulse propagating from left to right at unit speed. The other closures are similarly derived. The details are left to the reader. Q.E.D.

COROLLARY 4.1.1. *A real-time* DBCS *language is a real-time* DBCS *predicate.*

The class of linear-time DBCS predicates is also closed under the same operations [4]. The next theorem states that the class of linear-time DBCS languages is closed under reversal. That is, if L is a linear-time DBCS language, then so is $L^R = \{x^R \mid x \in L\}$. One proof is simple: If Z accepts L, then build Z' as the "mirror image" of Z. Then the leftmost nonboundary cell of Z' simulates the accept cell of Z. When this cell in Z' simulates an accept state, a pulse is propagated right at unit speed to put the accept cell of Z' into an accept state.

Another proof is given below to illustrate a use of the following lemma, the first of three DBCS transformation lemmas. In these results, an entire pattern is the desired output, not just the state of one cell. An important example is the firing squad lemma. (See [19] for a proof that multiplication of two binary integers of total length n can be accomplished by a DBCS within time $n/2$.)

LEMMA 4.2. *There is a* DBCS *which, given initial pattern x of length n, can reverse x in linear time. In fact, $F^{2n}(x) = x^R$.*

Proof. Consider Fig. 7. The leftmost nonboundary cell sends a pulse containing its initial state to the right at unit speed. The right b-cell sends a b-pulse one cell left at the first step; it begins to propagate left at unit speed only when the pulse from the leftmost nonboundary cell collides with it. Meanwhile, every other cell in the array sends a pulse containing its initial state to the left at 1/2 unit speed. Each of these pulses is reflected by the left b-cell but at unit speed. The cell where a reflected pulse collides with the left-propagating b-pulse maintains the state carried by the reflected

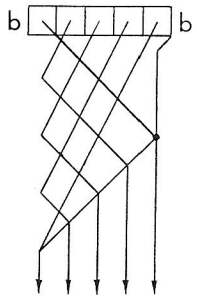

FIG. 7. Reversal in linear time.

pulse from the time of collision on. The collision of the right boundary pulse with the left boundary is the "computation complete" signal. Q.E.D.

THEOREM 4.3. *The class of linear-time* DBCS *languages is closed under reversal.*

Proof. If Z accepts L, then design Z' to accept L^R as follows: Z' reverses an input, as in Lemma 4.2, in linear time. The computation complete signal is used to initiate a firing squad. When it fires, in linear time, then Z' begins to simulate Z. Q.E.D.

The next lemma guarantees that a block of code can be shifted exactly m positions to the right in linear time, where m is the length of the code block. The notation X^m represents the set $\{x_1x_2 \cdots x_m \mid x_i \in X, 1 \leqslant i \leqslant m\}$.

LEMMA 4.4. *For each integer constant* $c > 0$, *there is a* DBCS $Z = (X, Q, \delta, b)$ *with quiescent state* 0 *and especially designated state* $@ \in X$ *such that* $F^{3m+c-3}(x@0^{n-m}) = ux@v$, *where* $x \in X_{@}^{m-1}$, $u \in Q_b^{m+c-1}$, *and* $n \geqslant 2m + c - 1$. ($X_{@} = X - @$.)

Proof. Consider Fig. 8, for which $c = 1$. Assume $x@ = q_1q_2 \cdots q_m$. The leftmost nonboundary cell sends a q_1-pulse to the right at unit speed. At the first step, the @-cell sends an @-pulse c cells to the right, where it maintains position until the q_1-pulse collides with it. At this time the @-pulse is sent to the right at 1/2 unit speed, and the cell at the collision site maintains state q_1 from then on. Meanwhile, each cell initially in state q_i, $2 \leqslant i \leqslant m$, has sent a q_i-pulse to the left at unit speed. These pulses, which are reflected by the left boundary at unit speed, propagate until colliding with the @-pulse propagating at 1/2 unit speed. The cell at which the collision of the q_i-pulse and @-pulse occurs maintains state q_i from the time of collision on. The computation complete signal is given by the collision of the two @-pulses. Q.E.D.

The final transformation generalizes the "center-finder" technique used in several of the preceding proofs.

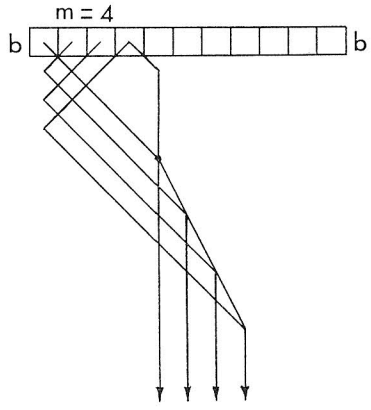

FIG. 8. Translation in linear time.

LEMMA 4.5. *Let p and q be any two positive integers. Then there is a* DBCS Z *which places a special marker state @ at the k-th cell, $k = \lceil q/(p+q)\rceil n$, of an input of length n within n time steps—i.e., there exists $t \leqslant n$ such that*

$$F^t(q_1q_2 \cdots q_{i-1}q_iq_{i+1} \cdots q_n) = (q_1q_2 \cdots q_{i-1}@q_{i+1} \cdots q_n) \qquad \textit{for} \quad i = \lceil q/(p+q)\rceil.$$

Proof. The left boundary cell sends a b-pulse L right at $1/p$ unit speed. The right boundary cell sends a b-pulse R left at $1/q$ unit speed. Let i be the position at which the two pulses collide. L requires pi steps to reach i. R requires $(n-i)q$ steps to reach i. But these times must be equal; hence $pi = (n-i)q$ and the lemma. Q.E.D.

5. DISCUSSION AND OPEN PROBLEMS

Pattern recognition with cellular automata has been approached via formal language theory by proving that any pattern set accepted by a cellular automaton must be a context-sensitive language. A special interest in time and memory requirements led to the introduction and study of the real-time DBCS languages, those languages accepted in real time by deterministic cellular automata which are bounded to use only "real memory"—i.e., only the memory of those cells to which an input is presented. Below is a list of several interesting butas yet open problems: (1) Are the context-free languages a subset of the real-time DBCS languages? (2) Are the real-time DBCS languages closed under concatenation and reversal? The real-time iterative acceptor languages are not [9]. (3) Do there exist nonlinear DBCS predicates—i.e., DBCS languages which require nonlinear recognition times [4]? The answer is yes for iterative acceptors [17].

Acknowledgment

I thank Dr. Michael Harrison of Berkeley and Albert Meyer of M.I.T. for suggesting the quasirealtime languages to me as a method for proving Corollary 3.8.1(1).

I also thank Dr. Stephen Hedetniemi of the University of Iowa for a careful reading of an early draft of this paper and for referring me to the paper of Kasami and Fujii [15].

References

1. M. A. Arbib, "Theories of Abstract Automata," Prentice-Hall, Inc., Englewood Cliffs, N. J., 1969.
2. E. R. Banks, "Information processing and transmission in cellular automata," Project MAC Report MAC TR 81, M.I.T., Cambridge, Mass., January, 1971.
3. Y. M. Barzdin, Complexity of recognition of symmetry in Turing machines, *Problemy Kibernetiki* **15** (1965), 245–248.
4. W. T. Beyer, "Recognition of topological invariants by arrays," Project MAC Report MAC TR-66, M.I.T., Cambridge, Mass., October, 1969.
5. R. V. Book and S. A. Greibach, Quasi-realtime languages, *Math. Systems Theory* **4** (1970), 97–111.
6. A. W. Burks, Ed., "Essays on Cellular Automata," University of Illinois Press, Urbana, Ill., 1970.
7. E. F. Codd, "Cellular Automata," Academic Press, New York, 1968.
8. S. N. Cole, "Real-time computation by iterative arrays of finite-state machines," Ph.D. Dissertation, Harvard University, 1964.
9. S. N. Cole, Real-time computation by n-dimensional iterative arrays of finite-state machines, *IEEE Trans. Computers* **C-18** (1969), 349–365.
10. P. C. Fischer, Generation of primes by a one-dimensional real-time iterative array, *J. Assoc. Comput. Mach.* **12** (1965), 388–394.
11. M. Gardner, On cellular automata, self-reproduction, the Garden of Eden and the game "life," Mathematical Games Department, *Sci. Amer.* **224** (1971), 112–117.
12. J. Hartmanis and R. E. Stearns, On the computational complexity of algorithms, *Trans. Amer. Math. Soc.* **117** (1965), 285–306.
13. F. C. Hennie, "Iterative Arrays of Logical Circuits," M.I.T. Press, Cambridge, Mass., 1961.
14. J. E. Hopcroft and J. D. Ullman, "Formal Languages and Their Relation to Automata," Addison-Wesley, Reading, Mass., 1969.
15. T. Kasami and M. Fujii, Some results on capabilities of one-dimensional iterative logical networks, *Electronics and Commun. Japan* **51-C** (1968), 167–176.
16. S. Y. Kuroda, Classes of languages and linear-bounded automata, *Information and Control* **7** (1964), 207–223.
17. S. R. Kosaraju, "Computations on iterative automata," Ph.D. Dissertation, University of Pennsylvania, August, 1969.
18. F. R. Moore and G. C. Langdon, A generalized firing squad problem, *Information and Control* **12** (1968), 212–220.
19. A. R. Smith III, "Cellular automata theory," Technical Report No. 2, Digital Systems Laboratory, Stanford University, Stanford, California, January, 1970.
20. A. R. Smith III, Cellular automata and formal languages, *in* "Proceedings of Eleventh Annual IEEE Symposium on Switching and Automata Theory," pp. 216–224, Santa Monica, California, 1970.

21. A. R. SMITH III, Cellular automata complexity trade-offs, *Information and Control* **18** (1971), 466–482.
22. A. R. SMITH III, Two-dimensional formal languages and pattern recognition by cellular automata, *in* "Proceedings of Twelfth Annual IEEE Symposium on Switching and Automata Theory," pp. 144–152, East Lansing, Mich., 1971.
23. V. I. VARSHAVSKY, V. B. MARAKHOVSKY, AND V. A. PECHANSKY, Synchronization of interacting automata, *Math. Systems Theory* **4** (1970), 212–230.
24. A. WAKSMAN, An optimum solution to the firing squad synchronization problem, *Information and Control* **9** (1966), 66–78.
25. H. YAMADA AND S. AMOROSO, Tessellation automata, *Information and Control* **14** (1969), 299–317.

Printed by the St Catherine Press Ltd., Tempelhof 37, Bruges, Belgium.