FOCS 12 (1971)
144-152.

TWO-DIMENSIONAL FORMAL LANGUAGES AND PATTERN RECOGNITION BY CELLULAR AUTOMATA

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

Abstract

A formal study of pattern recognition capabilities of cellular automata is undertaken based on a class of recently introduced grammars for two dimensions, the array grammars, which can be thought of as the two-dimensional generalization of context-sensitive grammars. The class of languages (patterns) generated by array grammars is shown to be precisely the class of languages accepted by cellular automata forming rook-connected finite subsets of the plane. Thus the usual generalization to rectangular array-bounded cellular automata is a special case of this class of machines. The concept of perimeter time is introduced as a natural measure of computing speeds for two-dimensional cellular spaces, and connectedness and convexity are related to this measure. The class of patterns with positive Euler number is shown to be linear-time recognizable by rectangular array-bounded cellular automata, thus solving an open problem of Beyer.

Introduction

This paper represents an extension of the study[1] of one-dimensional (1-D) bounded cellular automata to the two-dimensional (2-D) case. A 2-D bounded cellular automaton--i.e., a finite array of identical finite-state machines called cells, each connected to its four nearest neighbors (except possibly at the boundaries)--is intuitively a highly parallel pattern recognition device resembling the retina. A powerful tool for investigating the pattern recognition capabilities of 1-D cellular automata is formal language theory, but until recently no such tool was available for the 2-D case except in rudimentary exercises.[2] Milgram and Rosenfeld[3] have successfully provided a foothold by introducing array grammars and array automata. Their results are used here to investigate 2-D bounded cellular automata.

Definition. An array grammar is a 5-tuple $G=(V,V_T,P,\#,S)$, where V is the vocabulary consisting of a finite set of symbols, $V_T \subset V$ is the set of terminals, $\# \in V-V_T$ is the blank symbol, $S \in V-V_T$ is the start symbol, and P is a finite set of structure-preserving productions of form $\alpha \to \beta$ defined as follows:

Let J be a finite connected† subset of I^2, for I the set of integers. Then α and β are mappings from J into V with the restriction that if $\alpha(i,j) = a \in V_T$ then $\beta(i,j) = a$ (i.e., terminals are never rewritten).

An array A is a mapping from I^2 into V. A production $\alpha \to \beta$ is applicable to A if there is a translation τ of the domain J of α such that $A|\tau J=\alpha$. Array A' is directly derivable from A, written $A \Rightarrow A'$, if for $\alpha \to \beta$ applicable to A, $A'|\tau J=\beta$ and $A'|(I^2-\tau J)=A|(I^2-\tau J)$. Let $\Rightarrow^*$ be the transitive closure of $\Rightarrow$. Then for $A \Rightarrow^* B$, B is said to be derivable from A.

An initial array A_S is a mapping from I^2 onto $\{\#,S\}$ such that $\{(i,j)|A_S(i,j) = S\}$ is a singleton--i.e., an initial array is all blanks but for one S. A terminal array A_T is a mapping from I^2 into $V_T \cup \{\#\}$ such that $\{(i,j)|A_T(i,j) \in V_T\}$ is connected. The array language generated by an array grammar G is $L(G)=\{B|A \Rightarrow^* B$, where A is initial and B is terminal$\}$. It can be shown that domains $\{(0,0)\}$, $\{(0,0),(0,1)\}$, and $\{(0,0),(0,-1)\}$ suffice for the productions of the class of array grammars. That is, for any array grammar G there is another grammar G' whose productions use only these three domains and such that $L(G) = L(G')$.

Finally, an array grammar G is said to be monotonic if it cannot erase--i.e., if for arbitrary production $\alpha \to \beta$, $\alpha(i,j) \neq \#$, then $\beta(i,j) \neq \#$. In this case, L(G) is called a monotonic array language.

Informally, a Turing array acceptor is a Turing machine (nondeterministic, in general) on a 2-D tape which can move one square up, down, right, or left (u,d,r,ℓ, respectively) at each step, and can read or write a symbol on any scanned square. An initial tape is assumed to have only a finite set of nonblank squares, and the acceptor is initially scanning one of these. Furthermore, the set of nonblank squares is assumed to be connected. The machine is called an array-bounded automaton (ABA) if it bounces off blanks--i.e., should it move in direction X from a nonblank symbol to a blank symbol at time t, then it moves in direction $X+180^\circ$ at time t+1 without having altered the blank. Turing array acceptors (hence ABA) accept by final state.

Theorem (Milgram-Rosenfeld). The class of languages accepted by array-bounded automata is precisely the class of languages generated by monotonic array grammars.

A 2-D cellular automaton (or, equivalently, cellular space) is an infinite array of finite-state machines (FSM), called cells, where each cell is assigned a point in I^2. In general, a cell is nondeterministic. The local transition function δ of a cellular space obtains the next state of a cell as a function of not only the present state of the cell itself but the present states of the four nearest neighbors of the cell, which form its neighborhood. This neighborhood is sometimes called the von Neumann, or H_1, neighborhood, where $H_k = \{(i,j) \in I^2 | |i|+|j| \le k\}$ defines a general class of neighborhoods. Similarly, define $J_k = \{(i,j) \in I^2 | |i| \le k, |j| \le k\}$. A common element of this latter class of neighborhoods is J_1, the so-called Moore neighborhood, consisting of a cell and its eight nearest diagonal and orthogonal neighbors. It can be shown[4] that there is no loss of generality in assuming the H_1 neighborhood. Since the J_1 neighborhood is sometimes more convenient, the following lemma is occasionally invoked in the sequel.

Lemma 1.[4] For an arbitrary cellular space Z with a J_1 neighborhood, there is a cellular space Z' with an H_1 neighborhood which simulates Z in two times real time--i.e., two steps of Z' simulate one step of Z.

A configuration in a cellular space Z is a mapping from I^2 into the state set Q of each cell in Z. Define the global transition function F of Z to be the simultaneous invocation of δ at each cell in the space. Then F maps configurations into configurations. There is a special state q_0 in Q, called the quiescent state, which has the following property: If a cell and all its neighbors are in q_0 (are quiescent) at time t, then that cell must remain in q_0 at time t+1. The support of a configuration c is given by $\sup(c)=\{(i,j)|c(i,j) \neq q_0\}$. An initial configuration (i.e., at time zero) is assumed

†Throughout this paper, connected will mean rook-connected--i.e., points (i,j) and (i',j') are connected if and only if $|i-i'| + |j-j'| \le 1$. A subset K of I^2 is connected if and only if, for any two points p and q in K, there is a sequence of points $p_1, p_2, \ldots, p_n$ in K with $p_1=p$, $p_n=q$, and p_i connected to p_{i+1}, $1 \le i < n$.

to have finite support.

Definition. A (2-D) array-bounded cellular space (BCS) is a cellular space for which each cell has a specially designated state B, called the boundary state, and a local transition function restricted to map a cell in state B into state B in all cases. Furthermore, no boundary states can be created after time zero. The boundary cells (cells in state B)[†] at time zero in a BCS are assumed to delineate a connected subset of cells, called a retina of the BCS. A simply-connected BCS (SBCS) is a BCS for which each retina is simply-connected. A rectangular BCS (RBCS) is an SBCS for which each retina is restricted to form a rectangle. The rightmost cell in the uppermost row of a retina in a BCS Z is called the accept cell for that retina in Z. Notice that, since no boundary cells can be created after t=0, a given retina in Z remains fixed in Z for all t>0. Hence we shall call the accept cell of a retina in Z the accept cell of Z.

A BCS Z with retina R is a pattern recognition device in the following sense: Let $X\subset Q-\{B\}$ be the initial alphabet of Z. Let X^* be the set of all connected words on alphabet X, where a connected word is a mapping from a finite, possibly empty, connected subset of I^2 into X. Assume that R is initially programmed with elements from X. Thus for initial configuration c_0, $c_0(R) = w\varepsilon X^*$. The word (cf. pattern) w is said to be accepted by Z (on A) if there is a time t such that $[F^t(c_0)](i_R,j_R)\cap A\neq\phi$, where $A\subset Q-\{B\}$ is disjoint from X and (i_R, j_R) is the accept cell of Z. A language $L\subset X^*$ is accepted by Z (on A) if, for arbitrary $w\varepsilon L$, w is accepted by Z on A. A language $L\subset X^*$ is a BCS language if there is a BCS with initial alphabet X which accepts precisely L. Similarly, a language L is recognized by a BCS Z if it is not only accepted by Z but its complement $L'=X^*-L$ is rejected by Z--i.e., L' is accepted on A', where $A'\cap A = \phi$. A BCS predicate is a language L for which there is a BCS which recognizes L. The prefix D shall be used to denote the deterministic special case in acronyms as, for example, DBCS.

The speed of recognition of patterns by BCS shall be a major concern. For RBCS, a natural measure on the retinas is m+n, where m and n are the dimensions of a retina. Hence linear time will imply a number of time steps proportional to m+n, and area time is a number of time steps proportional to mn for RBCS and to the number of points in the retina for arbitrary BCS. A time measure of particular interest here will be called perimeter time--i.e., a number of time steps proportional to the perimeter[††] of a given retina. At best, perimeter time is linear time. At worst, perimeter time is area time. Note that for RBCS, perimeter time is linear time. External perimeter time is a number of time steps proportional to only the external perimeter of a retina (i.e., additions to the perimeter due to holes in the retina are not included).

BCS Languages and Predicates

The purpose of this section is to show that the BCS languages are precisely the languages generated by monotonic array grammars. An ABA (DABA) with word $w\varepsilon X^*$ on its initial tape can simulate a BCS (DBCS) with initial retina configuration w by raster scanning the square on its tape, "dancing" about each square to gather neighborhood information, and updating each square just as would the BCS. The simulation is straightforward except for the necessity of skirting the holes in the array, but this can be accomplished by using the boundary of a hole as a pushdown stack to record net up and down moves.[3] Thus one step of the simulated BCS requires area time for simulation by the simulating ABA.

Conversely, it is straightforward to simulate an ABA (DABA) by a BCS (DBCS) in at most two times real time (i.e., two steps of the BCS simulate each step of the ABA).[1] An ABA accepts by final state whereas a BCS accepts by the state of a prespecified cell, the accept cell. Milgram and Rosenfeld have presented a method whereby an ABA on an arbitrary square of the array on its tape can find its way to a prespecified square of the array, say, the rightmost square in the topmost row, in perimeter time squared--i.e., in kp^2 time steps, where p is the perimeter of the nonblank portion of the array and k is a constant. A simple consequence is that if an ABA accepts a language L within time T, then there is a BCS which accepts L within time $2T + kp^2$. The next theorem brings the time limit down to 2T + kp. This is accomplished by showing, in a lemma below, that the parallelism of a BCS can be exploited to speed-up the finding of the accept cell from an arbitrary cell in the retina.

The proof of this lemma is quite lengthy but is interesting in that it makes extensive use of the 1-D firing squad result (see Waksman[5]) and of the 1-D Dyck language recognizing BCS of Smith.[1] Both of these results are intrinsically cellular automata theoretic, and hence take on the appearance, at least, of basic theorems for cellular automata theory.

Lemma 2. Given a DBCS Z, there is a special cell in each retina R of Z which can be uniquely identified within perimeter time. That is, for c_0 an initial configuration in Z entirely quiescent on R which has perimeter p, there is a special state \$ and a special cell $(i_R,j_R)\varepsilon R$, the accept cell, and a time t = kp, k constant, such that $[F^t(c_0)](i_R,j_R) = \$$ and such that $[F^{t'}(c_0)](i,j)\neq\$$ for all t' if $(i,j)\neq(i_R,j_R)$ or for all t'<t if $(i,j) = (i_R,j_R)$.

Proof. The technique is sketched in steps:

(1) All upper right cells (i.e., those having north and east neighbors in boundary state B) of R identify themselves by entering, say, state A. This requires one time step.

(2) All boundary-layer cells (i.e., those cells, which are not B-cells, having a B-cell in their J_1 neighborhood) identify themselves. This requires, by Lemma 1, at most two time steps, the first of which occurs during execution of step (1).

(3) Each A-cell sends a pulse along the boundary layer in which it lies (and which was marked in step (2)) in the clockwise (cw) direction (i.e., so that B is always to the left of the direction moved if one faces in that direction). The pulse simulates an FSM M which remembers each direction, u,d,r, or ℓ, it moves in one step and marks a cell by the direction it moved to get to the cell. If M does not mark a cell with a u while traveling from A-cell C_1 to A-cell C_2, then it can erase the state A from cell C_2 since C_2 cannot be as high or higher than C_1. M halts after marking an A-cell in which it cannot erase the A. Thus all cells in a boundary layer containing an A-cell (those around a hole may not) get marked by one direction in perimeter time.

[†]In general, if a cell in a BCS is in state $q\varepsilon Q$, it shall be called a q-cell.

[††]All distances in this paper are measured using the so-called Manhattan city-block metric $\rho((i,j),(i',j')) = |i-i'| + |j-j'|$.

The crucial observation is that the u's and d's taken to be right and left parentheses, respectively, form a well-bracketed string in a given boundary layer only in the case where the expression is read cw beginning at the position of an A-cell in the topmost row of a boundary layer (there may be more than one). The following steps exploit this observation to locate the topmost A-cells in each boundary layer. It should be noted that there can be no A-cell in the topmost row of the boundary layer of a hole. (There can, however, be an A-cell in one boundary layer which also lies in another--internal--boundary layer for which it is not an A-cell. The FSM M in (3) can set up "shared" boundary layers to avoid the apparent difficulty.)

(4) A consecutive pair of A-cells is an A-cell and the nearest A-cell in the boundary layer in the cw direction. Each consecutive pair of A-cells remaining after step (3) and the boundary-layer cells between can be treated as a 1-D firing squad.[5] Each such squad is made to fire simultaneously--i.e., to begin step (5) below--on a command from a general which is the more cw A-cell of each pair. The command is issued when M in step (3) halts on an A-cell which it does not erase. This step requires perimeter time.

(5) The boundary-layer cells in each boundary are treated as a 1-D DBCS containing an initial pattern on the alphabet $\{u,d,r,\ell\}$. This space functions as a DBCS recognizer for the Dyck language on $\{u,d\}$.[1] The procedure would be quite simple if an entire boundary layer were triggered at once by step (4): Each u-cell sends its state left at unit speed (one cell per time step) as a u-pulse, and each d-cell sends a d-pulse right at unit speed. (Fig. 1.) When a d-pulse collides with a u-pulse, the u-pulse is changed to a u'-pulse, and the d-pulse is changed to a d'-pulse. If a a u'-pulse (d'-pulse) collides with a d-pulse (u-pulse), the u'-pulse (d'-pulse) is annihilated. If a u-pulse or d-pulse encounters an A-cell, then that A-cell is erased since it must lie within a well-bracketed string. Any A-cells remaining are in the topmost boundary-layer cells and are identified as such when a u'-pulse or d'-pulse impinges upon them, assuming no u-pulse or d-pulse has collided with the A-cell.

It is not the case, however, that an entire boundary layer fires simultaneously as a result of step (4), except in the special case of only one A-cell in the boundary layer. Hence it must be shown that the staggered firing induced by step (4) does not essentially alter the procedure just described. See Fig. 2 for an illustration of the technique used: An A-cell determines from its neighborhood whether the firing squads to it left and right have fired and goes into a special state to indicate this status. If a pulse impinges on an A-cell bordering an unfired firing squad, it backs off one cell and maintains position until triggered to proceed at unit speed by a special pulse, called an E-pulse. An E-pulse is generated by an A-cell at the moment when the unfired firing squad first begins firing (assuming the squad on the other side of the A-cell is already firing). The E-pulse propagates away from the newly fired squad at unit speed and is annihilated by the first A-cell (or former A-cell) it encounters. Should a pulse impinge on another which is maintaining position (ie., has backed off one position), then it too backs off one position and maintains position until an E-pulse arrives.

(6) In a manner exactly analogous to steps (1)-(5) above and occurring simultaneously with them, the C-cells in the bottommost boundary-layer cells are located, where a C-cell is a cell with west and south neighbors in the boundary state. Notice that the rightmost C-cell in the bottommost row is cw from the leftmost A-cell in the topmost row, and there are no A-cells or C-cells between them after step (5).

(7) The bottommost C-cells located in step (6) each send out an FSM M' counterclockwise (ccw) in the boundary layer. If M' encounters another C-cell, it halts. Otherwise it proceeds to the first A-cell where it waits for step (5) either to delete the A-cell or to locate it in the topmost row. In the former case, M' moves to the next A-cell and again waits for the results of (5). In the latter case, the A-cell is the desired unique boundary-layer cell and it is marked by a special temporary marker. Then M' proceeds ccw, erasing everything, until it returns to the special temporary marker which it changes to the desired state \$. This step takes external perimeter time. Hence the entire procedure takes perimeter time. Q.E.D.

<u>Corollary</u> 2.1. The accept cell is uniquely located in the external boundary layer in external perimeter time.

<u>Lemma</u> 3. Given a DBCS Z, let $S_R = \{(x_1,y_1),(x_2,y_2),\ldots,(x_s,y_s)\}$ be the set of points in arbitrary retina R consisting of exactly the rightmost cell in the topmost row of each boundary layer in R. Then the points in S_R can be uniquely identified by Z in perimeter time. That is, for c_0 an initial configuration entirely quiescent on R, there is a special state \$ such that $[F^{t_i}(c_0)](x_i,y_i) = \$$ for $t_i = k_i p$, $1 \leq i \leq s$, and such that $[F^{t'}(c_0)](x,y) \neq \$$ for all t' if $(x,y) \notin S_R$ for all $t' < t_i$ if $(x,y) = (x_i,y_i)$. (Should $(x_i,y_i) = (x_j,y_j)$, $i \neq j$, then special state \$' is assumed instead of \$.)

<u>Proof</u>. Lemma 2 provides for the location of the rightmost cell in the topmost row of the external boundary layer in R. The technique of proof of Lemma 2 is readily adapted for obtaining the other points of S_R. Let a D-cell be a cell with southwest J_1 neighbor in state B and its south and west neighbors in non-B states. There is always a D-cell in the topmost row of the boundary layer of a hole. The rightmost of these for each hole boundary layer is in the set S_R. The proof technique of Lemma 2, with 'D-cell" substituted everywhere for "A-cell" and "ccw" interchanged with "cw", will locate the rightmost D-cell in the topmost row of each boundary layer. Hence all points in S_R are located in perimeter time. Q.E.D.

<u>Lemma</u> 4. Given a DBCS Z, the outer and inner boundary layers can be identified in perimeter time. That is, for c_0 an initial configuration entirely quiescent on retina R, there are two special states I and E such that $[F^{t_i}(c_0)](x_i,y_i) = I$ for all (x_i,y_i) in the boundary layer of the i-th hole, $1 \leq i \leq s-1$, and $[F^{t_s}(c_0)](x_s,y_s) = E$ for all (x_s,y_s) in the external boundary layer, where $t_i = k_i p$, $1 \leq i \leq s$. Furthermore, $[F^{t'}(c_0)](x,y) \neq I$ or E for all t' if (x,y) is not in a boundary layer of R or for all $t' < t_i$ if $(x,y) = (x_i,y_i)$, $1 \leq i \leq s$.

<u>Proof</u>. Use Lemma 3 to locate the unique A-cell of the external boundary layer and the unique D-cell for each internal, or hole, boundary layer. Each of these cells acts as the general for a firing squad composed of its respective boundary-layer cells. If the general is an A-cell, then the squad fires by entering state E. If the general is a D-cell, the squad fires by entering state I. Q.E.D.

<u>Corollary</u> 4.1. The outer boundary layer can be identified in time proportional to the external perimeter.

In general, a pulse generated by a cell in state q will be called a q-pulse, even though it may change state during its propagation.

Theorem 5. For an arbitrary ABA (DABA) M, there is a BCS (DBCS) Z which simulates M within perimeter time. That is, if M accepts word w in language L within time T then Z accepts w in time k(T+p), where k is a constant and p is the perimeter of w.

Proof. As stated before, Z can straightforwardly simulate M in twice real time (in exactly real time if M is deterministic). Hence, when M accepts by entering final state F, a cell in Z enters a special "pre-accept" state F' which must be propagated to the accept cell of Z. This is readily done in perimeter time using the lemmas above:

During the simulation of M, Z is marking its accept cell and its outer and inner boundaries as described in the proofs of the lemmas. When a cell in Z enters F', this cell sends an F'-pulse up until a boundary is encountered. If the boundary has not yet been marked inner or outer (I or E, respectively), the F'-pulse maintains position there until it is so marked. If the boundary is outer, then the F'-pulse simply propagates along the boundary to the accept cell of the array, which enters an accept state for Z. If the boundary is inner, the F'-pulse moves along the boundary until it can move up again to another boundary. Eventually the F'-pulse must encounter the outer boundary. Q.E.D.

Corollary 5.1. The BCS languages are precisely the languages generated by monotonic array grammars. In particular, the BCS (DBCS) languages are precisely the languages accepted by ABA (DABA).

Corollary 5.2. If L is a monotonic array language which can be accepted by some ABA (DABA) within perimeter time, then L is a perimeter-time BCS (DBCS) language.

Previous work[6,7] in the area of pattern recognition by 2-D cellular automata has assumed rectangular retinas (i.e., RBCS). This special subclass of the BCS is now related to the results above, as are the SBCS.

Theorem 6. There is a DBCS Z which recognizes the language L of all simply-connected words on arbitrary finite alphabet X within external perimeter time.

Proof. The accept cell of each retina and the outer boundary layer are identified and labeled within external perimeter time, as in Corollary 2.1 and Corollary 4.1. Just after the outer boundary layer is labeled, the accept cell becomes general to the firing squad consisting of the outer boundary-layer cells. Upon firing, all of the cells which have a B to the right send a pulse left at unit speed. Any pulse colliding with a labeled boundary-layer cell terminates. Any pulse colliding with an unlabeled (i.e., inner) boundary-layer cell rebounds at unit speed to the outer boundary, where a reject pulse is sent to the accept cell. Meanwhile, a timing pulse is traversing the outer boundary layer; it is sent out from the accept cell just as the firing squad fires. If, during two complete circuits of the outer boundary layer by the timing pulse, no reject signal is received by the accept cell, then it accepts on the second arrival of the timing pulse. Q.E.D.

Corollary 6.1. Any perimeter-time SBCS (DSBCS) language is a perimeter-time BCS (DBCS) language and an external perimeter-time BCS (DBCS) predicate.

Theorem 7. The language L of all rectangular arrays on a given finite alphabet X is an external perimeter-time DBCS predicate.

Proof. There are eight possible types of "corner" cells as indicated in Fig. 3. These are recognized and tagged by a DBCS Z everywhere they occur in a given retina R. This requires at most two time steps. Simultaneously, the unique A-cell in the outer boundary layer is located and labeled with a \$ and the outer boundary layer is labeled. This requires external perimeter time, as guaranteed by Corollary 2.1 and Corollary 4.1. Just after the outer boundary layer of R is identified, the accept cell of R sends out an FSM M" ccw along the outer boundary layer. M" looks for the sequence of corner cells NE, NW, SW, SE (see Fig. 3) where the accept cell is of course, the NE corner cell. Should M" see a sequence of corner cells different from this as it makes a circuit of the outer boundary layer, it causes Z to reject the array upon its return to the accept cell. Otherwise, the outer boundary layer forms a rectangle. Hence, upon the return of M", the accept cell issues the command to fire to the firing squad formed from the outer boundary-layer cells between, and including, the NE and SE corner cells, which initiates the next step (cf. Theorem 6): All cells on the east side of the retina send a pulse left at unit speed. Should one of these pulses collide with an unmarked boundary-layer cell (i.e., an inner boundary-layer cell), then it is reflected to the east side of the retina and sent to the accept cell which promptly rejects the array. Should time elapse from the firing of the firing squad equal to twice the length of the east and south sides of the retina, then the accept cell accepts the array. (A simple timing pulse sent out by the accept cell can accomplish the clocking of this elapsed time.) Q.E.D.

Corollary 7.1 Any linear-time RBCS (DRBCS) language is a perimeter-time BCS (DBCS) language and an external perimeter-time BCS (DBCS) predicate.

Proof. The corollary follows immediately from Theorem 7 and the fact that any linear-time RBCS language is a linear-time RBCS predicate.[6] Q.E.D.

An important example is provided by Beyer[6] who constructed a DRBCS which recognizes connected black patterns on a white background within linear time. The following is an immediate consequence:

Corollary 7.2 Let L be the language on {black, white}* for which each word is a rectangular array containing a connected black subset of points and having all other points white. Then L is an external perimeter-time DBCS predicate.

Remark. Buneman[8] has demonstrated a monotonic array grammar for the L described in Corollary 7.2.

The generalization of Corollary 7.2 to arbitrary connected arrays is an open problem. An example indicating the difficulties involved is a black ring encircling, but not touching, a hole in the array.

Let a maze be any black and white rectangular array with exactly one occurrence each of a Start point and a Finish point (i.e., Start and Finish are two specially designated colors). A solvable maze has Start connected to Finish via a path of black points. Beyer[6] has proved that solvable mazes are recognizable in linear time by a DRBCS. Hence a second example:

Corollary 7.3. The language of solvable mazes on {black, white, Start, Finish}* is an external perimeter-time DBCS predicate.

As a final adaptation of Beyer's work, we note that linear-time RBCS languages are closed under translation (of a black subfigure to the right as far as possible in an array), rotation (of a square array by 90°), and dilation (of a black subfigure by a constant factor). The reader ~~is~~ referred to Beyer[6] for more details.

The closure properties of the larger class of BCS languages is also of interest. Richardson[9] has stated that the monotonic array languages (and hence the BCS languages) are closed under intersection, union, non-erasing parallel transformation[10], and what shall here be called spatial concatentation, and support closure, where the last two terms are defined as follows:

Definition. Let preimage(w) denote the subset of I^2 which is the preimage of word w in array language L. Then the spatial concatenation of L_1 and L_2 (denoted $L_1 \uplus L_2$) is the set of all words $w_1 \uplus w_2$ defined by

$$[w_1 \uplus w_2](i,j) = \begin{cases} w_1 \text{ if } (i,j)\varepsilon \text{ preimage } (w_1) \\ w_2 \text{ if } (i,j)\varepsilon \text{ preimage } (w_2) \end{cases}$$

where preimage $(w_1) \cup$ preimage (w_2) is a connected subset of I^2 and preimage $(w_1) \cap$ preimage $(w_2) = \phi$.

Definition. For array language L, let $L^1 = L$, $L^2 = L \uplus L$, and $L^k = L^{k-1} \uplus L$. That is, L^k is the k-fold spatial concatenation of L. Let L^+ , the set of all k-fold spatial concatenations of L, for k finite, be called the support closure of L. That is,

$$L^+ = \bigcup_i L^i \quad \text{, for } i \geq 1.$$

The BCS languages with time restrictions have the following easily proved closure properties:

Theorem 8. Any external perimeter-time BCS language is an external perimeter-time BCS predicate. The class of external perimeter-time BCS languages is closed under intersection, union, and set complement.

As an example of the closure properties of BCS languages, consider the language L consisting of oriented 45° right triangles (i.e., those which can be translated so that the right angle vertex is at the origin, each leg lies along an axis, and the hypotenuse lies in the northwest quadrant). Kirsch[2] has given monotonic array grammars for this language; hence it is a BCS language. Similarly, Dacey[11] has shown that each of the three languages obtained by rotating L by 90°, 180°, and 270°, respectively, are also BCS languages. Hence, by support closure, the set of polygons which can be built from oriented 45° right triangles is also a BCS language. In Reference 11, the proof that this language of polygons is a BCS language is derived without benefit of closure properties.

Special BCS Languages and Predicates

In this section, Jordan curves, the majority predicate and Euler number, and convexity are treated in the BCS framework. Here a Jordan curve is a closed, nonrepeating sequence $i_1, i_2, \ldots, i_n$ of connected points in I^2--i.e., i_j is connected to i_{j+1}, $1 \leq j < n$, and i_n is connected to i_1. All points in a Jordan curve are in one state, say black, with all neighboring points in some other state if they are not in the curve.

Theorem 9. The language $L \subset \{\text{black, white}\}^*$ with each word a rectangular array consisting of a single black Jordan curve on a white background is linear-time recognizable by a DRBCS.

Proof. (1) The DRBCS checks for one black component (i.e., a connected black subset of points in the given word) with one hole, Beyer[6] has shown that this requires linear time.

(2) Beginning at time t+1, and hence occurring during the execution of step (1), each black cell counts the number of white components in its J_1 neighborhood. This requires at most two time steps, by Lemma 1, and they are subsumed by step (1). Notice that there must be exactly two white components in the J_1 neighborhood of any black cell in a Jordan curve.

(3) A firing squad is initiated by the accept cell to trigger step (4). Initiation occurs at the completion of step (1). For DRBCS, this requires linear time.[6]

(4) Each black cell which has not recorded the presence of exactly two white components in its J_1 neighborhood sends a reject signal to the accept cell. Otherwise, the accept cell waits only m+n time steps from the time the squad fires before accepting, where the given array has dimensions m by n. Q.E.D.

The proof of the next theorem was given me by Albert Meyer of MIT in a conversation he and I had at the recent symposium on Computers and Automata, sponsored by Brooklyn Polytechnic Institute in New York City. The proof is a good example of the use of numbers to solve what is apparently a geometric problem. I also acknowledge the help of Derick Wood of McMaster University who pointed out the necessity of item (6) in the proof below.

Theorem 10. The majority predicate is linear-time recognizable by a DRBCS. That is, a DRBCS can recognize the language $L \subset \{\text{black, white}\}^*$ with each word consisting of a rectangular array with more black cells than white cells, in linear time.

Proof. (Meyer). Assume for the moment that m<n, where a retina has dimensions m rows by n columns. This restriction is relaxed in step (6).

(1) Each row shifts its black cells to the right and its white cells to the left. That is, each black cell sends a 1-pulse to the right at unit speed and each white cell sends a 0-pulse to the left at unit speed, where 1 represents black and 0 represents white.

(2) The string of 1's in each row can be thought of as a unary number. This number is converted to its binary representation, which is then stored right justified in its row. The conversion can be accomplished in linear time: Think of the row of cells as a register initially containing the binary representation of 0. The string of 1's is shifted into the rightmost cell in this register one at a time as they arrive from the left. Each 1 is added to the register in binary. That is, each cell can form the sum of any two bits and generate a carry. In particular it adds its current contents plus a carry generated at the previous step by the cell on its right and generates a carry bit (0 or 1). Similarly, the string of 0's at the left of each row is treated as a unary number which is also converted to its binary representation and stored "upside down" with its least significant bit at the extreme left of its row. A different alphabet, say {0',1'} is used for this binary number. The conversions take at most $n+\lceil \log_2 n \rceil$ time steps. Hence a pulse which traverses a row in both directions can serve as a clock for step (2) and initiate step (3) in 2n time steps.

(3) The binary numbers at the right end of each row are sent down the retina at unit speed. The bottom row acts as an accumulator which forms the sum of its current contents with the contents of the row above. Thus each cell must be able to form the sum of two bits

plus a carry bit generated by the cell on its right. The arrival of the topmost row signals the addition of the final, or m-th, addend. The propagation of a carry requires at most $\lceil \log_2 mn \rceil < 2\lceil \log_2 n \rceil$ additional steps. A similar sum is formed simultaneously at the upper left of the retina from the binary numbers stored at the left of each row. Hence at the completion of this step, within $m+\lceil \log_2 mn \rceil$ time steps, there are two binary numbers in the retina. The number at the lower right is the number of black cells and the number at the upper left is the number of white cells. A clock pulse which traverses one row in both directions signals completion of this step and initiates the next step.

(4) The number at the upper left is reversed and right justified. This can be done[1] in time proportional to n. Then the next step is initiated by a firing squad.

(5) The binary number at the lower right is sent up the retina at unit speed. The topmost row takes the difference between the two numbers and furnishes a sign indicating whether the number of black cells was greater than (+) the number of white cells or the number of black cells was less than or equal to (-) the number of white cells. The sign is sent to the accept cell.

(6) For the case of arbitrary m, it may be that one or both of the two numbers generated in (3) do not fit in one row. For example, if $m>2^n/n$ and the entire retina is black, then a number of length larger than n must fit in one row. Thus during the additions in step (3), an overflow would occur. Any such overflow signal is propagated to the accept cell. But clearly, if $m>n$, then $n \le m$ and the procedure outlined in steps (1)-(5) above, interchanging rows with columns, must work without overflows. Call the procedure as outlined in (1)-(5) above the "row approach" and the same procedure, with interchanged columns and rows, the "column approach". The DRBCS for the theorem will carry out steps (1)-(5) simultaneously but independently for both the row and column approaches. If an overflow signal is received at the accept cell for one of the approaches, then the result generated by that approach is ignored. Q.E.D.

Corollary 10.1. The set of all words of rectangular shape on {black,white}* with positive Euler number (i.e., with more components than holes) is a linear-time DRBCS predicate.

Proof. See Beyer[6], pp. 91-92.

Another interesting topological property is convexity. Here each cell in I^2 will be thought of as representing a square subset of points in E^2 (the real plane). That is, the cell at point (i,j) will represent all points in $\{(x,y) | i-1/2 \le x \le i+1/2,\ j-1/2 \le y \le j+1/2\}$. A cellular map is a finite nonempty subset of cells in E^2. By the definition of convexity for E^2, a convex cellular map is simply a rectangle. Hence, by Theorem 7, real convexity is an external perimeter-time DBCS predicate. The following definitions (after Sklansky[12]) are intended to provide a more interesting notion of convexity for cellular spaces.

The boundary b_J of cellular map J is the boundary of the union of the cells of J. The cellular boundary B_J of J is the union of cells in J containing at least one point of b_J. If there is a Jordan curve of cells representing precisely the cells of B_J then call this curve the irredundant boundary chain γ_J of J. Let g_J be the union of the boundaries of the cells in γ_J, if it exists, less the boundaries between cells of γ_J. (N.B., $g_J \supset b_J$.) The core boundary bC_J of J is the set of all points in g_J not connected (in the real sense) to b_J. The core C_J of J is the union of bC_J and its interior. J is said to be cored if and only if $C_J \neq \phi$. (N.B., C_J is simply connected and touches every cell in B_J.) A cellular blob is a cored cellular map. Not all connected cellular maps are cellular blobs--e.g., the cellular map represented by H_2.

A figure is a finite simply-connected subset of E^2--i.e., a simple closed curve and its interior. A figure is convex if it contains the line segment that joins any two points of the figure. A cellular map J is an image of a figure p if and only if either (1) $J \supset p$ and every cell in J containing a point not in p contains a boundary point of p, or (2) p is the limit of a sequence of figures each of which satisfies (1). Let I(p) be one of the images of figure p (there may be more than one, see Sklansky[12], p. 6). A polygon is a figure whose boundary contains only (nonintersecting) straight line segments. A minimum perimeter polygon (MPP) of J is any polygon p such that I(p) = J and such that there does not exist a polygon q with perimeter less than that of p and such that I(q) = J.

Definition. A cellular blob J is convex if and only if there is at least one convex figure v such that I(v) = J.

Theorem 11 (Sklansky). A cellular blob has a unique minimum-perimeter polygon. A cellular blob is convex if and only if its minimum-perimeter polygon is convex.

In the following proofs, B_J will be used as an abbreviation for the boundary layer which represents B_J. Similarly, C_J denotes those cells which represent C_J.

Lemma 12. Simply-connected cellular blobs on finite alphabet X are recognizable in external perimeter time by DBCS.

Proof. (1) That a given retina is simply connected is determined within external perimeter time by DBCS, by Theorem 6. If the DBCS determines that the retina is simply connected, then the accept cell emits an FSM which traverses the boundary layer and leaves a tag in each cell visited. If it tries to tag an already tagged cell, then a redundant boundary chain is detected and a reject signal is sent to the accept cell. Otherwise, the FSM completes its circuit of B_J to the accept cell, which then initiates step (2) below. A pattern which is not rejected by this step represents a simply connected cellular map with an irredundant boundary chain which is a cellular blob if it is also cored. Step (2) checks for a nonempty core.

(2) The accept cell initiates a firing squad in B_J just as it accepts the property of step (1). When B_J fires, each cell checks for a neighbor which is neither a B-cell nor a cell in B_J--i.e., checks for a cell in C_J. Each cell which detects such a neighbor sends an accept signal to the accept cell via B_J. One accept signal is sufficient for the accept cell to accept within external perimeter time. Q.E.D.

Theorem 13. The set of convex cellular blobs is an external perimeter-time DBCS predicate.

The proof of this theorem is quite lengthy and is hence delayed to a forthcoming paper on recognition of convexity by cellular automata. Briefly, Lemma 12 is used to detect retinas representing simply-connected cellular blobs. Then several tests are made to reject cellular blobs with obviously concave minimum-perimeter polygons. For example, Fig. 4 shows two cases which would be immediately rejected. A cellular blob J which passes all these tests has core C_J with a boundary b_{C_J} composed entirely of "stairsteps" of height one or two. One final, nontrivial test is then applied to check for a convex minimum-perimeter polygon.

A definition of convexity which is much more restrictive than that just discussed is the so-called x-y convexity of Unger.[13] A simply-connected cellular map J is <u>x-y convex</u> if, for arbitrary point (i,j) in J, (1) there is no point (i,j') in J such that points (i, a), $j' \leq a \leq j$, are not in J, and (2) there is no point (i', j) in J such that points (b,j), $i' \leq b \leq i$, are not in J. The reader can readily verify the next theorem with the use of Theorem 6.

<u>Theorem 14</u>. The set of x-y convex cellular maps is an external perimeter-time DBCS predicate.

<u>References</u>

1. Smith, A.R., "Formal Languages and Cellular Automata," Proceedings of 11th Annual Symposium on Switching and Automata Theory, Santa Monica, California, 1970, 216-224.
2. Kirsch, R.A., "Computer Interpretation of English Text and Picture Patterns," <u>IEEE Trans. Comp</u>., EC-14, 1964, 363.
3. Milgram, D.L., and Rosenfeld, A., "Array Automata and Array Grammars," Proceedings of IFIP '71 Congress, Yugoslavia, 1971.
4. Smith, A.R., "Cellular Automata Complexity Trade-Offs," <u>Information and Control</u>, June 1971.
5. Waksman, A., "An Optimum Solution to the Firing Squad Synchronization Problem," <u>Information and Control</u>, Vol. 9, 1966, 66-78.
6. Beyer, T., "Recognition of Topological Invariants by Iterative Arrays," Ph.D. dissertation, MIT, 1970.
7. Wolpert, L., "The French Flag Problem: A Contribution to the Discussion on Pattern Development and Regulation," Towards a Theoretical Biology: I, Prolegomena (C.H. Waddington, Ed.), Edinburgh University Press, 1968.
8. Buneman, P., "A Grammar for the Topological Analysis of Plane Figures," Machine Intelligence, 5 (D.Michie, Ed.), Edinburgh University Press, 1970.
9. Richardson, D., "Mechanical Recognition and Derivation of Patterns in Space," Unpublished notes, 1971.
10. Yamada, H., and Amoroso, S., "Tessellation Automata," <u>Information and Control</u>, 14, 1969, 299-317.
11. Dacey, M.F., "The Syntax of a Triangle and Some Other Figures," <u>Pattern Recognition</u>, 2, 1970, 11-31.
12. Sklansky, J., "Recognition of Convex Blobs," <u>Pattern Recognition</u>, 2, 1970, 3-10.
13. Unger, S.H., "Pattern Detection and Recognition," <u>Proc. of IEEE</u>, 47, 1959, 1737-1752.

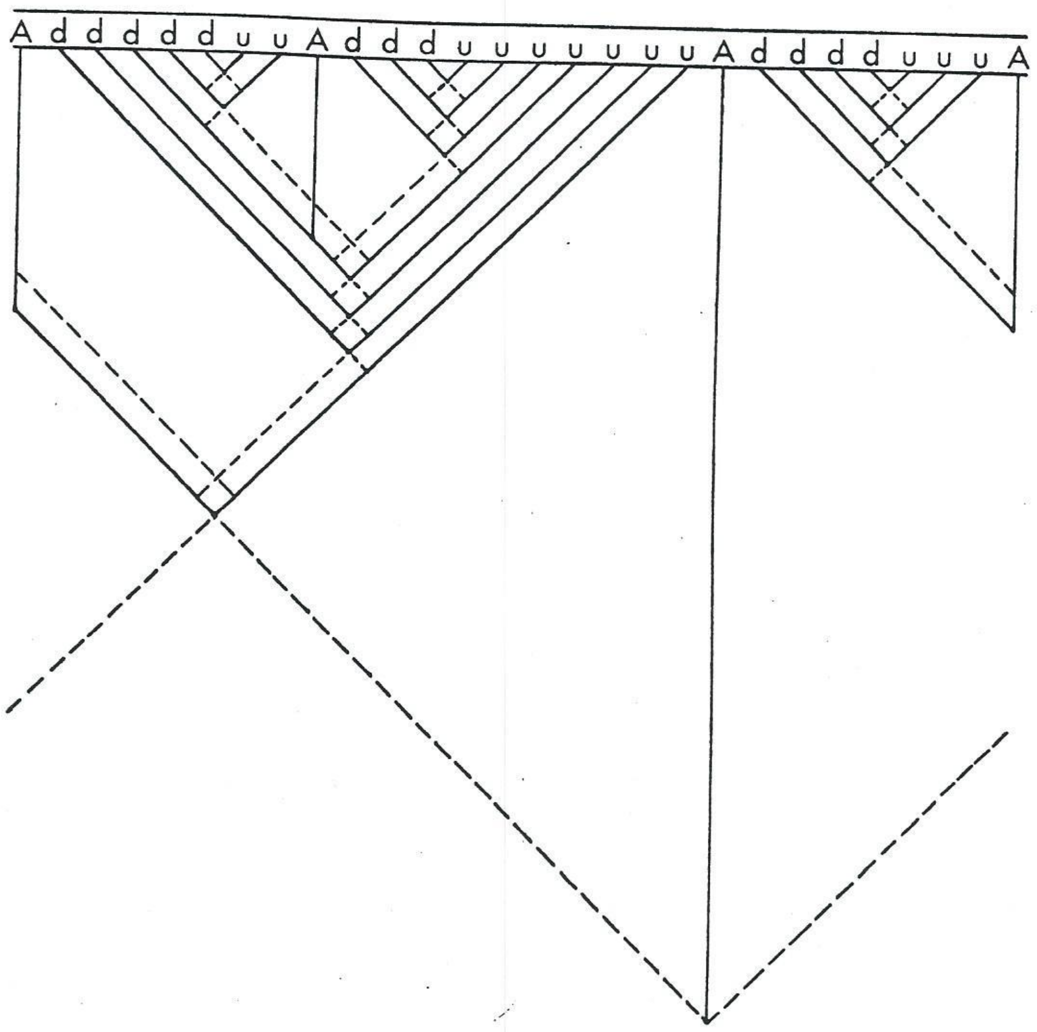

Fig. 1. Dyck boundary language recognition.

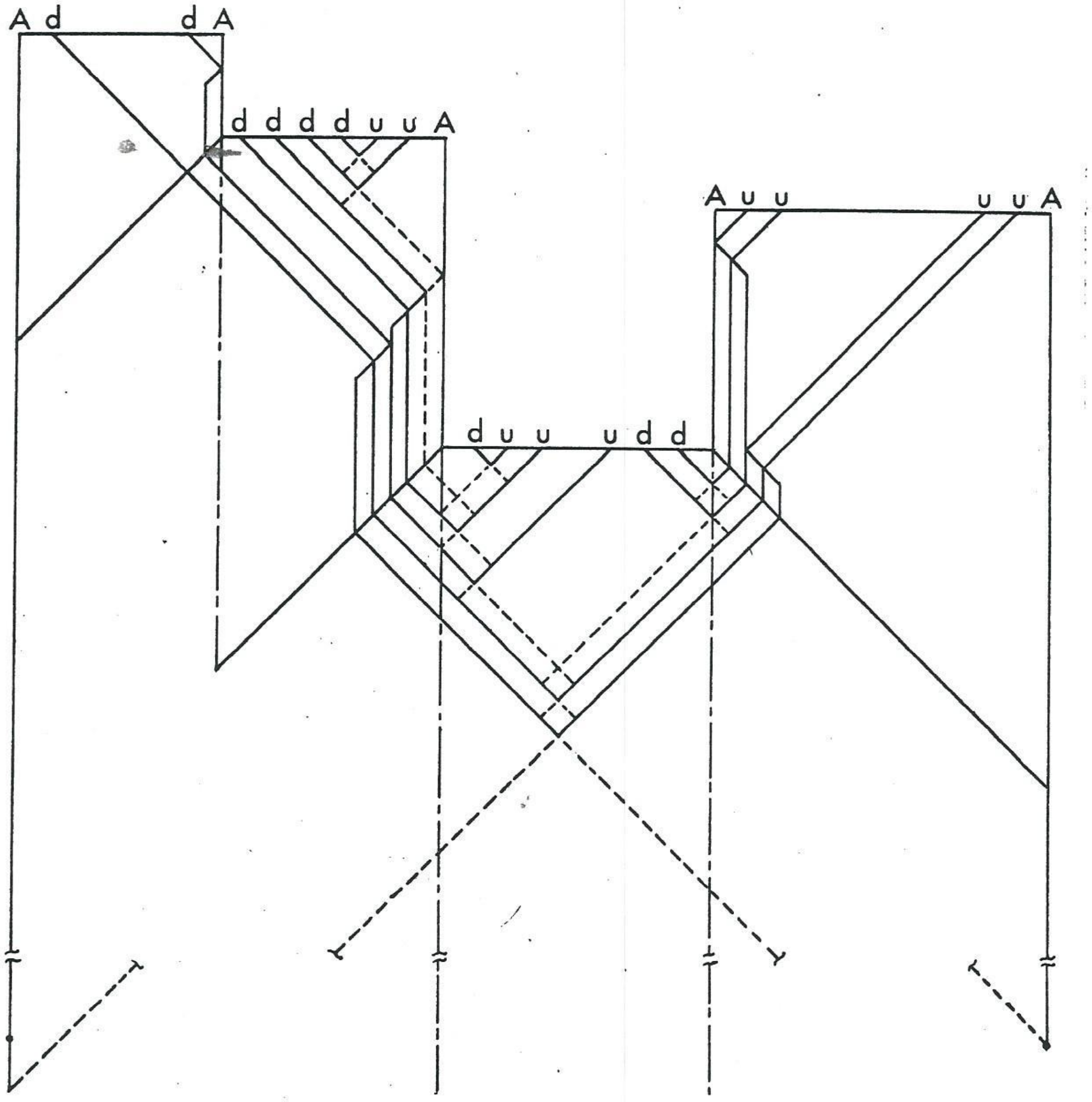

Fig. 2. Staggered firing squad version of Fig. 1.

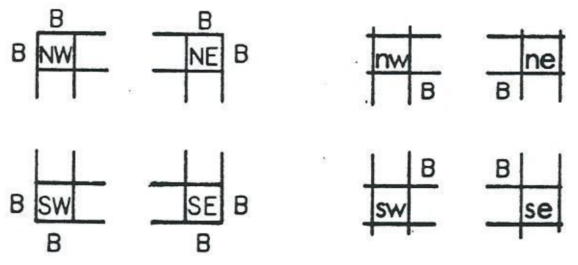

Fig. 3. The eight corner cell types.

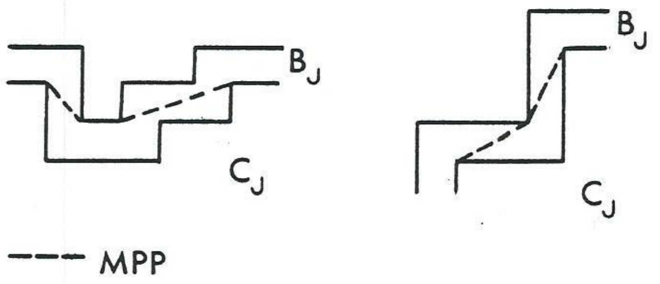

Fig. 4. Portions of two cellular blobs which are not convex.