Formally published (with corrections) as *Real-Time Language Recognition by One-Dimensional Cellular Automata*

FOCS = *IEEE Foundations of Computer Science Conference Record*

FOCS 11 (1970) 216-224

CELLULAR AUTOMATA AND FORMAL LANGUAGES

Alvy Ray Smith III
New York University
Bronx, New York 10453

Summary

A set of equivalences is established among cellular automata, iterative acceptors, and linear-bounded automata. However, cellular automata are shown to be inherently faster than iterative acceptors. Many positive results are presented to indicate that the context-free languages can, perhaps, be accepted in time n and space n by cellular automata.

Introduction

A cellular automaton[1] is an array of identical finite-state Moore machines which are uniformly interconnected. It is autonomous but for a spatial pattern of inputs at time t = 0. 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 the sequence is the image of a function of the preceding configuration, with only a single such function associated with each cellular automaton. Thus cellular automata are a subclass of the so-called tessellation automata[2] which allow a set of such functions.

A closely related class of devices shall here be called iterative automata.[3] An iterative automaton is also a uniform array of identical machines but is not autonomous, receiving a temporal input pattern at one specially designated input-output machine. The temporal sequence of outputs generated by this one machine is the object of interest. Thus, in a sense, iterative automata are to cellular automata what an on-line Turing machine is to an off-line Turing machine.

A cellular automaton is intuitively a pattern-receiving "retina", especially in the two-dimensional case. Hence the pattern recognition capabilities of cellular automata is particularly interesting. Initial work[4] on this problem has shown that these devices can recognize a wide variety of topological invariants, including connectivity, in linear time (time proportional to the perimeter of the given pattern). This paper represents an extension of this work obtained by restricting the classes of patterns studied to those called formal languages.

Definitions

The abbreviation n-D is used to mean n-dimensional, for n a positive integer. We shall usually consider the 1-D case except where otherwise indicated.

One may envision a 1-D cellular space, or cellular automaton, as an infinite strip of film, each frame of which represents a copy of a single finite-state machine. Each machine, or cell, operates in synchrony with all others in a given cellular space, and time is assumed to proceed in discrete time steps. Associated with each cell is a local transition function δ which obtains the next state of the cell as a function not only of the present state of the cell but also as a function of the present states of a specified set of neighboring cells, or neighbors, in its neighborhood. It can be shown[1] that a 3-cell neighborhood--a cell and its left and right neighbors--always suffices† in the 1-D case; hence we assume this neighborhood.

If Q is the state set of each cell, then the input set is QxQ. That is, the output of a cell is taken to be its state, and this output is used as input to the two nearest neighbors. Hence $\delta: Q^3 \to Q: (x,y,z) \mapsto y'$ is the local transition function for a cell in state y with left neighbor in state x and right neighbor in state z. Note that a global transition function Δ can be defined as the simultaneous invocation of δ at each cell. Thus Δ maps a configuration--i.e., an assignment of states to each cell in a cellular space--into another configuration. There is a special state $q_o \in Q$, called the quiescent state, such that $\delta(q_o,q_o,q_o) = q_o$.

We shall frequently restrict our study to "bounded" cellular spaces. Thus, although a cellular space is infinite in extent, we can force it to act finitely by setting two cells initially to a special boundary state b. Then δ is restricted in such a way that no information can flow through the cells in state b and so that the cells in state b never change from that state. Hence the finite portion of the cellular space between the two boundary cells acts as a finite cellular space. But if we desire to process a larger pattern--i.e., the finite portion of a configuration between the two boundary cells--then we simply set the boundary cells farther apart. The following definitions are intended to capture these ideas.

Definition 1. A deterministic bounded cellular space (DBCS) is a 1-D cellular space--denoted by the 4-tuple (X,Q,δ,b)--with the 3-cell nearest-neighbor neighborhood, state set Q, and deterministic local transition function $\delta: Q^3 \to Q$ restricted as follows:

(1) $b \in Q$ is a special boundary state;
(2) $X \subset Q_b = Q-\{b\}$ is the initial alphabet;
(3) $\delta(q_i,b,q_j) = b$ for arbitrary $q_i, q_j \in Q$;
(4) two and only two cells, the boundary cells, are in state b at time t = 0.

Definition 2. The pattern transition function for a DBCS $Z = (X,Q,\delta,b)$ is the function $F: Q_b^* \to Q_b^*$ 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 $F(\Lambda) = \Lambda$, the empty string, where n is the number of cells in Z between the boundary cells.

†In fact, for an unbounded 1-D cellular space, a 2-cell neighborhood is always sufficient.[1]

Definition 3. An element of Q_b^* is said to be a pattern for DBCS $Z = (X,Q,\delta,b)$. Let $R: Q_b^* \to Q$ be the extraction function which extracts the rightmost element of a finite pattern: $R(q_1q_2\ldots q_n) = q_n$ and $R(\Lambda) = b$.

We shall be interested in bounded cellular spaces as language acceptors in the following sense:

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

A 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 whether (or not) $x \in L$ 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 DBCS language if there is a DBCS acceptor $Z = (X,Q,\delta,b,A)$ such that $L = L(Z) \equiv \{x \in X^* \mid (\exists t)[R(F^t(x)) \in A]\}$. Similarly, L is a DBCS predicate if it is recognized by some DBCS recognizer. A real-time DBCS language (predicate) is a DBCS language (predicate) which is accepted (recognized) within $T(n) = n$. Similarly, the adjective linear-time implies $T(n) = cn$.

Thus a string is accepted if, when embedded between two boundary cells in some DBCS acceptor, action of the pattern transition function causes the rightmost cell to eventually pass into an accept state.

The concepts above are readily generalized to the nondeterministic case in the obvious way: A nondeterministic bounded cellular space is defined just as is a DBCS with the exception that $\delta: Q^3 \to 2^Q$ is a nondeterministic local transition function with the restriction that $\delta(q_i,b,q_j) = \{b\}$, for arbitrary $q_i,q_j \in Q$. A language is accepted in this case if at some time t it is possible for the rightmost pattern cell to enter an accept state.

An n-D iterative automaton is an n-D cellular automaton with a distinguished cell which has a local transition function augmented to be a function of an external input also. A string is said to be accepted by an iterative automaton if the distinguished cell ultimately goes into an accept state and emits a corresponding output. An iterative automaton used in this accepting mode is called an iterative acceptor. A real-time iterative acceptor accepts within time $T(n) = n$. A real-space iterative acceptor uses no other cells than does a real-time iterative acceptor. That is, for the 1-D case, only the distinguished cell, the n cells immediately to its right, and the n cells immediately to its left can ever change state. Hence a real-time iterative acceptor is a special case of a real-space iterative acceptor. An iterative acceptor is nondeterministic (deterministic) if its local transition function, including that of the distinguished cell, is nondeterministic (deterministic)--just as for cellular automata.

A third class of machines with which we shall be concerned are the linear-bounded automata. We assume the reader is familiar with these devices, especially as defined in [5]. Two classes of machines are said to be equivalent if they accept the same class of languages.

For symbol conventions, a brief review of the familiar context-free languages follows: A context-free grammar G is a 4-tuple (V_N,V_T,S,P) with finite non-terminal alphabet V_N, finite terminal alphabet V_T disjoint from V_N, $S \in V_N$ the starting symbol, and P a finite set of productions of the form $\alpha \to \beta$, where for $V = V_N \cup V_T$, $\alpha \in V$ and $\beta \in V^*$. For $w_1,w_2 \in V^*$ and production $\alpha \to \beta$ we write $w_1\alpha w_2 \Rightarrow w_1\beta w_2$ and take $\Rightarrow^*$ to be the transitive closure of $\Rightarrow$. The language $L(G)$ generated by G is given by $L(G) = \{w \mid S \Rightarrow^* w \in V_T^*\}$ and is called a context-free language. A linear context-free language is generated by a linear context-free grammar--i.e., a context-free grammar for which every production is of the form $Y \to a$, $Y \to Xa$, or $Y \to aX$ for $a \in V_T$ and $X,Y \in V_N$. A general knowledge of context-free language theory and terminology is assumed--see, for example, Hopcroft and Ullman,[12] for further details.

We shall make use of the following easily proved theorem [4] (see also [1] and [3]):

The Speed-Up Theorem (for DBCS). Let k be an arbitrary positive integer. For an arbitrary DBCS acceptor $Z = (X,Q,\delta,b,A)$ with $|Q| = r$, there exists a DBCS acceptor $Z' = (X,Q',\delta',b,A)$ with $|Q'| = 8r^k$ such that if Z accepts within time T(n) then Z' accepts within time $(T(n)/k)+n$.

In particular, if Z accepts L within linear time $T(n) = cn$, then there is a Z' which accepts L within $T(n) = (1+\varepsilon)n, \varepsilon > 0$.

Equivalence and Complexity Results

Theorem 1. The class of n-D nondeterministic (deterministic) bounded cellular spaces is equivalent to the class of n-D nondeterministic (deterministic) linear-bounded automata.

Proof. The proof is straightforward, except for one small subtlety in the nondeterministic case, but tedious. Hence only a sketch of the proof technique is presented with emphasis on this subtlety. Furthermore, only the 1-D case will be considered, the generalization to the n-D case being straightforward. See[4] for a proof of this theorem for the 2-D deterministic case.

First, let L be the language accepted by linear-bounded automaton M. Then construct bounded cellular space acceptor Z such that $L = L(Z)$ as follows. Let each cell of Z simulate one square of the tape of M. Thus the boundary cells in Z simulate the endmarkers on the tape of M. Let each cell of Z also be capable of simulating the state behavior of M but not the actual moves, of course. Thus

only one cell at a time will ever be simulating M. A move of M is simulated by one cell's ceasing to simulate M while one of its neighbors commences the simulation.

The simulation is effected by giving each state of a cell C in Z two coordinates, say (α,β). Here α will represent the state of M if C is simulating M else it will be some special state - (hyphen) when C is not simulating M. The other coordinate β will represent one tape symbol of M. It represents the scanned symbol only if $\alpha \neq -$.

Let any cell with $\alpha \neq -$ be called the head cell. Thus our simulation will ensure that only one cell at any one time is the head cell. Notice that, in the nondeterministic case, the head cell and its two nearest neighbors are all possible candidates for position of head cell after a one step simulation of M. To ensure that only one of the three will become head cell, we break the simulation of one step of M into two steps of Z. During the first step, Z will nondeterministically simulate the nondeterministic behavior of M. During the second step, Z will deterministically simulate the move of M. Clearly, in the deterministic case, one step of Z suffices to simulate one step of M. The interested reader can readily fill in the remaining details.

Conversely, let L' be the language accepted by a bounded cellular space Z'. Then construct a linear-bounded automaton M' such that L' is the language accepted by M' as follows. In contrast to the simulation above which is real-time (or twice real-time), this simulation will require n steps for each step of Z' for n the length of the input pattern.

The input to Z' is recorded on the tape of M' one state symbol per tape square. Then M' makes one complete pass along its tape for each step of Z', "dancing" about each square to obtain neighborhood information for updating that square before proceeding to the next square. Again we leave the details to the interested reader, there being no difficulty involved in this case in the generalization to the nondeterministic mode.

Hence pattern recognition by cellular arrays 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 proofs below assume the 1-D case, the n-D generalizations being straightforward.

Lemma 2. The class of languages accepted by n-D nondeterministic (deterministic) bounded cellular spaces is accepted by nondeterministic (deterministic) n-D real-space iterative acceptors.

Proof. A bounded cellular space can be simulated--in its language accepting mode--by an iterative acceptor as follows: An initial string for the cellular space is input temporally into the distinguished cell of the simulating iterative acceptor, leftmost symbol first. Each symbol is shifted left until the entire input string is arrayed spatially in the cells of the iterative acceptor. A signal initiated by the final input symbol must be propagated left to halt the shifting of the string to the left. This entire input operation can be accomplished in time 2n for a string of length n. Then a firing squad[6] is simulated. When it "fires", the cells of the iterative acceptor simultaneously begin functioning exactly like the cells of the simulated cellular space. If the string is accepted by the cellular space, then the rightmost cell accepts. But by the input procedure, this cell is the distinguished cell of the iterative acceptor.

Lemma 3. For an arbitrary n-D nondeterministic (deterministic) iterative automaton, there exists an n-D nondeterministic (deterministic) cellular space which simulates it in real time.

Proof. The input string to the simulated iterative automaton is embedded one symbol per cell in the simulating cellular space. The first temporal input symbol is assumed to be the rightmost symbol in the spatial array of the input. The two cells at either end are placed in "endmarker" states. The right endmarker 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 into the right endmarker 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 can be considered to 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 out of the cell simulating the distinguished cell.

Theorem 4. The class of n-D nondeterministic (deterministic) bounded cellular spaces is equivalent to the class of n-D nondeterministic (deterministic) real-space iterative acceptors.

Proof. We need only prove the converse of Lemma 2, but only slight modification of the proof of Lemma 3 is required for this goal. Clearly, Lemma 3 is true for iterative acceptors as a special case of iterative automata. Specifically, the left and right endmarker 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.

Corollary 4.1. If an arbitrary n-D deterministic iterative acceptor accepts language L within time T(n), then there is an n-D DBCS which accepts L within time T(n).

Corollary 4.2. The context-sensitive (deterministic context-sensitive) languages are precisely the languages accepted by nondeterministic (deterministic) bounded cellular spaces and real-space iterative acceptors.

Corollary 4.3. Any context-free language is accepted by some 2-D DBCS within time $T(n) = (1+\varepsilon)n, \varepsilon>0$.

Proof. Kosaraju[7] proves that a 2-D deterministic iterative acceptor exists for each context-free language, which accepts within time $T(n) = 6n$ and hence within time $T(n) = (1+\varepsilon)n$ by the speed-up theorem for iterative acceptors.[3] (The straightforward adaptation of the Kosaraju proof to DBCS drops the time requirement from $T(n) = 6n$ to $T(n) = 3n/2$, but this is still, unfortunately, greater than real time.)

The next theorem makes explicit the difference in computing power between cellular spaces and iterative acceptors. It says that the converse of Corollary 4.1 is false. But first we need a lemma.

A palindrome in X^* is a word of the form ww^R, where $w \in X^*$ and w^R is the string w written in reverse order.

Lemma 5. $L_1 = \{ww^R | w \in X^*\}$ is a real-time DBCS language; so is $L_2 = L_1X^*$ and $L_3 = X^*L_1$.

Proof. Consider the space-time diagram of Fig. 1. This is a convenient heuristic device for designing a DBCS (e.g., see [6,10]) in the sense that once one can draw a satisfactory diagram, he can implement it by an appropriate local transition function. This implementation is a tedious task which we shall typically leave to the reader.

Each cell sends its state to the left and right at unit speed (one cell per step). 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) of a palindrome; Should two pulses carrying different states ever collide at the cell in position i then that cell goes into a special state, say 0, signifying that position i cannot be the center cell of a palindrome, and it remains in that state. The pulse sent left by the right boundary cell acts as a "collection" pulse, which "reflects" from the center cell of the pattern to the right boundary. Suppose we require that this pulse send an accept state to the right if and only if it finds a non-0 center cell at the center of the given pattern. Then we have designed Z_1 such that $L_1 = L(Z_1)$ is accepted within real time. But suppose we require the collection pulse to send an accept state to the right if and only if it finds at least one non-0 cell before, or as, it collides with the center cell of the pattern. Then we have designed real-time acceptor Z_3 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 we have designed Z_2, the real-time acceptor of L_2.

Theorem 6. There exists a (context-free) language, not accepted within real time by any deterministic iterative acceptor, which is a real-time DBCS language. (Hence deterministic bounded cellular automata are inherently faster than deterministic real-time iterative acceptors.)

Proof. Consider the language X^*L_1 of Lemma 5. Cole[3] has shown that no deterministic real-time iterative acceptor of any dimension can accept X^*L_1.

This result suggests the following interesting, and as yet unsolved, problem: Are the context-free languages a subset of the real-time DBCS languages? The remainder of this paper is devoted to partial solution of this problem. For example, the next theorem answers the question in the affirmative for linear context-free languages. The proof is based on the Kasami adaptation[8] of the Younger algorithm,[9] reviewed below:

Consider a linear context-free grammar G with $a_i \in V_T, 1 \le i \le n$, and $X, Y \in V_N$. If $Y \Rightarrow^* a_i \ldots a_j, 1 \le i \le j \le n$, then there is an X such that $X \Rightarrow^* a_i \ldots a_{j-1}$ and $Y \to Xa_j$ is a production or $X \Rightarrow^* a_{i+1} \ldots a_j$ and $Y \to a_i X$ is a production. Define $N(i,i) \equiv \{A | A \in V_N$ and $A \to a_i \in P\}$, $1 \le i \le n$. Define $N(i,j) = N_1(i,j) \cup N_2(i,j)$ where $N_1(i,j) = \{Y | Y \to Xa_j \in P, X \in N(i,j-1)\}$ and $N_2(i,j) = \{Y | Y \to a_i X \in P, X \in N(i+1,j)\}$. Then $a_1a_2 \ldots a_n \in L(G)$ if and only if $S \in N(1,n)$.

Theorem 7. Any linear context-free language is a real-time DBCS language.

Proof. Let $G = (V_N, V_T, S, P)$ be a linear context-free grammar with $V = V_N \cup V_T$. Then design DBCS acceptor $Z = (V_T, Q, \delta, b)$ such that Z accepts L(G) in real time by the following scheme: At time t = 0 a candidate $a_1a_2 \ldots a_n \in V_T^*$ is embedded one symbol per cell between the boundary cells of Z. At t = 1 the i-th cell (i.e., the cell initially holding a_i) computes $N(i,i)$. At t = k the i-th cell computes $N(i-k+1,i)$ if $k \le i$; if $k > i$, the cell maintains the state it was in at t = k-1.

We must show that it is possible for the i-th cell in Z to calculate $N(i-k+1,i)$ at $t=k \le i$. At t=k-1, this cell computes $N(i-k+2,i)$, and its left neighbor computes $N(i-k+1,i-1)$. But by the algorithm above, this is sufficient information for computation of $N(i-k+1,i)$ at t = k if the i-th cell also contains in its memory a_i and the finite set P, two pieces of information easily provided.

Thus at time t = n, the rightmost non-boundary cell computes $N(1,n)$ and accepts or rejects.

Corollary 7.1. Any linear context-free language can be accepted in real time by a DBCS with only a two-cell neighborhood, consisting of a cell and one of its nearest neighbors.

Corollary 7.2. Any linear context-free language can be accepted in $T(n) = (1+\varepsilon)n, \varepsilon > 0$, by a 1-D deterministic real-space iterative acceptor.

Theorem 8. The class of real-time DBCS languages is closed under intersection and complementation, hence under union and set difference.

Proof. Let L_1 and L_2, real-time DBCS languages on alphabet X, be accepted in real time by DBCS Z_1 and Z_2 respectively. Design DBCS Z to accept $L_1 \cap \overline{L_2}$ in real time as follows: Each state of a cell in Z is an ordered pair. The first element of the pair simulates Z_1; the second simulates Z_2. The passage of real time is determined by propagating a "clock" pulse from the left boundary cell to the right boundary cell at unit speed. If and only if both elements of the state pair of the rightmost non-boundary cell are simulating accept states when the

clock pulse arrives, then Z accepts. (Once an element of the state pair of the rightmost cell goes into a simulated accept state, it is assumed to remain in that state.)

Design DBCS Z' to accept the complement L_1' of L_1 in real time as follows: Each cell of Z' simulates a cell of Z. Meanwhile a clock pulse is propagated from left to right to determine real time. If and only if the rightmost non-boundary cell is not simulating an accept state of Z when the clock pulse arrives at it, then Z' accepts.

The rest of the theorem follows from $L_1 \cup L_2 = (L_1' \cap L_2')'$ and $L_1 - L_2 = L_1 \cap L_2'$.

Corollary 8.1. 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 $L^R = \{x^R | x \in L\}$. One proof is simple: If Z accepts L, then build Z' as the "mirror image" of Z. Then the leftmost non-boundary cell of Z' simulates the rightmost non-boundary cell of Z. When this cell in Z' simulates an accept state, a pulse is propagated right at unit speed to put the rightmost non-boundary 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. (See[1] for a proof that multiplication of two binary integers of total length n can be accomplished by a DBCS within time n/2.)

Lemma 9. 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. 2: The leftmost non-boundary cell sends a pulse containing its initial state to the right at unit speed. The rightmost boundary cell sends a pulse one cell left at the first step; it begins to propagate left at unit speed only when the pulse from the leftmost non-boundary 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 boundary but at unit speed. The cell where a reflected pulse collides with the left-propagating boundary pulse maintains the state carried by the reflected pulse from the time of collision on. The collision of the right boundary pulse with the left boundary is the "computation complete" signal.

Theorem 10. 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 9, 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.

The next lemma guarantees that we can shift a block of code of length m exactly m positions to the right in linear time.

Lemma 11. Let $x^m y = q_1 q_2 \cdots q_n$ be a pattern of length $n \geq 2m$ with q_m a specially designated marker state. Then there is a DBCS $Z = (X, Q, \delta, b)$ with $q_i \in X$, $1 \leq i \leq n$, such that $F^{3m}(x^m y) = u x^m v, u \in Q_b^m$.

Proof. Consider Fig. 3. The leftmost non-boundary cell sends a pulse containing its state q_1 to the right at unit speed. At the first step, the marker cell (in state q_m) sends a right boundary pulse one cell to the right. This cell remains in the right boundary state until the pulse sent from the left collides with it, at which time it sends the right boundary pulse to the right at 1/2 unit speed and maintains state q_1 from then on. Meanwhile, each cell initially in state q_i, $2 \leq i \leq m$, has sent a "q_i pulse" containing its state q_i to the left at unit speed. (To ensure that all and only these cells act in this manner, a firing squad on the leftmost m pattern cells, using the marker state as an initiator (the "general"), can be used.) These pulses, which are reflected by the left boundary at unit speed, propagate until colliding with the right boundary pulse. The cell at which the collision for the q_i pulse occurs maintains state q_i from the time of collision on. The computation complete signal is given by the collision of the q_m pulse and the right boundary pulse. Time required is 3m.

Lemma 12. 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, $k = \left\lceil \frac{q}{p+q} \right\rceil n$, of an input of length n within n time steps--i.e., there exists $t \leq n$ such that $F^t(q_1 q_2 \cdots q_{i-1} q_i q_{i+1} \cdots q_n) = (q_1 q_2 \cdots q_{i-1} \#\# q_{i+1} \cdots q_n)$ for $i = \left\lceil \frac{q}{p+q} \right\rceil$.†

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

Lemma 13. $L_4 = \{ww | x \in X^*\}$ is a real-time DBCS language.

Proof. Cole[3] has demonstrated a 1-D iterative acceptor which accepts L_4 in real time. The lemma follows from Corollary 4.1.

Lemma 14. $L_5 = \{a^m b^m c^m | m \geq 1\}$ is a real-time DBCS language.

Proof. Consider Fig. 4. To erase confusion, let B be the boundary state in this case. At the first step, each ab boundary, bc boundary, Ba boundary, and cB boundary is specially marked. Each ab boundary sends a pulse to the right at 1/2 unit speed checking for all b's. Each bc boundary sends

† $\lceil x \rceil$ is the integer just larger than or equal to x.

a pulse left at unit speed checking for all b's. The Ba boundary sends a pulse right at unit speed checking for all a's. Should it collide with a bc boundary pulse at an ab boundary, then form Ba^ib^i is guaranteed. The cB boundary sends a pulse left at 1/2 unit speed checking for all c's. Should it collide with an ab boundary pulse at a bc boundary, then form b^jc^jB is guaranteed. Furthermore, should it also collide with the Ba boundary pulse, which has determined the existence of form Ba^ib^i, at the same bc boundary then form $Ba^mb^mc^mB$ is guaranteed and an accept pulse is sent right.

Lemma 15. $L_6 = \{a^n | n \text{ is prime}\}$ is a real-time DBCS language.

Proof. Let $s_1 s_2 \ldots s_i \ldots$ 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 1-D deterministic iterative automaton, say M. Design Z to accept L_6 as follows: Let Z simulate M, as in Corollary 4.1, with the rightmost non-boundary cell simulating the distinguished cell of M. 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 M generates a 1 just as the b pulse arrives from the left, then Z accepts.

Theorem 16. 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 non-empty subset of the former.)

Proof. L_1, L_2, and L_3 of Lemma 5 are context-free, but L_4 of Lemma 13, L_5 of Lemma 14, and L_6 of Lemma 15 are not context-free.

Theorem 17. A Dyck language is a real-time DBCS language.

Proof. Let D be the Dyck language on alphabet $X = \{a_1, a_2, \ldots, a_m, b_1, b_2, \ldots, b_m\}$. That is, if each a_i, $1 \le i \le m$, is one type of left parenthesis and if each b_j, $1 \le j \le m$, is the type of right parenthesis corresponding to a_j, then D is the set of all well-formed strings of parentheses in X. Consider Fig. 5 in order to design DBCS Z to accept D in real time. Each cell in Z is given two channels--i.e., each state is an ordered pair. Each left (right) parenthesis a_i (b_j) 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 a b_j with $i=j$, then the left and right parentheses "cancel out".

The boundary cells also propagate the boundary state b right and left at unit speed. Should a left channel ever contain b while the corresponding right channel contains a b_j, 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. 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. Should two boundary states collide without having previously created a reject signal, then an accept state is propagated right at unit speed.

Since any regular language is linear context-free, Theorems 7, 8, and 17 ensure that the intersection of any Dyck language and any regular language is a real-time DBCS language. Any context-free language is the homomorphic image of some such intersection but this approach to the proof that context-free languages are real-time DBCS languages has not yet borne fruit. So far we have only the following pieces of information to add to those already generated, none of which cast out the possibility.

Theorem 18. There exist inherently ambiguous context-free languages which are real-time DBCS languages. Hence there exist nondeterministic context-free languages which are real-time DBCS languages.

Proof. $L = \{a^ib^jc^k | i=j \text{ or } j=k\}$ is an inherently ambiguous context-free language. L is also a real-time DBCS language as we show by constructing real-time DBCS acceptor Z such that $L = L(Z)$. Consider Fig. 6 which illustrates a DBCS Z_1 for accepting $\{a^ib^jc^k | j=k\}$. To avoid confusion let B be the boundary state here. The Ba, ab, bc, and cB boundaries are specially marked. 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^jc^jB is guaranteed. The ab boundary pulse carries this information to the rightmost non-boundary cell. Meanwhile, each cell initially in state a sends a pulse right at unit speed. The collision of the ab boundary pulse with the rightmost non-boundary cell causes that 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.

We construct Z_2 to accept $\{a^ib^jc^k | i=j\}$ by a simple modification of the DBCS illustrated in Fig. 4. Eliminate the ab boundary pulse traveling right at 1/2 unit speed and the cB boundary pulse traveling left at 1/2 unit speed. Once the Ba and bc pulses collide at an ab boundary, having encountered only a's and b's respectively, the Ba 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.

Use Theorem 8 to construct Z such that $L(Z) = L(Z_1) \cup L(Z_2)$.

Theorem 19. There exist context-free languages not recognizable by any multitape Turing machine in real time which are real-time DBCS languages.

Proof. Hartmanis and Stearns[11] have shown that the language $L = \{yxsy'x^R | x \in \{0,1\}^* \text{ and } y,y' \in \{\Lambda\} \cup \{0,1,s\}^*\{s\}$ cannot be recognized by a multitape Turing machine in real time. We now illustrate a DBCS Z which accepts L in real time. Consider Fig. 7.

Each cell sends a "q_i pulse" containing its initial state q_i right and left at unit speed. Should two s pulses collide at cell C, then C begins to act as if it were the center cell of a palindrome as in the proof of Lemma 5. Should an s pulse propagate to or through C, then C ceases to check for palindromes. Should an s pulse (or b pulse) collide with the right b pulse at cell C while C is checking for palindromes, then the b pulse checks for a non-0 at C (i.e., for existence of a palindrome). One such non-0 before or at the collision of the left and right b pulses is sufficient for propagation of an accept state to the right.

Another possibility which must be accounted for is an input of form xsx^R. Have each s cell also act as a center cell of a palindrome. Should an s pulse ever propagate to or through such a cell, then it ceases checking for palindromes. However, should an s 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-0 just as above.

Discussion and Open Problems

We have approached pattern recognition with cellular automata 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 us to introduce and study the real-time DBCS languages, those languages accepted in real time by deterministic cellular automata which are bound 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 but as yet unsolved problems.

(1) Are the context-free languages a subset of the real-time DBCS languages?

We have shown only partial answers to this question. For example, the linear context-free languages are real-time DBCS languages.

(2) Are the real-time DBCS languages closed under concatenation and reversal?

The answer to the general question is probably No. (It is for real-time iterative acceptor languages.[3]) The real-time DBCS languages have been shown closed under union, intersection, complementation, and set difference.

(3) Do there exist non-linear DBCS predicates --i.e., DBCS languages which require non-linear recognition times?

It seems that the answer to this very interesting problem should be Yes (as it is for iterative acceptors[7]) but attempts to use the diagonalization proof technique of, say Hartmanis and Stearns,[11] have all failed so far. The major difficulty appears to be the real memory requirement. This is essentially the problem first posed by Beyer.[4]

References

1. Smith, A.R., III, "Cellular Automata Theory", Technical Report No. 2, Digital Systems Laboratory, Stanford University, Stanford, California 94305, Jan. 1970.
2. Yamada, H., and Amoroso, S., "Tessellation Automata", Information and Control, 14, 299-317, 1969.
3. Cole, S.N., "Real-time Computation by n-Dimensional Iterative Arrays of Finite-State Machines", IEEE Transactions on Computers, Vol. C-18, 349-365, April 1969.
4. Beyer, T., "Recognition of Topological Invariants by Iterative Arrays", Ph.D. dissertation, MIT, 1970.
5. Kuroda, S.-Y., "Classes of Languages and Linear-Bounded Automata", Information and Control, 7, 207-223, 1964.
6. Waksman, A., "An Optimum Solution to the Firing Squad Synchronization Problem", Information and Control, Vol. 9, 66-78, 1966.
7. Kosaraju, S.R., "Computations on Iterative Automata", Ph.D. dissertation, University of Pennsylvania, August 1969.
8. Kasami, T., "A Note on Computing Time for Recognition of Languages Generated by Linear Grammars", Information and Control, 209-214, 1967.
9. Younger, D.H., "Recognition and Parsing of Context-Free Languages", Information and Control, 189-208, 1967.
10. Fisher, P.C., "Generation of Primes by a One-Dimensional Real-Time Iterative Array", JACM, Vol. 12, 388-394, July 1965.
11. Hartmanis, J., and Stearns, R.E., "On the Computational Complexity of Algorithms", Trans. Amer. Math. Soc., 117, 285-306, 1965.
12. Hopcroft, J.E., and Ullman, J.D., Formal Languages and Their Relation to Automata, Addison-Wesley, Reading, Mass., 1969.

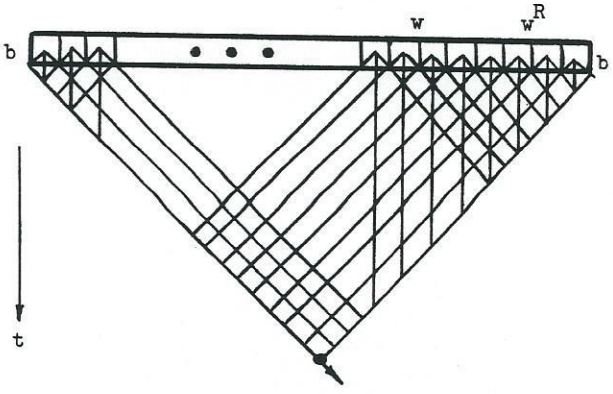

FIG. 1

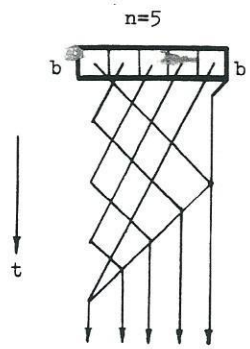

FIG. 2

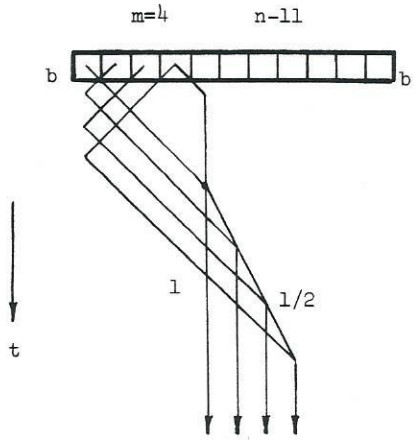

FIG. 3

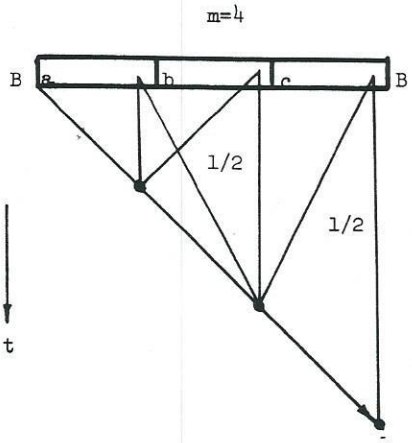

FIG. 4

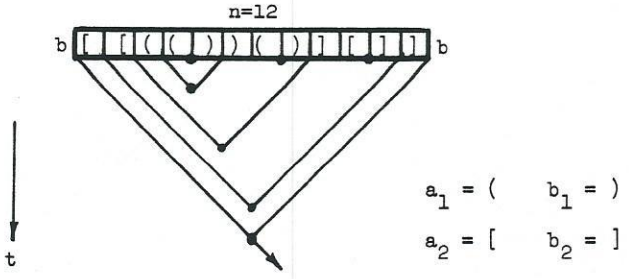

FIG. 5

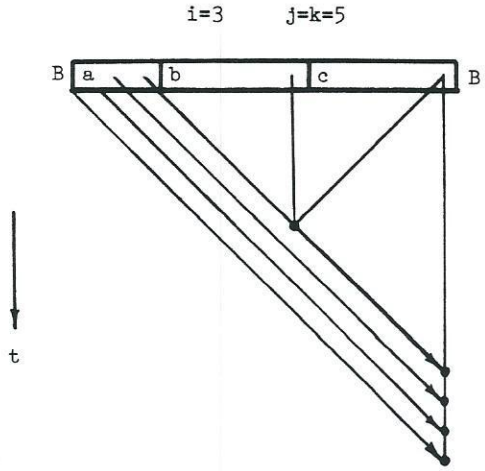

FIG. 6

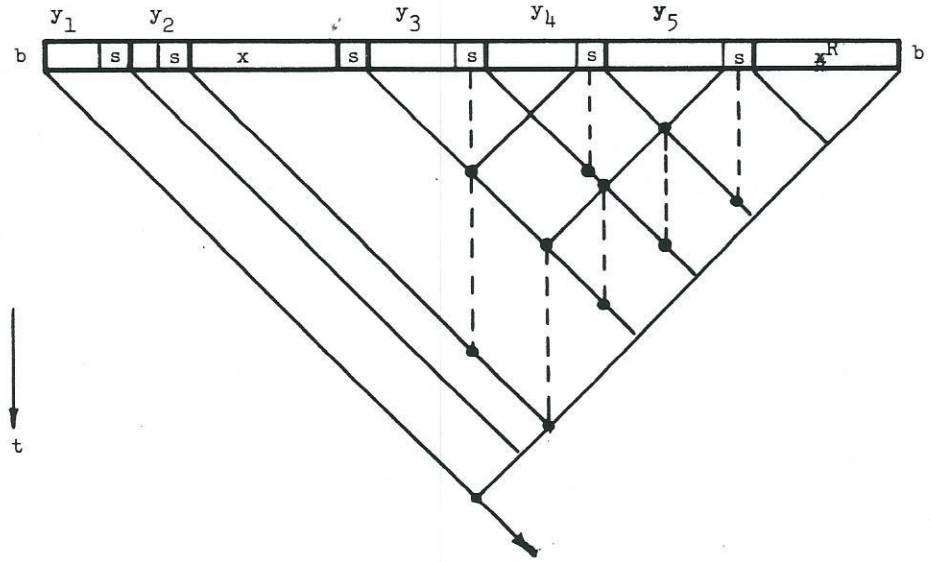

FIG. 7