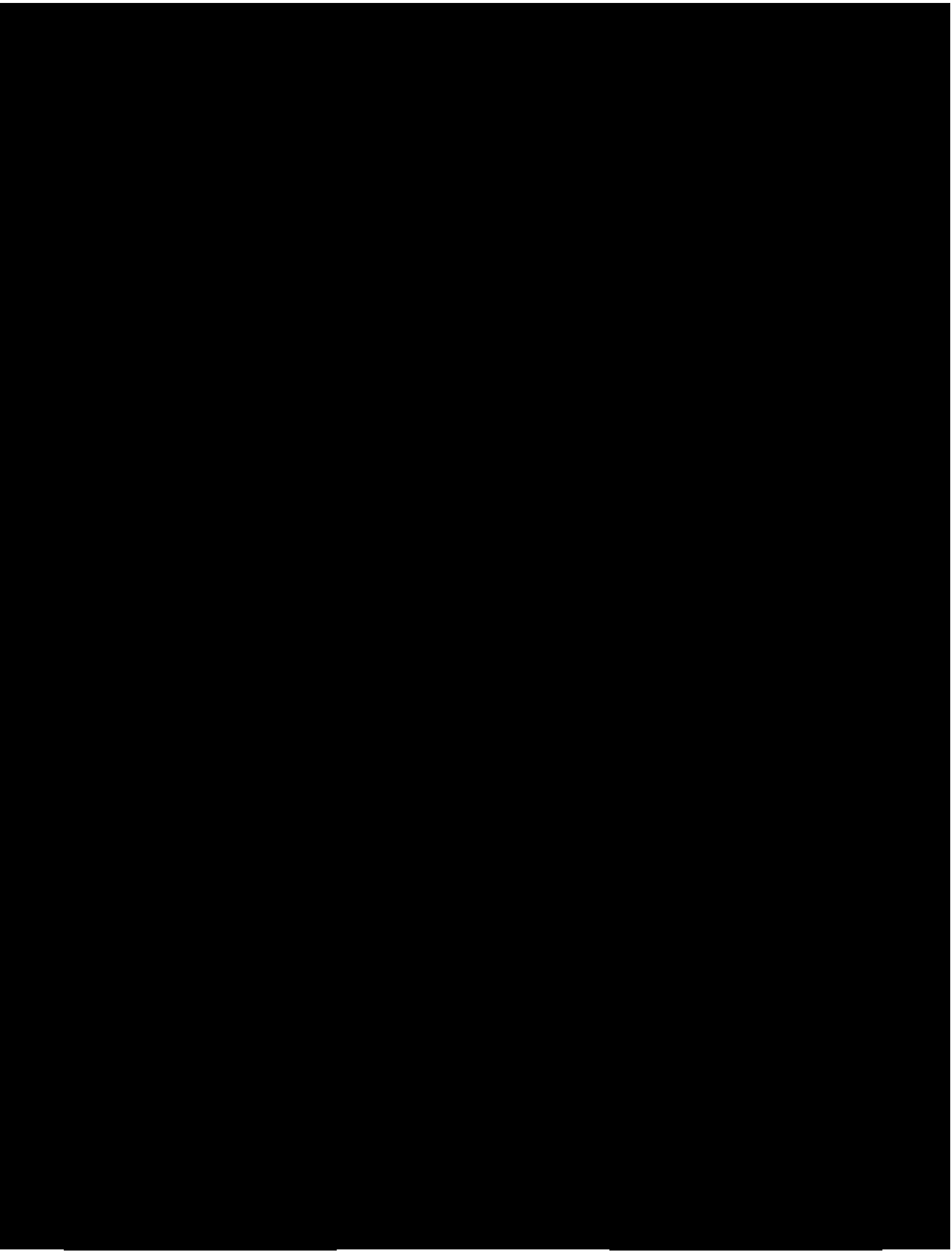


*This blank page was inserted to preserve pagination.*

# RECOGNITION OF TOPOLOGICAL INVARIANTS BY ITERATIVE ARRAYS

Wendel Terry Beyer

24 October 1969

PROJECT MAC

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Cambridge Massachusetts 02199

Massachusetts Institute of Technology
Project MAC
545 Technology Square
Cambridge, Massachusetts
02139

## ACKNOWLEDGEMENTS

I wish to express my gratitude to Seymour Papert for his supervision of this thesis and to Marvin Minsky and Mike Paterson, the other members of my committee. I especially appreciate the time and energy given so unsparingly by Mike Paterson and the enthusiasm of Manuel Blum, who helped get this work underway. Many others have given me assistance and guidance at one time or another. Finally I am deeply indebted to my wife, Kathleen, whose continued encouragement and aid made it all possible.

Work reported herein was supported in part by Project MAC, and M.I.T. research project sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Nonr-4102(01). Reproduction of this report, in whole or in part, is permitted for any purpose of the United States Government.

Government Contractors may obtain copies from:

Defense Documentation Center, Document Service Center, Cameron Station, Alexandria, VA 22314

Other U.S. citizens and organizations may obtain from:

Clearinghouse for Federal Scientific and Technical Information (CFSTI) Sills Building, 5285 Port Royal Road, Springfield, VA 22151

# RECOGNITION OF TOPOLOGICAL INVARIANTS* BY ITERATIVE ARRAYS


## Abstract

A study is made of the recognition and transformation of figures by iterative arrays of finite state automata. A figure is a finite rectangular two-dimensional array of symbols. The iterative arrays considered are also finite, rectangular, and two-dimensional. The automata comprising any given array are called cells and are assumed to be isomorphic and to operate synchronously with the state of a cell at time t+1 being a function of the states of it and its four nearest neighbors at time t. At time t=0 each cell is placed in one of a fixed number of initial states. The pattern of initial states thus introduced represents the figure to be processed. The resulting sequence of array states represents a computation based on the input figure. If one waits for a specially designated cell to indicate acceptance or rejection of the figure, the array is said to be working on a recognition problem. If one waits for the array to come to a stable configuration representing an output figure, the array is said to be working on a transformation problem.

Chapter 2 contains a general theory of recognition. Thorems on the amount of time required to perform recognition and on methods of speeding up recognition are presented. Some properties of the classes of recognizable figures are given. Arrays are compared to other types of figure recognition devices. In the last section the class of linear predicates is studied. A linear predicate is a family of figures which can be recognized in time proportional to the perimeter of the figure.

Chapter 3 contains a study of the recognition of some topologically invariant properties of figures. A fundamental transformation of figures is presented and is then used to show that a wide variety of topologically invariant properties form linear predicates including connectivity and maze solvability. Two properties whose linearity is open are discussed.

Chapeter 4 contains a brief study of transformation problems. Some general theorems are presented as well as discussions of specific transformations. An optimal solution to the two-dimensional firing squad synchronization problem is also presented in Chapter 4.

In addition to the formal results, several open questions are presented and some iterative programming techniques are considered.


*This report reproduces a thesis of the same title submitted to the Department of Mathematics, Massachusetts Institute of Technology, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

TABLE OF CONTENTS


| Chapter | | | Page |
|---|---|---|---|
| | Acknowledgements | | 2 |
| | Abstract | | 3 |
| 1 | INTRODUCTION | | 5 |
| | 1.1 | The Topic | 5 |
| | 1.2 | The Background | 7 |
| | 1.3 | The Layout | 10 |
| 2 | THEORY OF RECOGNITION | | 13 |
| | 2.1 | Basic Definitions | 13 |
| | 2.2 | An Example | 21 |
| | 2.3 | Timing | 25 |
| | 2.4 | Speed-Up | 31 |
| | 2.5 | Computing Power | 46 |
| | 2.6 | Linear Predicates | 60 |
| 3 | RECOGNITION OF TOPOLOGICAL INVARIANTS | | 65 |
| | 3.1 | Basic Terminology | 65 |
| | 3.2 | An Example | 68 |
| | 3.3 | A Fundamental Transformation | 71 |
| | 3.4 | Linear Recognition of Topological Invariants | 79 |
| | 3.5 | Euler Number | 90 |
| | 3.6 | Topological Match Problem | 92 |
| 4 | DISPLAY PROBLEMS | | 119 |
| | 4.1 | Introduction | 119 |
| | 4.2 | Specific Problems | 124 |
| | 4.3 | Two-Dimensional Firing Squad | 132 |
| | Bibliography | | 137 |
| | Index | | 140 |
| | Index of Symbols | | 143 |
| | Biographical Note | | 144 |

# CHAPTER 1

# INTRODUCTION

## 1.1 The Topic

In this thesis we study the recognition and transformation of figures by iterative arrays of finite state automata. For our purposes a figure is a finite rectangular two-dimensional array of symbols. Our iterative arrays are also finite, rectangular, and two-dimensional. We call the automata, which make up such an array, cells. All the cells in an array are assumed to be of the same type, that is, isomorphic. The cells on the edges and corners of an array may operate in a manner quite distinct from those in the interior, but this is to be thought of as an effect which takes place because these cells can sense that they are on the edges rather than because they are inherently different from the interior cells. All of the cells are placed in the array with common orientation and each cell is connected to its four nearest neighbors. The array functions synchronously, with the state of each cell at time $t + 1$ being a function of the states of it and its four nearest neighbors at time $t$.

At time $t = 0$ we place each cell in some initial state. The configuration of initial states thus introduced represents a figure which is taken to be the input to the array. Given an input figure, the array proceeds from state to state with the state transitions of the array being determined by the transition function of the cell type from which the array was constructed. The progression of array

states may be interpreted as a computation based on the input figure. Of the many interpretations possible, we will consider two which we call recognition and display.

In a recognition computation we view the array as an acceptor of figures in much the same way that a finite state automaton may be viewed as an acceptor of tapes. Two cell states are designated as final states corresponding to accept and reject. These states are assumed to be terminal. We input a figure, allow the computation to proceed, and observe some specially designated cell, say the northwest corner cell. When that cell enters one of the two final states, we say that the figure has been accepted or rejected.

In a display computation we view the array as a device for performing a transformation of the input figure. Certain cell states are designated as final states and are assumed to be terminal. We input a figure, allow the computation to proceed, and observe the array until it enters a state in which each cell is in a final state. We interpret the resulting configuration of final states as a figure and take that figure to be the output of the computation.

The major portion of this thesis is devoted to the study of recognition. A central role in this study is played by the concept of predicates which are simply collections of figures. Usually the figures comprising a particular predicate share some common property which is of interest. Given a particular predicate and a cell type, we say that the cell type recognizes the predicate if the arrays of that type accept exactly those figures belonging to the predicate and reject all others. We consider questions such as the following:

"What may be said about the class of predicates which are recognizable by arrays?"

"What may be said about the speed with which a given predicate can be recognized?"

"How powerful are arrays as recognition devices?"

"To what other devices may they be compared?"

A modest theory of recognition is developed in an attempt to answer these and other questions. In addition the application of arrays to some specific recognition problems is considered. These problems include the recognition of connectivity, simple connectivity, and more complicated topologically invariant predicates. Display problems are treated briefly in a concluding chapter.

## 1.2 The Background

One of the earliest uses of iterative arrays was by von Neumann who used the structure of a regular array of identical automata as the framework for a study of self-reproducing automata. The von Neumann manuscript has been edited and completed by Burks[16].

Hennie[8] has performed an extensive analysis of the functioning of iterative arrays in several dimensions. In his work arrays are classified according to the number of dimensions, the number of directions of signal flow, and whether or not the cells have an internal memory. Hennie's cells are equipped with external input and output lines. Figures are presented to the array by placing an appropriate stimulus on each input line and maintaining the stimulus until an

appropriate output is obtained. He does not always assume that the cells have been reset to a canonical state at the time the input is presented. Thus a given input figure may cause different behavior in the array depending on the configuration of states at the time the input was presented. If an array always achieves a steady state no matter what its initial configuration and input, it is said to be stable. If an array has exactly one steady state corresponding to any given input, it is said to be regular. Hennie studies the question of determining the stability or regularity of a family of arrays, given a description of a typical cell. He finds algorithms for answering these questions in most one-dimensional cases and goes on to show that the same questions are recursively unsolvable in higher dimensions under all but the most severe restrictions on signal flow. He also studies the relative computing power of arrays of various types. Many of the questions studied by Hennie deal with arrays per se as opposed to the application of arrays to computational problems. Since we tend to emphasize the latter type of problem, we feel that our work forms a complement to that of Hennie.

Many people are introduced to iterative arrays via the one-dimensional firing squad synchronization problem. This problem has been credited to John Myhill (1967) by Moore[11]. Solutions of varying degrees of efficiency and generality have been published by Waksman[17], Balzer[3], and Moore and Langdon [12]. The two-dimensional firing squad is discussed in Section 4.3 below.

The real time computing power of iterative arrays has been studied by Cole[6], Atrubin[2], and Fischer [7]. In their models, one cell

of the array is equipped with input and output channels. A time sequence of inputs is fed into this cell and a computed output sequence is produced. Arrays are thus viewed as another form of sequence transducer. Atrubin shows that multiplication of two binary coded numbers may be performed in real time by an infinite one-dimensional array. Fischer shows that an infinite one-dimensional array can generate the characteristic function of the set of prime integers in real time. Cole performs extensive studies of the real-time computational powers of infinite iterative arrays in arbitrary dimensions. He establishes relations between the real-time computing power of such arrays and the information capacity of the inter-cell connections. The use of arrays as figure computers is not considered in these papers.

The use of arrays to process two-dimensional figures has been considered by Atrubin [1] who analyzes several examples of simple figure transformations, but makes no attempt to formulate a general theory.

Many algorithms for serial computers have been published which find a natural setting in iterative arrays. Examples are the shortest path method of Lee[9] and the picture processing of Rosenfeld and Pfaltz[14].

The review of highly parallel computers by Murtha[13] contains the designs of many theoretical and actual computers which incorporate regular arrays of identical processing elements. Among those discussed we might mention the "spatially oriented" computer of Unger[15] and the ILLIAC IV described by Barnes[4].

Our work has been greatly influenced by the work of Minsky and Papert[10] on the perceptron. In their book they state:

> "Good theories rarely develop outside the context of a background of well-understood real problems and special cases. . . . Accordingly, our best course would seem to be to strive for a very thorough understanding of well-chosen particular situations in which these concepts (parallel, serial, etc.) are involved."

This thesis is an attempt to analyze a special case of parallel processing in the same spirit and in a manner compatible with that of Minsky and Papert.

## 1.3 The Layout

Chapter 2 contains a general theory of recognition. After some basic definitions, an example of the solution of a recognition problem is given. Next the amount of time required for the solution of recognition problems is taken up. We give the Interdependence Theorem which allows us to predict the future state of a cell, given sufficient information about the current states of it and its neighbors. The Interdependence Theorem is used to establish lower bounds on the recognition time of most predicates and is also the basis of the Speed-Up Theorem. An adaptation of a well-known iterative technique, the Speed-Up Theorem states that if recognition can be carried out by an array within time $T(m,n)$ where the array is of size $m \times n$, then for any integer $k$ a second array can be constructed which will carry

out the computation within time $\frac{1}{k} \cdot T(m,n) + m + n + 2$. Finally we have the Minimizing Theorem, which says that two distinct methods of recognizing a predicate may be combined to obtain a method which is as fast on any figure as the faster of the two.

Chapter 2 continues with a study of the computing power of arrays. We find that arrays are not universal computers, but are more powerful as figure recognizers than the pebble automata of Blum and Hewitt[5]. In fact, arrays are equivalent in power to, although faster than, linear bounded automata which are allowed to walk about on a figure. This equivalence was to be expected since the amount of storage available to an array increases linearly with the size of the figure. We show that the class of recognizable predicates forms a Boolean algebra. Some undecidability results are obtained including the undecidability of whether or not a given cell type recognizes a given predicate.

Chapter 2 ends with a brief study of the class of linear predicates (those which are recognizable in time proportional to the perimeter of the array). It is shown that a linear predicate is, in a certain well-defined sense, recognizable almost as fast as any predicate. This fact is interesting because it is shown in Chapter 3 that some intuitively very complicated predicates are linear. The class of linear predicates is shown to be a Boolean algebra and some unsolvable problems are presented. Finally we discuss the open question of whether or not all recognizable predicates are linear.

Chapter 3 contains a study of some topologically invariant predicates (predicates over black and white figures which depend only

on the manner in which the holes and components of the figures are embedded within each other). We first develop a simple but very powerful transformation of figures called the connectivity transformation. Using this transformation as a basis, we prove that a wide variety of predicates including "connectivity" and "simple connectivity" are linear. It is shown that it can be determined whether or not a maze is solvable in less time than is required for the transmission of a signal along the shortest path of the maze. The problems of solving multilevel mazes and developing a three-dimensional connectivity transformation are discussed. It is shown that the solution in linear time of multilevel mazes would imply that any predicate recognizable by a finite state automaton was linear.

Chapter 4 contains a brief description of the use of arrays in display problems and presents some typical figure transformations which may be carried out. Several open questions are presented.

Throughout the thesis many open questions are raised. These questions may be referenced by looking in the index under "open questions."

In addition to the formal results obtained, we have included discussions of several interative programming techniques. These techniques were developed to solve specific problems, but are of general interest. They may be found by looking in the index under "programming techniques."

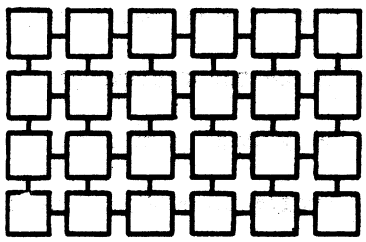
# CHAPTER 2

# THEORY OF RECOGNITION

In this chapter we will formalize the notion of iterative array and study some aspects of the theory of the recognition of figures by iterative arrays.

## 2.1 Basic Definitions

Consider a finite, rectangular, two-dimensional iterative array of finite state automata. Such an array is pictured below with the automata represented by squares. The lines connecting the squares represent communication channels between the automata.

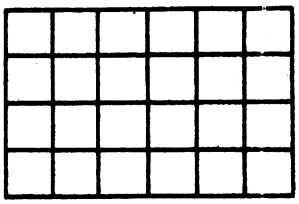


The automata used in such a construction are called cells and the entire arrangement is called an iterative array or simply an array. Other terms used in the literature include "cellular array" and "iterative array of logical circuits." We assume that all of the cells in the array are isomorphic and have been placed in the array with common orientation. Each cell is connected with its four nearest neighbors. The array functions synchronously with the state of a cell at time t + 1 being a function of the states of it and its four nearest neighbors at time t.

The above description assumes that every cell in the array has four nearest neighbors while in the diagram it appears that the cells on the edges and corners of the array have fewer than four. We have here a conflict of interest. On the one hand we would like to be able to treat all the cells as if they were the same, allowing us to make statements such as "All cells are isomorphic" and "Each cell is connected to its four nearest neighbors." On the other hand we definitely want to make use of the fact that the cells on the edges and corners can operate in a manner different from those in the interior. Fortunately these two points of view can be resolved by a simple technical device. Informally we will continue to think of an array as a finite rectangular arrangement of cells. Formally, however, we will picture this finite array of cells as being embedded in a two-way infinite cellular space. All of the cells in this space which are not within the finite array will be in a special terminal state, e, called the edge state. Thus the only real computation within the space takes place within that finite portion known as the array. Any cell in the array can determine where it lies with respect to the boundary of the array by determining which of its neighbors are in edge states. Thus, for example, the northwest corner cell of the array can operate in a manner which is completely unlike any other cell in the array and yet we can consider it to be isomorphic with all other cells in the array. The description of a "typical" cell in an array must actually describe the behavior of that cell in each of the sixteen possible positions in which it can find itself with respect to the boundary of an array.

For the sake of simplicity we will omit from our diagrams all

cells which do not lie within the array and will suppress the lines indicating intercell connections. Our diagram now becomes simply:



Since the cells which do not lie within the array are always in state e, we seldom have to mention them explicitly. Nevertheless they are tacitly assumed to be present at all times.

To operate an array as a recognition device for black and white figures, one designates two of the cell's states to be initial states corresponding to black and white. At time t = 0 every cell in the array is placed in one of the two initial states and the pattern of states thus created represents the figure to be processed. Beginning with the initial state representing the figure, the array proceeds from state to state until finally a designated cell (we will always use the northwest corner cell) enters one or the other of two specially designated final states thus indicating whether the figure has been accepted or rejected. The final states are assumed to be terminal. Note that we use only one accept state rather than many. The use of a single terminal accept state is merely a technical convenience and causes no loss of generality. By using the techniques of Theorem 2.5 below, any cell type with multiple non-terminal accepting states can be converted into a cell type with a single terminal accept state.

An array operating in the mode described in the preceding paragraph is said to be working on a recognition problem. The

extension of recognition problems to include input figures of more than two colors and to n-way classification rather than binary classification is straight forward.

These informal remarks motivate the following definition.

Definition

A cell type M is a 5-tuple (S,I,F,e,g) where

S is a finite set of cell states;

I is a subset of S called the initial states;

F is a subset of S called the final states;

e is a distinguished member of S called the edge state; and

g is a function $g: S \times \begin{matrix} S \\ S \\ S \end{matrix} \times S \longrightarrow S$ called the transition function,

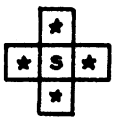
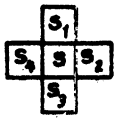
such that final states are terminal

(i.e. $\forall s \in F,\ g\ \left( \begin{matrix} \star \\ \star\ s\ \star \\ \star \end{matrix} \right) = s$ )

and edge states are conserved.

(i.e. $\forall s \in S,\ g\ \left( \begin{matrix} \star \\ \star\ s\ \star \\ \star \end{matrix} \right) = e \iff s = e$ )

Two notational devices have been introduced in the above definition. One is the two-dimensional cartesian product. Rather than express the domain of the transition function as the usual linear cartesian product of sets, we have taken the liberty of arranging the factors of the product in a two-dimensional manner which more clearly illustrates the process being modelled. The second notational device

introduced is the don't care symbol, $\star$, which is used to replace universally quantified variables. Thus "$g($  $) = s$" is notational shorthand for "$\forall s_1 \forall s_2 \forall s_3 \forall s_4\, g($  $) = s$." The set over which quantification takes place is usually understood.

Let M be a cell type. An array composed of cells of type M is said to be an array of type M. If the array has m cells per column and n cells per row, it is said to be an m×n array of type M. The notion of cell type formalizes the mechanism underlying the operation of an array. Corresponding to the idea of an instantaneous description in the theory of Turing machines, we have the following definition.

Definition

> An m×n description of type M (or simply a description if m, n, and M are understood) is an m×n matrix with entries in S, where S is the set of cell states of the cell type M. An initial description is a description, all of whose entries are initial states.

Note that a m×n description contains only enough information to determine the states of the cells within an m×n array. The states of the remaining cells in the space are formally set equal to e by the following definition.

Definition

If D is an m×n description of type M then the state of cell (i,j) in D, denoted $D_{ij}$, is the (i,j)-th entry in D provided $1 \leq i \leq m$ and $1 \leq j \leq n$, and is e otherwise, where e is the edge state of M.

Note that we have introduced matrix type coordinates. For example cell (1,1) is the northwest corner cell and cell (m,n) is the southeast corner cell.

The transition function is now used to define the obvious notion of successor.

Definition

Let D be an m×n description of type M. The successor of D is the m×n description of type M, D', given by

$$D'_{i,j} = g\left( \begin{array}{ccc} & D_{i-1,j} & \\ D_{i,j-1} & D_{i,j} & D_{i,j+1} \\ & D_{i+1,j} & \end{array} \right)$$

where g is the transition function of M.

We can now formalize the concept of a computation.

Definition

An m×n computation of type M is an infinite sequence $\mathcal{D} = D^0, D^1, D^2, \ldots$ of m×n descriptions of type M such that

$D^0$ is an initial description and

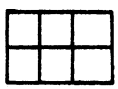
$D^{i+1}$ is the successor of $D^i$ for all $i \geq 0$.

If $\mathcal{D} = D^0, D^1, D^2, \ldots$ is a computation, then $D^t_{i,j}$ is said to be the state of cell (i,j) at time t in the computation $\mathcal{D}$.

This completes the formalization of the terms necessary for describing an array and its functioning. We now formalize the terms necessary to describe recognition problems.

Definition

An m×n figure over the set I ( or simply figure if m, n, and I are understood) is an m×n matrix with entries in I.

We will most frequently consider figures over the set $I_2 = \{b,w\}$ representing black and white. When we wish to represent a specific figure over $I_2$, we will use a diagram of the form rather than standard matrix notation. Note that a figure has a specific size. Thus and  are distinct figures even though both are blank.

Recognition problems involve the separation of all figures over a fixed set into two classes, the accepted figures and the rejected

figures. We concentrate our attention on one of the two classes.

Definition

A predicate over the set I is a subset, $\psi$, of the set of all figures over I. The complement of $\psi$, denoted $\overline{\psi}$, is the set of all figures over I which are not in $\psi$.

It is the predicate which allows us to make connections between figures which have similar properties. For instance we could form a predicate by taking the set of all blank figures or the set of all figures containing five or fewer black squares.

Finally we relate computations to recognition of predicates. For this purpose we will fix a set $F = \{a, r\}$ which represents the final two states of any cell type involved in a recognition problem. The states a and r correspond to accept and reject respectively.

Definition

Let $\psi$ be a predicate over the set I and let M be a cell type. We say M accepts $\psi$ (respectively rejects $\psi$) if the following hold:

i) I is the set of initial states of M.

ii) $F = \{a, r\}$ is the set of final states of M.

iii) Given any computation $\mathcal{D} = D^0, D^1, D^2, \ldots$ of type M, we have $D^0 \in \psi \iff \exists\, t$ such that $D^t_{1,1} = a$ (respectively $D^t_{1,1} = r$).

We say that M recognizes $\Psi$ if

i) M accepts $\Psi$, and

ii) M rejects $\overline{\Psi}$.

Note that we actually require the set of initial states of M to be the same set as that over which $\Psi$ is defined. This not only eliminates the need for introducing an arbitrary correspondence, but actually makes a figure and an initial description the same formal object.

We have made an arbitrary choice of cell (1,1), the northwest corner cell, as being the cell which is designated to give the accept or reject signal. One could make a case for having the designated cell be somewhere more centrally located in the array, but then one would either have to introduce additional machinery so that that cell could be singled out, or have the array compute the location of that cell each time it performed a computation.

## 2.2 An Example: $\Psi_{PAR}$

Perhaps it would be best at this point to give some life to our definitions by considering an example.

Let $\Psi_{PAR}$ be the predicate over $I_2$ consisting of all figures with an odd number of black cells. This parity predicate plays an important role in the work of Minsky and Papert on the perceptron[10]. They show that $\Psi_{PAR}$ is very difficult for a perceptron to recognize and use this fact to show that many other predicates which are reducible in perceptron theory to $\Psi_{PAR}$ are also difficult for a

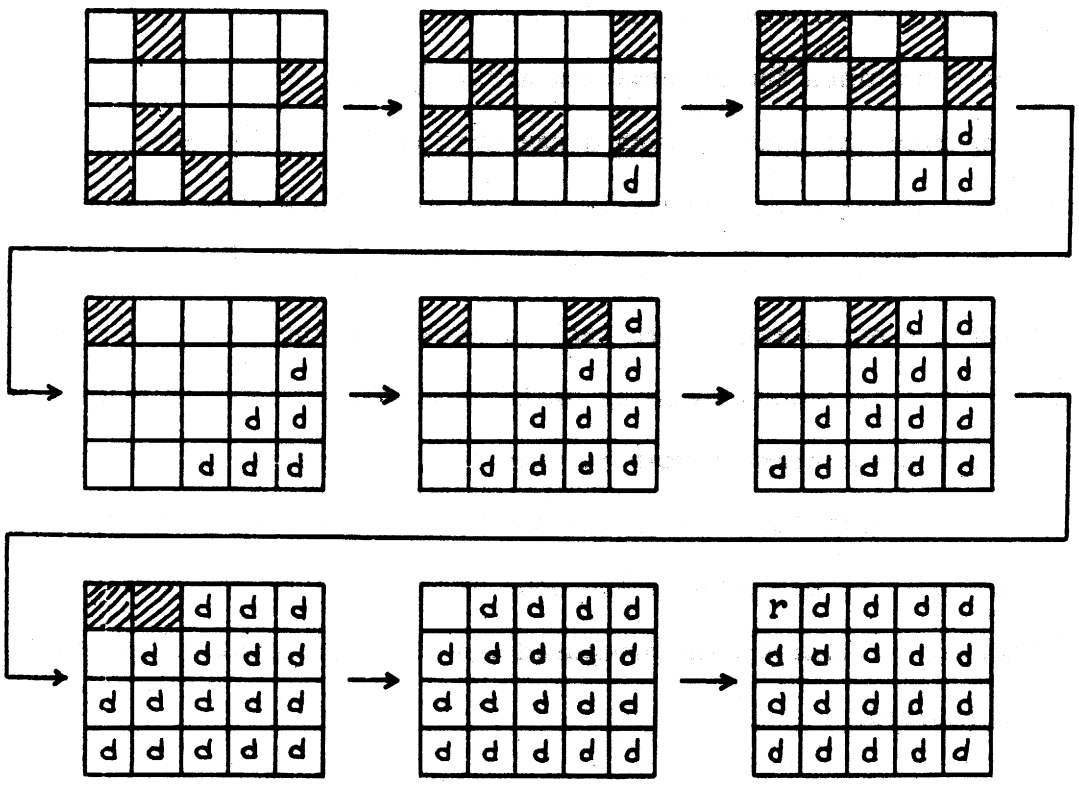
perceptron to recognize. We will see that $\Psi_{PAR}$ is quite easily recognized by arrays. It is introduced here only as an example and plays no role in our theory.

We now describe a cell type $M_{PAR}$ which recognizes $\Psi_{PAR}$. The most precise way of describing $M_{PAR}$ would be to list its states and display a state transition table. Unfortunately a cell type with s states has $s^5$ rows in its complete state transition table. Since the cell type we are about to describe has 6 states, its transition table would have $6^5 = 7,776$ entries, making it quite unreadable as well as shedding little light on the method by which $M_{PAR}$ carries out its computation. Use of the don't care symbol, $\star$, drastically reduces the number of entries needed, but still leaves the underlying method of computation to be puzzled out be the reader. It has been found that the best method of describing the functioning of a cell type is to describe the action of a typical array composed of cells of that type on a typical figure. Attention is thus focused on the method of computation rather than the machinery which carries it out. In most cases we do not even attempt to give a complete list of the states involved in the cell type. The reader who is interested in more detailed analysis of cell types may refer to Hennie[8] and Atrubin[1].

Operation of $M_{PAR}$: Assume the figure to be processed is at least 2×2. The cases of 1×n and m×1 are easy modifications of the main idea. Each cell which is not on the northern edge of the array simply copies the state of its southern neighbor. In this way the information within each column of the array is shifted to the north.

When the information arrives at the northern edge, it is shifted to the west by the cells on the northern edge. All information eventually arrives at the northwest corner. Each cell on the northern edge of the array combines the information arriving from the south with that arriving from the west in such a way that parity is preserved. The northwest corner cell eventually arrives in state b or w depending on whether the parity was odd or even respectively. All that remains is to cause the northwest corner cell to enter the appropriate final state. This is accomplished by having a contagious "done" signal d begin in the southeast corner at time t = 1 and spread at the rate of one cell per unit time toward the northwest corner cell. By the time the northwest corner detects the done signal, all of the parity information will have been processed and the appropriate final state can be entered.

The following diagram illustrates the process for a particular 4x5 array.



$M_{PAR}$ as described above has six states:

| | | |
|---|---|---|
| b | black | initial states |
| w | white | |
| d | done | |
| e | edge | |
| a | accept | final states |
| r | reject | |

The reader who is concerned with the number of states used in $M_{PAR}$ may wish to show that $M_{PAR}$ could be modified to have only five states.

## 2.3 Timing

Note that $M_{PAR}$ in general takes m + n - 1 time units to recognize an mxn figure. We will be quite interested in the amount of time required to recognize various predicates, so we now define some appropriate terminology.

**Definition**

Let M be a cell type with initial states I; P, a figure over I; and $\mathcal{D} = P, D^1, D^2, D^3, \ldots$ the computation of type M with P as its initial description. If t is a real number, we say that M recognizes P within time t provided that $D^{[t]}_{1,1}$ is a final state of M, where $[\,]$ is the greatest integer function.

It would certainly have been adequate in the above definition to restrict t to the positive integers. However, the slight generality obtained by allowing t to range over the reals will be useful later. The timing functions in the following definitions are real valued for the same reason.

**Definition**

Let $\Psi$ be a predicate over I; M, a cell type which recognizes $\Psi$; and T(m,n), a real-valued function. We say that M recognizes $\Psi$ within time T(m,n) provided that M recognizes every mxn figure over I within time T(m,n).

The following definition forms a basis for measuring the complexity of predicates with respect to iterative arrays.

Definition

Let $\psi$ be a predicate and T(m,n) a real-valued function. We say that $\psi$ is recognizable within time T(m,n) if there exists a cell type M such that M recognizes $\psi$ within time T(m,n).

Thus we would say that $M_{PAR}$ recognizes $\psi_{PAR}$ within time m + n - 1 and that $\psi_{PAR}$ is recognizable in time m + n - 1. As we shall see in Corollary 2.1.1 below, $\psi_{PAR}$ is not recognizable in time m + n - 2. This result is an application of the Interdependence Theorem. To properly state the Interdependence Theorem, we need three more definitions.

Definition

Let $c_1 = (i_1, j_1)$ and $c_2 = (i_2, j_2)$ be two cells. The distance between $c_1$ and $c_2$ is given by the function $\rho(c_1, c_2) = |i_1 - i_2| + |j_1 - j_2|$.

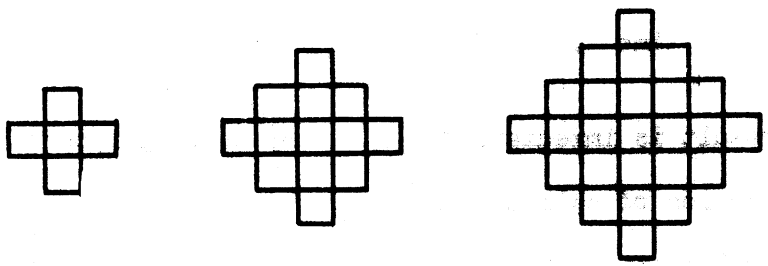
Note that the distance function $\rho$ is a metric on the set of all cells and that the distance between two cells equals the amount of time it takes a signal to travel from one to the other.

**Definition**

If d is a positive integer, then the **d-neighborhood** of a cell c is the set

$$N_d(c) = \{ c' \mid c' \text{ is a cell and } \rho(c,c') \leq d \}.$$

Since neighborhoods are defined over the two-way infinite cellular space of cells, there are no edge effects and hence all d-neighborhoods have the same size and shape. Indeed the d-neighborhood of a cell c contains $1 + 2 \cdot d \cdot (d + 1)$ cells and forms a diamond shaped cluster of cells about c. For example, 1-, 2-, and 3-neighborhoods have the following shapes:



Note that the state transition function for any cell type is a function which has as its domain a two-dimensional cartesian product of cell states in the shape of a 1-neighborhood and which is used to predict the state of the central cell of that neighborhood at one time unit in the future. The following definition and theorem generalize this concept.

**Definition**

Let d be a positive integer and M a cell type with set of states S. Then a **d-neighborhood function of**

type M is a function whose domain is the two-dimensional cartesian product of S with itself a total of $1 + 2 \cdot d \cdot (d + 1)$ times (arranged in the shape of a d-neighborhood) and whose range is S.

Theorem 2.1 (Interdependence)

Let M be a cell type and d a positive integer, then there exists a d-neighborhood function of type M, say f, such that if $\mathcal{D} = D^0, D^1, D^2, \ldots$ is any computation of type M, t is any non-negative integer, and c is any cell, then the state of c at time t + d can be obtained by applying f to the d-neighborhood of c at time t.

PROOF: The result is immediate for d = 1 by letting f be the transition function of cell type M.

Assume by induction that it holds for all $d < n$ and let $d = n$. Then for each $c' \in N_1(c)$ we have $N_{d-1}(c') \subset N_d(c)$ by the triangle inequality. Hence by induction the state of cell c' at time t + d - 1 is a function of the states at time t of all cells in $N_d(c)$, for all $c' \in N_1(c)$. But by definition the transition function gives the state of cell c at time t + d as a function of the states at time t + d - 1 of all $c' \in N_1(c)$. Composing these functions gives the desired function f. □

The Interdependence Theorem is fundamental and has several important applications. It can be viewed in two different ways. As

stated, it allows one to make predictions about the future state of a cell given sufficient current information about the neighbors of that cell. It will be used in this guise to prove the Speed-Up Theorem (Theorem 2.2 below). Viewed in another way, the Interdependence Theorem states that two cells which are at a distance of d + 1 from each other cannot influence each other's behavior for the next d units of time. In this form it is simply the statement that signals propagate at unit distance per unit time, but avoids mentioning signals as such. In this latter form it may be used to establish minimal time results, such as the following.

Corollary 2.1.1

$\psi_{PAR}$ is not recognizable within time less than

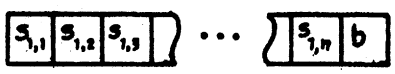
$$T_{MIN}(m,n) = \begin{cases} m + n - 1 \text{ if either } m = 1 \text{ or } n = 1 \\ m + n - 2 \text{ if } m,n \geq 2. \end{cases}$$

PROOF: Let M be a cell type which recognizes $\psi_{PAR}$.

First consider the case where m = 1. Let $D^0$ be an initial description, say

$D^0 =$


Now create the initial description $E^0$ by adding a black square to the eastern end of $D^0$. Thus

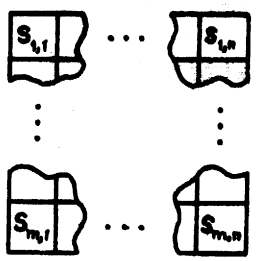
$E^0 =$ 


Now the cells in the (n-1)-neighborhood of cell (1,1) have the same

states in both $D^0$ and $E^0$. Hence by the Interdependence Theorem, cell (1,1) will be in the same state in both $D^{n-1}$ and $E^{n-1}$. But since M recognizes $\psi_{PAR}$ and since $D^0$ and $E^0$ have opposite parity, this state cannot be a final state. Thus M takes at least $n + m - 1 = n$ units of time to recognize $\psi_{PAR}$ when $m = 1$.

The case where $n = 1$ is similar.

Finally assume that $m,n \geq 2$. Consider a typical initial description.




Note that the southeast corner cell (m,n) is at a distance $m + n - 2$ from cell (1,1) and hence by the Interdependence Theorem cell (1,1) is independent of the state of cell (m,n) for the first $n + m - 3$ units of time. Hence since $\psi_{PAR}$ depends on the state of cell (m,n), we have that M requires at least $m + n - 2$ units to recognise $\psi_{PAR}$. □

Note that the only property of $\psi_{PAR}$ which we used in the above proof was that it depended on the initial state of cell (m,n). Thus we have actually proved the following.

Corollary 2.1.2

> No reasonable predicate, $\psi$, can be recognized in time less than $T_{MIN}$, where by reasonable we mean that for any $m,n \geq 1$ there exist two figures P and P' which differ only in their (m,n) entry and such that

$P \in \Psi$ and $P' \notin \Psi$

The reader with an eye for detail will note that the cell type $M_{PAR}$ constructed earlier actually achieves the theoretical minimum timing only for the cases n = 1 or m = 1 and it is one unit slower than the minimum for the cases where $m,n \geq 2$. This minor defect can be remedied by modifying $M_{PAR}$. The smallest number of states the author has found for such a modified machine is seven. Details are left to the reader.

## 2.4 Speed-Up

We now turn to the Speed-Up Theorem. The central idea behind this theorem is that by packing information into fewer cells in an array, the information can be processed at a higher rate since the amount of time it takes for a signal to travel from the location of one piece of information to the location of another has been reduced. This idea has occurred to many people and uses of it may be found in the papers of Cole[6], Fischer[7], and Hennie[8]. In their formulations no information is initially present in the computer and hence the packing can be done as the information is input to the computer. In this way they achieve any desired degree of speed-up without having to pay a price in processing time, although they do increase the number of states per cell. In our formulation, the information to be processed is initially present in the array and some time must be spent in packing it into a smaller area within the array. Thus we must pay a price in both time and states to achieve a speed-up.

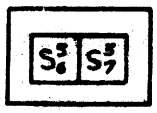
One-Dimensional Case: As a warming up exercise for the two-dimensional Speed-Up Theorem, we first consider a one-dimensional example. Assume that we have set up some figure in a 1×9 array of cells of type M and let us observe its operation for a few time units.

$D^0 =$ | $S_1^0$ | $S_2^0$ | $S_3^0$ | $S_4^0$ | $S_5^0$ | $S_6^0$ | $S_7^0$ | $S_8^0$ | $S_9^0$ |

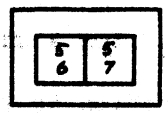
$D^1 =$ | $S_1^1$ | $S_2^1$ | $S_3^1$ | $S_4^1$ | $S_5^1$ | $S_6^1$ | $S_7^1$ | $S_8^1$ | $S_9^1$ |

$D^2 =$ | $S_1^2$ | $S_2^2$ | $S_3^2$ | $S_4^2$ | $S_5^2$ | $S_6^2$ | $S_7^2$ | $S_8^2$ | $S_9^2$ |

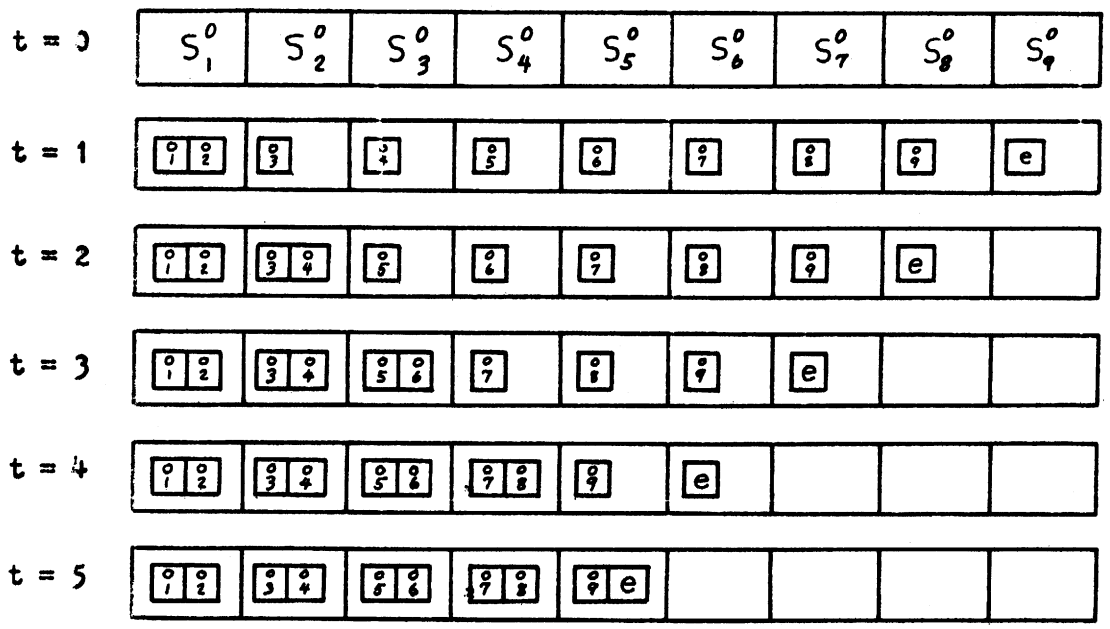
The symbol $S_j^t$ of course represents the state of cell (1,j) at time t. We are going to describe a cell type $M_2$, where each cell must have the ability to simulate two cells of type M. Our notation should be transparent. Namely




or more simply




represents one cell of type $M_2$ which is currently in a state representing two cells of type M, one of which is in state $S_6^5$ and the other of which is in state $S_7^5$. Extensions of this notation will be introduced without further comment. Since we will be talking about both M cells and $M_2$ cells at the same time, we will find it convenient to refer to the former as cells and the latter as modules. For example, we say that each module in an $M_2$ array simulates two cells. Now let us observe the operation of a 1×9 array of type $M_2$ when started with the same figure. The first five steps of this operation are shown in the diagram on the next page.

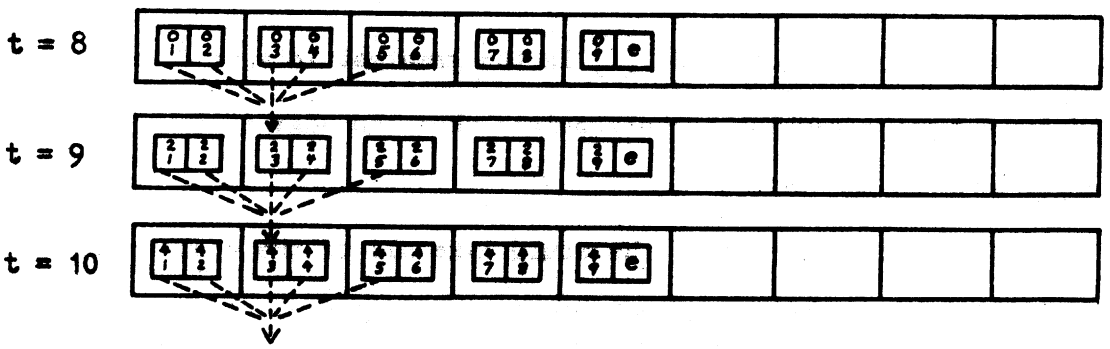



At this point, the $M_2$ array has packed the original figure by a factor of two and is ready to begin processing the figure at a rate exactly two times that of the M array. Note that the packing process takes $n - \left[\frac{n}{k}\right]$ time units in general, where n is the length of the array, k is the packing factor, and $[\cdot]$ is the greatest integer function. (We consider the packing to be complete when the edge state e can no longer move to the left.)

One problem now arises. We would like all of the modules in the packed portion of the array to begin their simulation simultaneously. This is accomplished by using a firing squad procedure such as that described in Balzer[3]. The last module to be packed (module (1,5) in our example) acts as the general of a firing squad with the soldiers being the modules to the left of the general. The modules in the firing squad retain their packed information on one level while

carrying out the firing squad process on another level. When the firing squad goes off, the simulation begins. Note that the firing squad process, if carried out as described above, will take $2 \cdot \left[\frac{n}{k}\right]$ units to begin simulation. Thus the simulation gets under way at time $t = n + \left[\frac{n}{k}\right]$. The reader who is fond of firing squad problems may show how this time can be reduced to $t = n$ by beginning some firing squad activity at time $t = 0$ instead of waiting for the packing to be completed. Note that $t = n$ is the earliest the simulation can begin, since by the Interdependence Theorem module (1,1) cannot begin simulation before it is aware of the existence of the right-hand end of the array and this cannot happen before $t = n$. We will omit the details of the firing squad mechanism, since an alternate method for synchronization will be given later.

Let us assume then that at time $t = 9$ our array begins simulation. The process will look as follows.




Dotted arrows have been drawn to show how module (1,2), which is responsible for simulating cell (1,3), has access to all of the information necessary for predicting the state of cell (1,3) two units into the future via the Interdependence Theorem. In general with a packing factor of k, one obtains a simulation which is faster

by a factor of k.

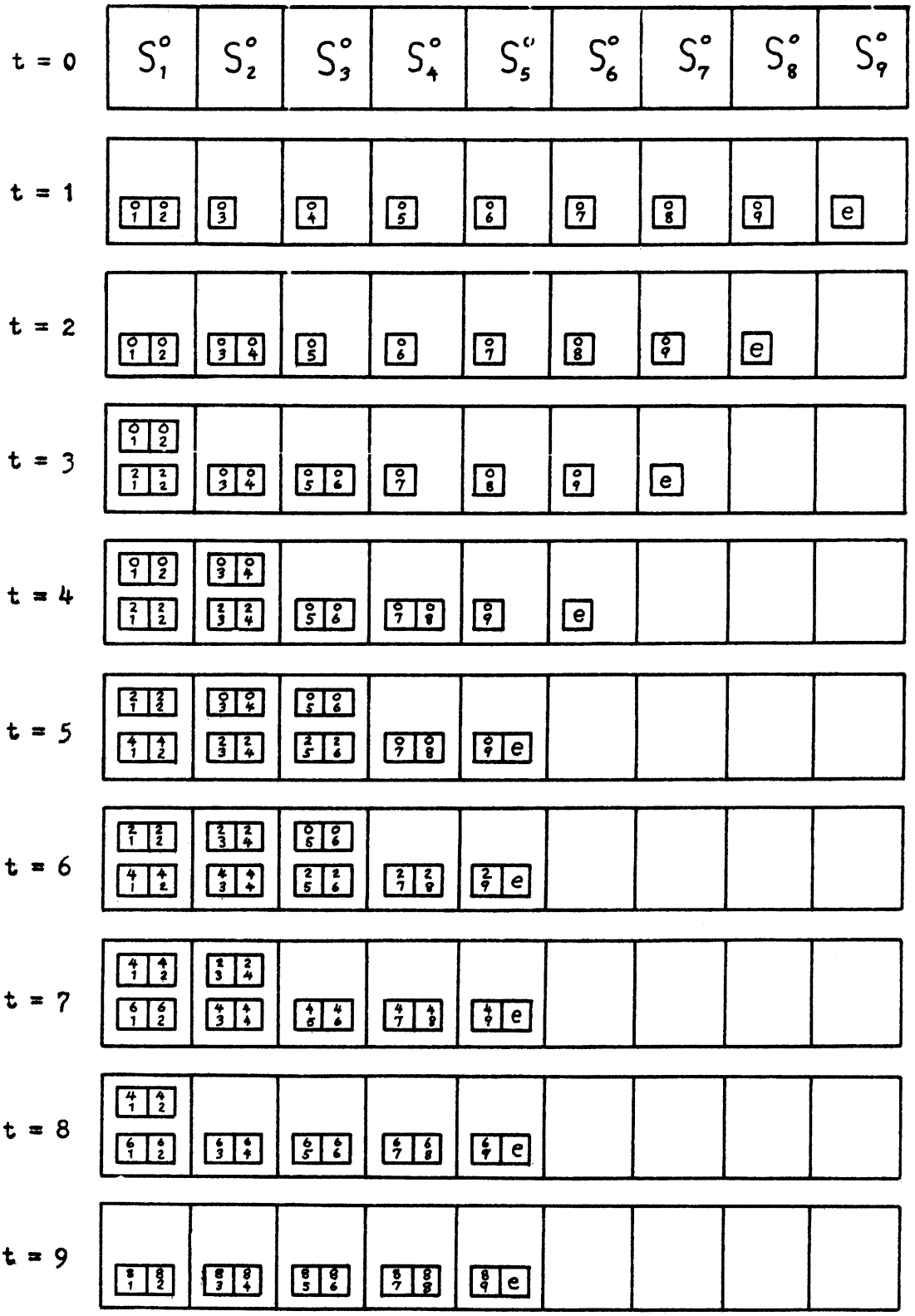
The simulation continues until module (1,1) senses that cell (1,1) is about to enter a final state. Instead of simulating this entry, module (1,1) actually enters that final state itself. Thus the $M_2$ array accepts or rejects the initial figure according as the M array would have accepted or rejected it.

Since simulation begins at step m, recognition takes place at time $t = n + \left\lceil \frac{T(m,n) - 1}{k} \right\rceil$, where T(m,n) is the time in which M carries out its recognition. The number of states required is approximately $8 \cdot s^k$ where s is the number of states of cell type M and where we assume one possesses an eight state solution to the firing squad problem.

Progressive Synchronization: Before turning to the two-dimensional case, we present a second solution to the synchronization problem which does not use the firing squad. This second solution is slightly faster than the firing squad method, but uses approximately $2 \cdot s^{2k}$ states per cell. We present it here because it is of interest in its own right. Let us call it the method of progressive synchronization.

In progressive synchronization the simulation begins to take place while the packing operation is still under way. Each module carries out a simulation step as soon as the necessary information becomes available to it. Modules which begin to simulate before packing is complete carry out their simulation at a reduced rate because of limited information availability. The last module to be packed immediately begins simulation at full speed and eventually catches up with the modules which began simulation earlier. Larger and larger blocks of modules become synchronized until at last the

entire array is synchronized and simulating at full speed.



The diagram on the preceding page illustrates progressive synchronization as it would be used in our example. From t = 9 on simulation proceeds as in the firing squad case.

Each module in the above diagram must actually have one more bit of information so that it can indicate to its neighbors when it has made a simulated state transition. We have omitted this bit from the diagrams above.

Observe how module (1,2) carries out a simulation step each time the necessary information becomes available in the neighboring modules. Also observe how the last module to be packed, module (1,5), is able to begin simulation at full speed as soon as it is packed. Since this module is packed at time $t = n - \left[\frac{n}{k}\right]$, the recognition will take place at time $t = n - \left[\frac{n}{k}\right] + \left[\frac{T(m,n) - 1}{k}\right] + 1$ or before. The case where recognition takes place sooner is due to the fact that module (1,1) is several simulation steps ahead for a while.

To recapitulate, we can create, using firing squad techniques, a cell type which performs recognition in time $t = n + \left[\frac{T(1,n) - 1}{k}\right]$ and which has on the order of $s^k$ states. By using the method of progressive synchronization, we can create a cell type which performs recognition in time $t = n + 1 - \left[\frac{n}{k}\right] + \left[\frac{T(1,n) - 1}{k}\right]$ and has on the order of $s^{2k}$ states.

The method of progressive synchronization introduced above belongs to a class of iterative programming techniques which we shall call <u>methods of non-uniform simulation</u>. These methods all have the following characteristics in common:

1) they involve simulation of one array by another,

2) a module in the simulator may keep multiple copies of the cells that it is simulating, and

3) simulated time may be distorted, that is, at any given moment modules which are spatially separated in the simulating array may be working at different points in simulated time.

We will use a method of non-uniform simulation in the proof of Theorem 2.5.

Speed-Up: We now turn to the two-dimensional Speed-Up Theorem. The statement of the theorem could be sharpened in several ways. Preferring the more simple and workable statement for our main result, we will leave the sharpening to a corollary.

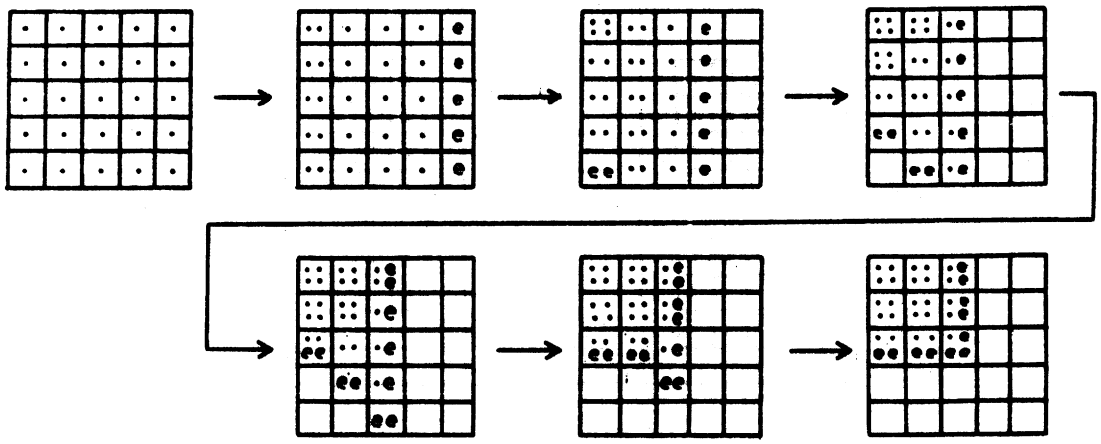
Theorem 2.2 (Speed-Up)

Let $\psi$ be a predicate and let M be a cell type which recognizes $\psi$ in time T(m,n). Then given any positive integer k, there exists a cell type $M_k$ which recognizes $\psi$ in time $\frac{1}{k}.T(m,n) + m + n + 2$.

PROOF: The two-dimensional case is quite similar to the one-dimensional case. We perform packing by a factor of k followed by or overlapped with simulation at a rate k times faster than the original array. Again we have our choice of using a two-dimensional firing squad or of using a method of progressive synchronization.

Packing: Packing is accomplished by performing simultaneous packing within each row and when all the cells in a given column are

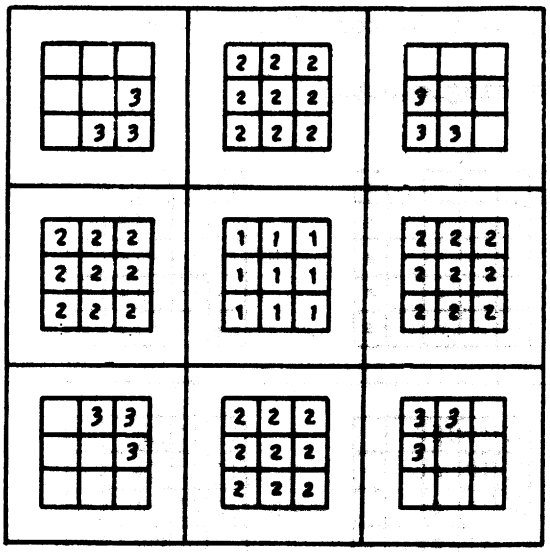
fully packed, the column is then packed. The following diagram illustrates packing by a factor of two, where each simulated cell is represented by a single dot.



Packing is completed in $m - \left[\frac{m}{k}\right] + n - \left[\frac{n}{k}\right]$ units of time. Assuming that progressive synchronization is used, simulation gets under way immediately.

Simulation: In considering two-dimensional simulation, we find a problem which wasn't present in the one-dimensional case. It is due to the fact that we do not allow diagonal neighbor connections in our model.

Consider a portion of an $M_3$ array attempting to simulate an M-array at a rate three times faster than normal.



The central module in the above diagram has responsibility for simulating the cells labeled 1. In order to advance the states of these cells by three units, the central module must have access to the cells labeled 2 and 3. Because diagonal connections are prohibited, access is available only to those cells labeled 2.

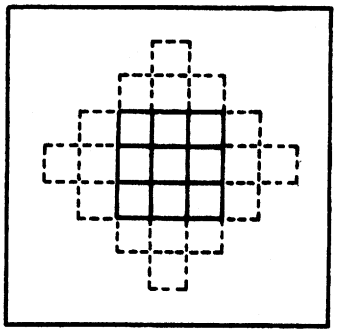
There are several ways around this problem. One is to allow diagonal connections between cells. In that case, assuming diagonal connections were also allowed in the M-array, the central module in the diagram above would require access to all simulated cells shown. Thus it would actually have access to exactly the information needed to advance the states of the cells for which it is responsible.

Within the restriction of nearest neighbor interaction, we could simply pack by a factor of six and then use two steps of real time to allow each module to get information around the corner from its diagonal neighbor and advance the states of the cells for which it is

responsible by six steps. Thus an average speed-up of three would result. The cost in terms of states, however, would be high. In order for each module to pass information around the corner during the intermediate step, it would have to have provisions in its memory for remembering additional simulated cells. The smallest number of additional simulated cells required seems to be $\frac{3}{2} \cdot k^2 - k - 1$. Thus in the case above where $k = 6$, we would require each module in the simulator to have memory capacity for holding eighty-three simulated cells, just to provide an average speed-up factor of three. When compared with the nine simulated cells required per module in the diagonal connection case, the cost of this method seems excessive.

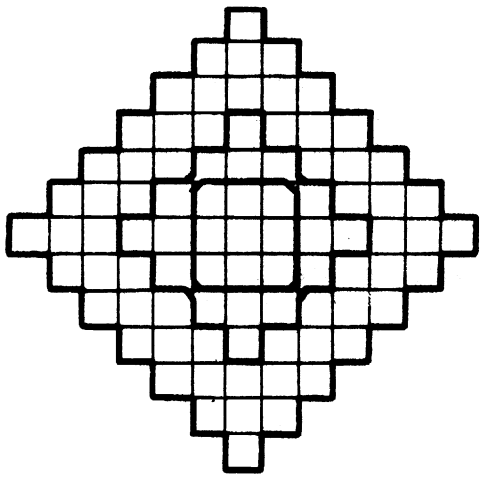
A more reasonable method is now described which requires twenty-five simulated cells per module to produce a speed-up of three. It is apparent that a module, if it is to advance the cells for which it is responsible, needs access to information about the states of the cells for which its diagonal neighbor is responsible. Instead of having this information computed by the diagonal neighbor and then passed around the corner via a nearest neighbor, we let the nearest neighbor compute the information directly, thus making it available one time unit sooner.

Consider the diagram on the following page which represents one module in the simulator array.



The module contains twenty-five simulated cells. The central nine, which are drawn with solid lines, are the same as those for which it would be responsible under the earlier scheme. We say that these are the cells for which it is primarily responsible. One thinks of these primary cells as being the ones which are "really" being simulated and that the sixteen secondary cells are simulated merely as a convenience for the benefit of the module's nearest neighbors.

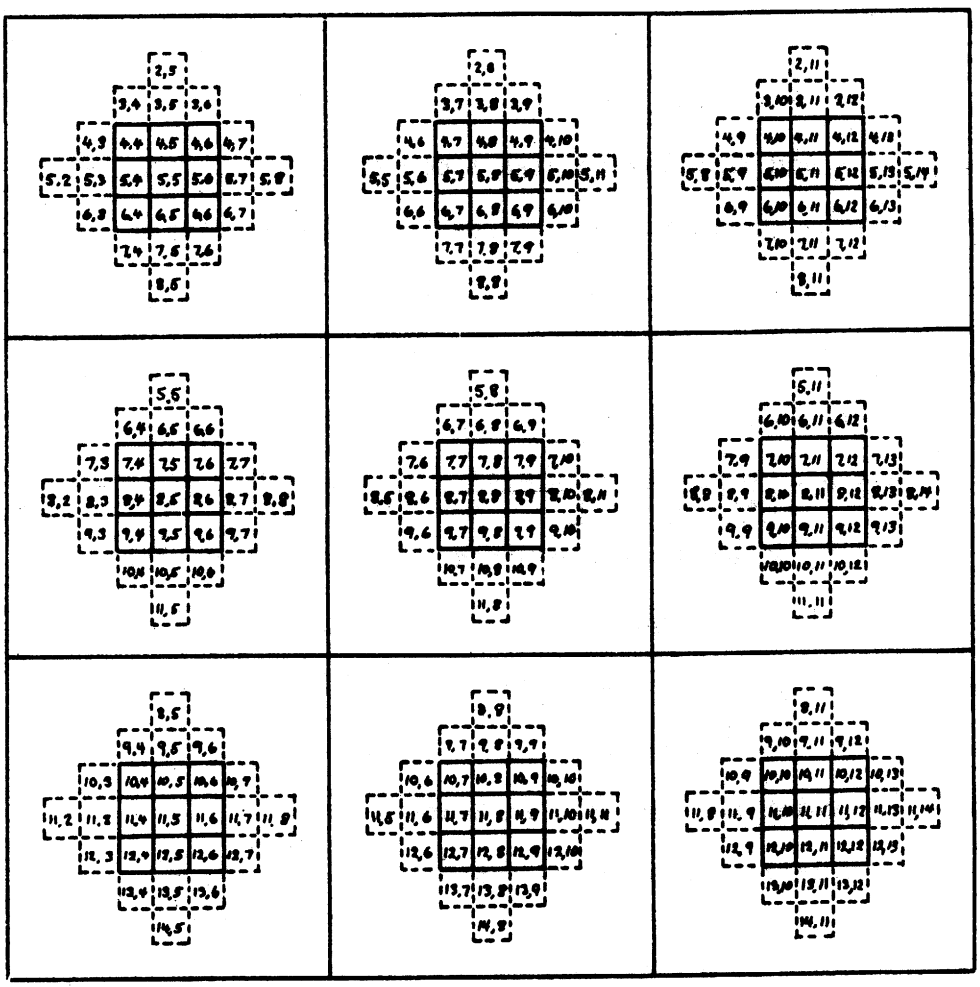
The following diagram shows a portion of an array which is to be simulated.



Three concentric heavy outlines have been drawn. The innermost outline contains a set of nine cells for which a module is primarily

responsible. The middle outline contains the twenty-five cells for which this module is both primarily and secondarily responsible. The outer outline shows the set of cells to which the module has access via its nearest neighbors at any given step. Note that the cells contained in the outer outline are exactly those cells to which access is required for a speed-up of three.

The following diagram presents the situation from the point of view of the simulating array.



The preceding diagram shows a 3x3 set of modules and within each the simulated cells for which it is responsible. Each simulated cell has been labeled with its row and column number. Some cells such as (8,8) are simulated by as many as five different modules. All are simulated by at least three modules.

Of the many methods of simulation considered by the author, the above method requires the fewest states for the case of nearest neighbor interaction. It requires $2\cdot(k^2 + k)$ states (for k even) and $2\cdot(k^2 + k) + 1$ states (for k odd).

Conclusion: We have all the pieces necessary to construct a cell type $M_k$, given M and k. To recapitulate, $M_k$ first performs packing by a factor of k and then begins high speed simulation. Simulation is begun either by the method of progressive synchronization or by the use of the two-dimensional firing squad which is discussed in Section 4.3.

The method of progressive synchronization requires on the order of $s^4\cdot(k^2 + k)$ states per cell and recognition is completed on or before time $t = m + n - \left[\frac{m}{k}\right] - \left[\frac{n}{k}\right] + \left[\frac{T(m,n) - 1}{k}\right] + 2 \leqslant \frac{1}{k}\cdot T(m,n) + m + n + 2$.

The firing squad method requires on the order of $s^2\cdot(k^2 + k)$ states per cell and recognition is completed on or before time $t = 2(m + n) - \left[\frac{m}{k}\right] - \left[\frac{n}{k}\right] + \max\{m,n\} + \left[\frac{T(m,n) - 1}{k}\right]$. The details of the latter formula follow from a knowledge of the two-dimensional firing squad which is discussed in Section 4.3.

This completes our proof of the Speed-Up Theorem. □

We now state as a corollary to the above proof the complete statement of the Speed-Up Theorem.

Corollary 2.2.1

There exists a known effective procedure which, when given a positive integer k and a cell type M, produces a cell type $M_k$ such that

i) M and $M_k$ have the same initial and final states

ii) any m×n figure P which is recognized by M in time t is recognized by $M_k$ in time $m + n - \left[\frac{m}{k}\right] - \left[\frac{n}{k}\right] + \left[\frac{t-1}{k}\right] + 2$.

In the proof of the Speed-Up Theorem, we mentioned the idea of having packing going on in one layer of the array and a firing squad going on in another layer of the array. This conceptual trick of resolving the processing of an array into several semi-independent but simultaneous processes and of picturing them as taking place in different layers of the array is simple but powerful. As an application, we present

Theorem 2.3 (Minimizing)

Let $\psi$ be a predicate and let $M_1$ and $M_2$ be cell types which recognize $\psi$ in times $T_1(m,n)$ and $T_2(m,n)$ respectively. Then there exists a cell type M which recognizes $\psi$ in time $T(m,n) = \min\{T_1(m,n), T_2(m,n)\}$.

PROOF: Let M consist of two layers. One layer behaves like $M_1$, the other like $M_2$. Cell (1,1) goes to its final state as soon as either

layer completes its recognition. Since both layers recognize the same predicate, there can be no conflict. □

## 2.5 Computing Power

We now turn to the study of the class of predicates which can be recognized by arrays.

**Definition**

A predicate $\Psi$ is said to be a cellular predicate if there exists a cell type M which recognizes $\Psi$. A predicate $\Psi$ is said to be a finite predicate if it contains only a finite number of figures.

**Theorem 2.4** (Boolean/Finite)

The class of cellular predicates forms a Boolean algebra and contains all finite predicates. Furthermore, given two cellular predicates $\Psi_1$ and $\Psi_2$ which are recognizable in time $T_1(m,n)$ and $T_2(m,n)$ respectively, then both $\Psi_1 \cup \Psi_2$ and $\Psi_1 \cap \Psi_2$ are recognizable in time $\max\{T_1(m,n), T_2(m,n)\}$.

PROOF: Proof of the closure of the class of cellular predicates under the Boolean operations of union, intersection, and complementation is directly analogous to the proof of this property for regular events. Layers are used as the method of combining two distinct cell types.

To prove the second statement, let $\Psi$ be any finite predicate and let k be the smallest integer such that all figures in $\Psi$ are

k×k or smaller. Construct a cell type which packs by a factor of k + 1. As soon as cell (1,1) is packed it will have all of the information necessary to accept or reject the figure since any figure larger than k×k must be rejected and since there are only a finite number of figures k×k or smaller. The entire process will be completed by at least time t = 2k - 1.

In recursion theory one finds that there are sets which are recursively enumerable but not recursive. That is there are sets which can be accepted (or rejected) by Turing machines, but which cannot be recognized by Turing machines. The following theorem shows there is no analogous situation for arrays.

Theorem 2.5

Let $\Psi$ be a predicate and let M be a cell type which accepts (or rejects) $\Psi$. Then $\Psi$ is a cellular predicate and a cell type M' which recognizes $\Psi$ can be effectively constructed from M.

PROOF: Let us assume that M accepts $\Psi$, the case where M rejects $\Psi$ being similar. We will give two different ways in which the cell type M' may be constructed.

(Method 1). Let an M' array consist of two layers. The bottom layer acts as M does and may eventually accept or reject $\Psi$, in which case M' does also. The top layer acts as a counter with a capacity of $(s - 1)^{mn}$, where s is the number of states of cell type M. (We use s - 1 rather than s since the edge state e does not matter for

the present purposes.) When the counter reaches the value $(s-1)^{mn}$, a signal is sent to the cell (1,1) which rejects the figure if it has not already been recognized. The basis for this method is that the entire m×n array of type M, which is being simulated in the bottom layer, has only $(s-1)^{mn}$ states and hence must be in a loop if it has not recognized the figure by time $t = (s-1)^{mn}$. It is assumed that the reader has no difficulty in seeing how to orgainize the top layer of M' into a counter of capacity $(s-1)^{mn}$.

(Method 2a). In this method we will detect possible looping of the M array by having two layers in the M' array each of which behaves as an M array. One layer runs at half the speed of the other. The two layers will be in identical states at time t = 0, but the slower layer will immediately fall behind. When the two layers again achieve identical states, we know that they must be in loops, since the faster layer must be entering this state for at least the second time. When such a looping condition is detected, the figure may be rejected if it has not already been recognized.

The problem here is how to detect when the two layers are in the same state. One straight forward but time consuming method would be to have a third layer which observes and controls the first two. The third layer would cause the other two to advance by one and two steps respectively and would then inhibit their action. Each cell of the third layer would then compare the states of the corresponding cells in the other two layers and would generate a signal indicating whether or not they were in the same state. These signals would be accumulated at one point, say cell (1,1), in much the same manner as the

parity information was accumulated by $M_{PAR}$. Cell (1,1) would then decide whether the figure should be rejected or whether another step in the simulation should be carried out. If the latter is the case, a firing squad could be started to initiate another step in the simulation. This method clearly burns up a lot of time, since the accumulation portion takes about m + n steps and the firing squad takes another $m + n + \max\{m,n\}$ steps. A much faster method is now described. It will be more expensive in terms of states required.

(Method 2b). We will use a method of non-uniform simulation. The basic idea is that waves of simulation spread out from the southeast corner. Ahead of each wave the two layers are at simulated times $t_1$ and $t_2$. After a wave passes, they are at simulated times $t_1 + 1$ and $t_2 + 2$. Between the waves of simulation come comparison waves which accumulate the necessary comparison information. As each comparison wave washes over cell (1,1), all the information necessary to decide whether or not a loop as been entered is present and cell (1,1) can make its decision. The speed of the method comes from the fact that one wave may follow immediately behind another. □

One way of studying the class of cellular predicates is to compare it to the classes of predicates recognizable by other types of recognition devices such as the perceptron. We have already seen one cellular predicate, namely $\Psi_{PAR}$, which is not in the class of predicates recognizable by order or diameter limited perceptrons. Other predicates which have been considered by Minsky and Papert[10] in connection with the perceptron will be discussed in Chapter 3.

Hennie [8] has classified arrays by the number of directions in which signals may flow within the array and then compared the relative computing power of various classes.

Other types of recognition devices may be obtained by modifying normal one-dimensional tape-accepting devices to operate on two-dimensional tapes. Consider for a moment a universal computer which is able to walk about on a two-dimensional tape, sense the edges without stepping off the tape, and which can read the symbols written on the tape. (A two counter machine provides a more tidy mental image of this process than a Turing machine, since the latter must drag its tape behind and is always in danger of tripping over it.) We will use the phrase two-dimensional universal computer to describe such a device. Given a figure represented on a tape, the machine can wander about on the figure and eventually accept or reject the figure. The phrases two-dimensional finite state automaton, two-dimensional push-down automaton, and two-dimensional linear bounded automaton describe similar adaptations of one-dimensional devices.

Blum and Hewitt[5] introduced a special class of two-dimensional devices called pebble automata. These devices are just two-dimensional finite state automata, which are provided with a fixed finite number of markers (called pebbles) which they carry about with them and leave on squares as temporary markers. Upon returning to a square on which a marker had been previously placed, the automaton can sense its presence, pick it up, and carry it off for further use if so desired.

**Definition**

Given two classes $C_1$ and $C_2$ of devices for recognizing predicates, we say that $C_1$ *is strictly more powerful than* $C_2$ provided that the class of predicates recognized by devices in $C_2$ is a proper subclass of the class of predicates recognized by devices in $C_1$. If these classes are the same, we say that $C_1$ *is equivalent in power to* $C_2$. Note that the relation thus defined induces a partial ordering on the family of classes of computing devices.

**Theorem 2.6** (Non-Universality)

Two-dimensional universal computers are strictly more powerful than iterative arrays.

PROOF: Certainly a universal computer can recognize anything which can be recognized by an iterative array. The existence of a predicate which is recognizable by a universal computer, but which is not a cellular predicate can be obtained by a diagonalization argument as follows. Set up some fixed effective method for coding descriptions of cell types into figures. Then consider the predicate $\psi_{DIAG}$ given as follows:

$$P \in \psi_{DIAG} \iff \left\{ \begin{array}{l} \text{P is a coded description of a cell} \\ \text{type M such that an array of type M} \\ \text{would reject P.} \end{array} \right.$$

Now $\psi_{DIAG}$ is certainly an effectively computable predicate. Assume it is a cellular predicate and let M' be a cell type which recognizes it.

Let P' be the coded description of M'. Then $P' \in \psi_{DIAG} \iff M'$ rejects P' by definition of $\psi_{DIAG}$. But M' rejects $P' \iff P' \notin \psi_{DIAG}$ since M' is assumed to recognize $\psi_{DIAG}$. This contradiction shows that $\psi_{DIAG}$ is not cellular. □

Before stating the next two theorems, let us describe the model of two-dimensional linear bounded automata which we will use. We could assume that such a device has a one-dimensional auxiliary working tape which is bounded in length by a linear function of the number of squares in the input figure. We will, however, take the more natural approach that the device has no auxiliary tape, but rather uses the input figure itself as a tape and is able to both read and write on the figure with a finite set of symbols of which the initial figure's symbols may be only a subset. This model seems much more natural in the current context. We also assume that the automaton is always started in its initial state on square (1,1).

Theorem 2.7

Two-dimensional linear bounded automata are strictly more powerful than pebble automata.

PROOF: Using the model of a linear bounded automaton(LBA) described above, it is easy to see how, given any pebble automaton (PA), we can construct an LBA to simulate it (in fact to simulate it in real time).

To show that LBA's are strictly more powerful than PA's, we carry out the same diagonalization argument found in the proof of Theorem 2.6,

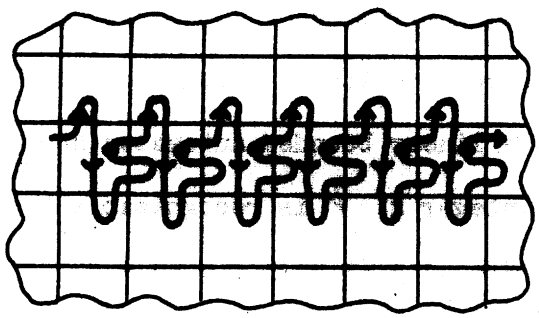
replacing universal computer by LBA and iterative array by PA. There are some things to check. First one must check that a coding of PA's can be produced such that a single LBA can look at a coded description of any PA and then simulate that PA operating on its own description. Second since a PA may begin looping rather than accepting or rejecting a figure, we must ensure that the LBA can detect such loops. This can be done by an adaptation of the second method of detecting loops in the proof of Theorem 2.5. Namely two copies of the PA are simulated, one copy at one rate and the other, at half the rate. In between simulated steps the LBA checks to see if the two simulated PA's are in the same location and same state and if they have their pebbles all arranged in the same manner. This must eventually happen if the PA's enter a loop and hence the LBA can determine if the PA described has rejected its own description by looping. □

Theorem 2.8

Iterative arrays are equivalent in power to two-dimensional linear bounded automata. Indeed, given any predicate $\psi$ which is recognizable in time $T(m,n)$ by a linear bounded automaton and given any positive integer $k$, $\psi$ is recognizable by an array in time $\frac{1}{k}\cdot T(m,n) + (1 + \frac{1}{k})(m + n) + 2$.
Conversely, given any predicate $\psi$ which is recognizable in time $T(m,n)$ by an array and given any positive integer $k$, $\psi$ is recognizable in time $4mn(\left\lceil \frac{1}{k} T(m,n) \right\rceil + 1)$.

PROOF: It is clear that an array can simulate a linear bounded automaton(LBA) in real time. However, the LBA can make its decision about the figure from any position on the figure whereas the array must deliver its answer to cell (1,1). Thus if the LBA takes time T(m,n) to recognize the figure, then an array might take as long as T(m,n) + m + n - 2. Given any positive integer k, we have by the Speed-Up Theorem that $\psi$ can be recognized by an array in time $\frac{1}{k}\cdot(T(m,n) + m + n - 2) + m + n + 2 \leq \frac{1}{k}\cdot T(m,n) + (1 + \frac{1}{k})(m + n) + 2$.

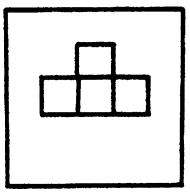
It is also not difficult to see how an LBA might simulate an array. It could use a working alphabet which would allow it to represent two different cell states within one square of the figure. One of these states would represent the state of the cell at the "current time" and the other the state of the cell at the "next time." By making a pass over the array it would be able to update the states of each cell. The first method of passing over the array which one considers usually involves tracing out a path as indicated in the following diagram, which shows the LBA making a scan of one row.



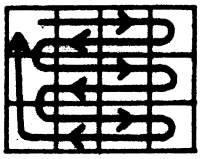
This is the path traced out by an LBA which examines each of the neighbors of a cell in turn so that it can update the state of the cell. The black dots in the diagram above represent the points at

which the LBA has accumulated enough information to update a cell. Note that the LBA makes seven movements per cell.

If instead of attempting to update the state of each cell on each pass over the array, we allow several preparatory passes, we can reduce the average number of movements per cell to four. In this case the LBA keeps four simulated cells within each square of the tape. One represents the cell located at that position and the other three represent its western, northern, and eastern neighbors.

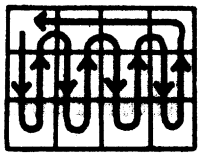


The LBA begins in the northwest corner and describes the following path.



During this pass each square can be marked to represent the states of its western and eastern neighbors.

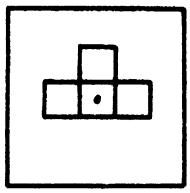
The LBA next describes the following path.



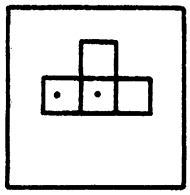
During this pass each square is marked with the state of its northern neighbor upon the first visit from the LBA and upon the second visit

the LBA, which is coming up from the south, has all the information necessary to advance the state of the cell by one time unit.

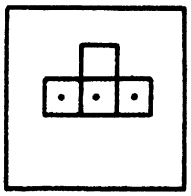
An observer stationed at a given cell in the interior of the array would note the following build up of information where a dot represents information about the current state of a cell.



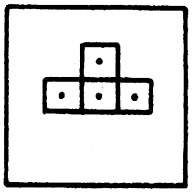
Before first visit by the LBA.



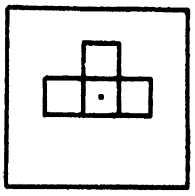
After first visit.



After second visit.



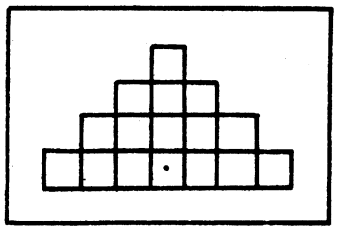
After third visit.



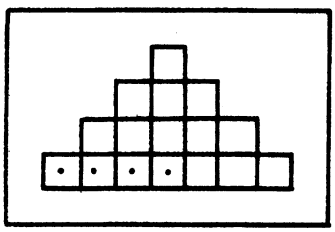
After fourth visit.

(The state of the cell has now been advanced by one simulated time unit.)

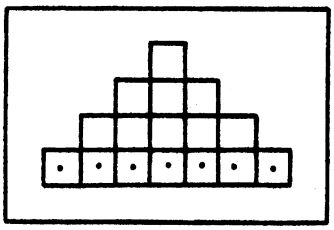
The entire process involves an average of four movements per cell per unit simulated time. But given any positive integer k, we can simulate not just one time unit per pass, but k time units per pass simply by increasing the information deposited in each square by the LBA on its first three visits to the square. Of course the LBA must carry along more local information in this case. For example, if k = 3, then the information capacity of each square would be sixteen cells and our observer would note the following sequence of events.



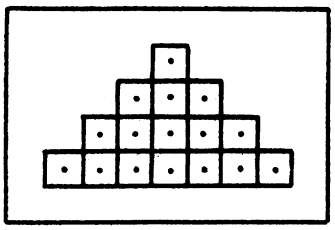
Before first visit by the LBA.



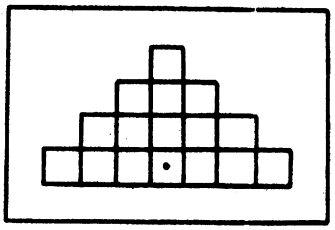
After first visit.



After second visit.



After third visit.



After fourth visit.

(The state is now advanced by three simulated time units.)

The timing result stated in the theorem now follows immediately. □

Corollary 2.8.1

Iterative arrays are strictly more powerful than pebble automata.

PROOF: Immediate from Theorems 2.7 and 2.8. □

Let us return for a moment to the idea of a two-dimensional universal computer.

Definition

A predicate $\Psi$ is said to be effectively recognizable if there exists a two-dimensional universal computer which recognizes $\Psi$. We say we are given an effectively recognizable predicate $\Psi$ provided we are given a finite description of a universal computer which recognizes $\Psi$.

We now state some undecidabiltiy results. Note that Theorem 2.6 proved the existence of a non-cellular effectively recognizable predicate.

Theorem 2.9

Given an effectively recognizable predicate $\Psi$, it is in general undecidable whether or not $\Psi$ is a cellular predicate.

PROOF: By Theorem 2.6 there exist effectively recognizable predicates which are not cellular. Let $\Psi'$ be such a predicate. Now given any Turing machine T, let $\Psi_T$ be given by

$$P \in \Psi_T \iff \begin{cases} P \in \Psi' \text{ and } P \text{ is an } m \times n \text{ pattern and } T \\ \text{doesn't halt in } m \cdot n \text{ steps when started} \\ \text{on a blank tape.} \end{cases}$$

Now $\Psi_T$ is certainly effectively recognizable. Furthermore $\Psi_T$ is finite $\iff$ T eventually halts on a blank tape, by definition of $\Psi_T$. But $\Psi_T = \Psi' \iff$ T doesn't halt on a blank tape, again by definition of $\Psi_T$. Since finite predicates are cellular by Theorem 2.4 and since $\Psi'$ is not cellular, we have that $\Psi_T$ is cellular $\iff$ T halts on a blank tape. The theorem follows by the undecidability of the halting problem for Turing machines. □

Theorem 2.10

Given a cellular predicate $\Psi$ and a cell type M, it is in general undecidable whether or not M recognizes $\Psi$.

PROOF: Let T be a Turing machine and M' be a cell type which recognizes $\Psi$. Construct a cell type $M_T$, which first simulates T starting on a blank tape for m steps or until T makes an excursion of

more than n units from its starting point. (A similar construction based on the Post correspondence problem rather than on Turing machines may be found in Chapter 3 of Hennie[8].) If T has not halted by the time the simulation ends, $M_T$ then simulates M' and accepts or rejects $\psi$ according as M' does. If T has halted, then $M_T$ simulates M' and makes the opposite classification from the one M' would have made. Thus $M_T$ recognizes $\psi \iff$ T never halts. The theorem follows. □

Definition

Two cell types are said to be equivalent if they recognize the same predicate.

The following corollaries due to Hennie[8] are immediate from the above theorems. Hennie shows the corollaries hold even if one assumes that signals can only travel from east to west and from south to north within the array.

Corollary 2.10.1 (Hennie)

Equivalence of cell types is undecidable.

Corollary 2.10.2 (Hennie)

Given a cell type M, it is undecidable whether or not it accepts any figures.

## 2.6 Linear Predicates

We now define an important class of predicates.

Definition

A predicate $\Psi$ is said to be linear if there exist non-negative integers p, q, and r such that $\Psi$ is recognizable within time pm + qn + r.

In view of Corollary 2.1.2, we might be justified in saying that the following theorem shows that linear predicates can be recognized "almost" as fast as any predicate.

Theorem 2.11

If $\Psi$ is a linear predicate, then for any real $\varepsilon > 0$, $\Psi$ is recognizable within time $(1 + \varepsilon)(m + n) + 2$.

PROOF: Let p, q, and r be such that $\Psi$ is recognizable within time pm + qn + r. Let $s = \max\{p + r, q + r\}$. Then since $m,n \geq 1$, we have $pm + qn + r \leq (p + r)m + (q + r)n \leq s(m + n)$. Hence $\Psi$ is recognizable within time $s(m + n)$. Let k be a positive integer such that $\frac{s}{k} < \varepsilon$. Then by the Speed-Up Theorem $\Psi$ is recognizable within time $\frac{s}{k}\cdot(m + n) + m + n + 2 \leq (1 + \varepsilon)(m + n) + 2$. □

Corresponding to Theorem 2.4 we have the following.

Theorem 2.12

The class of linear predicates is a Boolean algebra which contains all finite predicates.

PROOF: The containment of all finite predicates was shown in the proof of Theorem 2.4. Since we are interested in the values of the recognition-time functions only over the positive quadrant in (m,n)-

space, we can bound the maximum of two linear functions by a linear function. The result then follows from Corollary 2.4.1. □

Corresponding to Theorem 2.5 we have the following.

Theorem 2.13

Let $\Psi$ be a predicate and M a cell type which accepts (or rejects) $\Psi$ in linear time. Then there exists a cell type M' which recognizes $\Psi$ in linear time.

PROOF: Assume M accepts $\Psi$, the case of M rejecting $\Psi$ is similar. Since M accepts $\Psi$ in linear time, we have by similarity to Theorem 2.11 that $\Psi$ must be acceptable in time $(1 + \varepsilon)(m + n) + 2$ for any $\varepsilon > 0$. Let M" be a cell type which accepts $\Psi$ in time $2(m + n) + 2$. Modify M" by adding a second layer which counts up to $2(m + n) + 3$ and then rejects the figure if it hasn't already been accepted by M". The cell type so constructed is M'. □

Note that the correspondence of Theorem 2.13 to Theorem 2.5 is not quite complete. Namely we do not claim that M' is effectively constructable given M. It is certainly effectively constructable given M and an integer k such that M always accepts $\Psi$ by time $t = k(m + n)$. However, the problem of whether or not it is possible to effectively compute such a k, given just M and the knowledge that M accepts in linear time is open. A partial result is given by the following theorem due to Mike Paterson (unpublished).

Theorem 2.14 (Paterson)

Given a predicate $\Psi$, a cell type M which recognizes $\Psi$, and given that there is an integer k such that M recognizes $\Psi$ in time $k(m + n)$, then the problem of finding a minimum such integer k is in general unsolvable.

PROOF: Let $\Psi$ be the empty predicate. Given any Turing machine T, construct a cell type $M_T$ which behaves as follows. It simulates T for n steps on a blank tape. If T has not halted, it rejects the input figure at time $2(m + n)$. If T has halted, it rejects the input figure at time $3(m + n)$. Thus $M_T$ recognizes $\Psi$ in time $2(m + n)$ $\iff$ T never halts, and in time $3(m + n)$ $\iff$ T eventually halts. The result follows. □

Similarly we have the following.

Theorem 2.15

Given a predicate $\Psi$ and a cell type M which recognizes $\Psi$, it is in general undecidable whether or not M recognizes $\Psi$ in linear time.

PROOF: As in the proof of Theorem 2.14, let $\Psi$ be the empty predicate and for any Turing machine let $M_T$ be a cell type such that

$M_T$ recognizes $\Psi$ in time $2(m + n)$ $\iff$ T doesn't halt

and $M_T$ recognizes $\Psi$ in time $2(m\ n)$ $\iff$ T halts. The result follows. □

We will see in the next chapter that some predicates which would intuitively seem to require time on the order of $m \cdot n$ are in fact linear. Indeed, the existence of a cellular predicate which is not linear is an open question. There are many candidates for non-linearity, but there are no known methods of proving them to be non-linear. The only method of establishing minimal time bounds which is available is the Interdependence Theorem, which is only useful in establishing bounds less than $m + n - 1$. Cole[6] in his thesis on iterative computers has established certain computations which cannot be done in real time under restrictions on intercell communications. However his model receives information from the external world as the computation progresses and can be overloaded as he shows. In our model the input has already been digested at time $t = 0$, so no confusion of inputs is possible. Modified diagonalization arguments have been tried by the author and by several other people; but promising as they seem, every such argument has contained a flaw. Finally we might add that this open question is related via Theorem 2.8 to an apparently open question about deterministic linear bounded automata. Assuming $m = 1$, Theorem 2.8 implies that if all cellular predicates were linear, then all predicates (i.e. languages) recognizable by deterministic LBA (using our particular model in which the input tape also serves as the working tape) could be recognized in time proportional to the square of the length of the tape. Hence the existence of a language requiring on the order of $n^3$ units of time for its recognition would show that non-linear cellular predicates exist. As far as the author knows, no such language has been found.

# CHAPTER 3

## RECOGNITION OF TOPOLOGICAL INVARIANTS

In this chapter we will study the recognition of some specific predicates over black and white figures. All of these predicates depend only on topological properties of the figure such as connectivity, simple connectivity, number of components, Euler number, and so on. The principal result of this chapter is a fundamental transformation of figures which allows the construction of algorithms for recognizing in linear time a wide variety of these predicates. Many of these predicates have been studied in Minsky and Papert[10] and in Blum and Hewitt[5]. The interested reader may thus compare arrays with perceptrons and pebble automata with regard to recognizing these predicates.

### 3.1 Basic Terminology

Assumption: Unless otherwise specified, all figures and predicates in this chapter are assumed to be over $I_2$ (i.e. black and white).

We begin by establishing some terminology.

**Definition**

Two cells at $(i,j)$ and $(p,q)$ are said to be adjacent if $|i - p| + |j - q| \leq 1$ and are said to be neighboring if $|i - p| \leq 1$ and $|j - q| \leq 1$.

Two black cells are connected if there is a chain of pairwise adjacent black cells beginning with one and ending with the other. Two white cells are connected if there is a chain of pairwise neighboring white cells beginning with one and ending with the other.

Note the asymmetric definition of connectedness for black and for white cells. Some such asymmetric definition is necessary if one is to retain such "nice" properties as the Jordan Curve Theorem. A notion of connectedness which is symmetric with respect to black and white can be obtained by assuming that each cell "touches" all of the neighboring cells except the ones to the northeast and southwest. This notion which is derived from a hexagonal partition is, however, asymmetric with regard to direction.

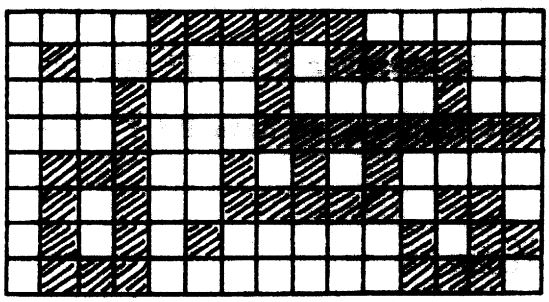
We assume that figures are presented against a white background. Hence the following definition.

Definition

The equivalence classes of black cells under the relation "connected" are called the components of P. The equivalence classes of white cells under the relation "connected" which do not contain cells on the border (that is cells in rows 1 or m or in columns 1 or n) are called holes. The remaining equivalence classes of white cells are lumped together into a class of white cells called the

background. A component or hole which contains only one cell is said to be isolated, otherwise non-isolated.

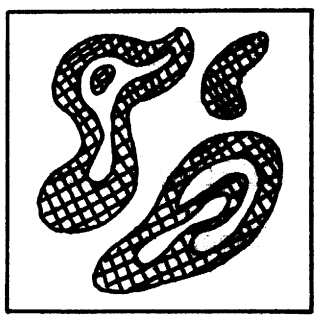
For example, the following figure has five components (two of which are isolated) and three holes (one of which is isolated).

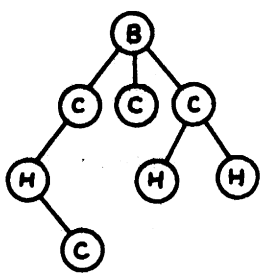


Another example is an 8×8 checkerboard which according to our definitions has thirty-two components, all of which are isolated, and no holes.

Definition

Given a figure, one can construct an associated tree which represents the containment relationships between the background, the components, and the holes. A figure and its associated tree are shown below.





Two figures which have isomorphic trees are said to be topologically equivalent, where by isomorphic trees we mean isomorphic as labeled graphs or as unoriented rooted trees.

Definition

A predicate $\Psi$ is said to be topologically invariant if whenever P and P' are topologically equivalent figures, we have $P \in \Psi \iff P' \in \Psi$.

## 3.2 An Example: $\Psi_{CONN}$

The first topologically invariant predicate we study consists of the set of all connected figures.

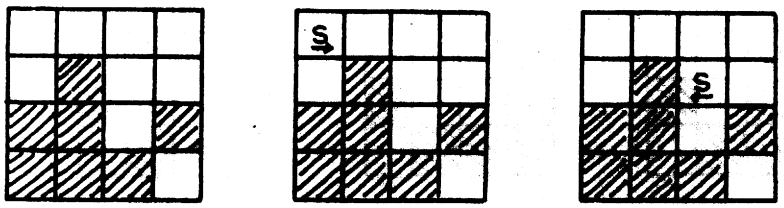
Definition

The predicate $\Psi_{CONN}$ is given by

$P \in \Psi_{CONN} \iff$ P contains at most one component.

If we were to ask the reader at this point to design a cell type which recognizes $\Psi_{CONN}$, his first attempt might very well be an erase-one-component-and-see-if-anything-is-left algorithm. We now describe such an algorithm.

At time t = 0 the northwest corner cell emits a scanning signal s which begins to scan the array row by row in a back and forth manner until it encounters a black cell. This stage of the process is illustrated in the diagrams on the following page.

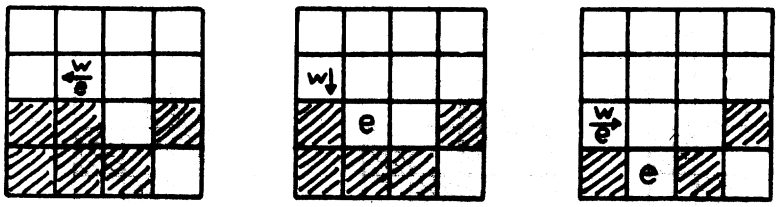



t = 0 t = 1 t = 6

At the point at which s encounters the first black cell, two things happen. First of all a chain reaction of erasure is set off within the component to which the black cell belongs. The black cell which was struck by s turns white and emits an erase signal e to each of its four neighbors. The e signals are ignored by white cells, but an e signal striking a black cell causes it to turn white and emit e signals to its four neighbors. In this manner the entire component is erased.

The second thing which happens when s encounters the first black cell is that s changes into a waiting signal w. The waiting signal continues the same zigzag scanning motion which s had been using, but does not interact with either black or white cells or with e signals which are propagating around the array in various directions. The w signal eventually completes the scan of the array and strikes one of the bottom corners of the array. At this point the erasure of the component is guaranteed to be complete.

The first three steps of the erasure process are shown in the figure on the following page.



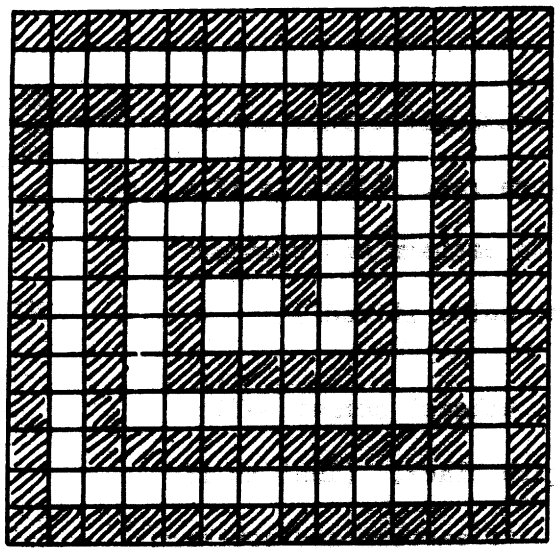

t = 7 t = 8 t = 9

When w strikes the corner at the end of its scan, the figure contains one less component than it did to begin with. All that remains to be done is to see if the remaining figure is blank. This is accomplished by having the w signal rebound from the last corner as an accept signal a which scans up the array searching for a black cell.

If a encounters a black cell, it is converted into a reject signal r which heads directly for the northwest corner to cause a reject. If a doesn't encounter a black cell, it eventually strikes the northwest corner causing an accept.

The case of the blank figure is handled by having the s signal rebound as a when it completes its scan.

The cell type implicitly described above recognizes connectivity in time approximately 2mn and hence is not linear. The reader is challenged to find a faster method of recognizing connectivity before reading on. A good (or bad, depending on your point of view) example to keep in mind while searching for a linear method is the figure illustrated on the following page which has length and area of about $\frac{1}{2}mn$.



## 3.3 A Fundamental Transformation

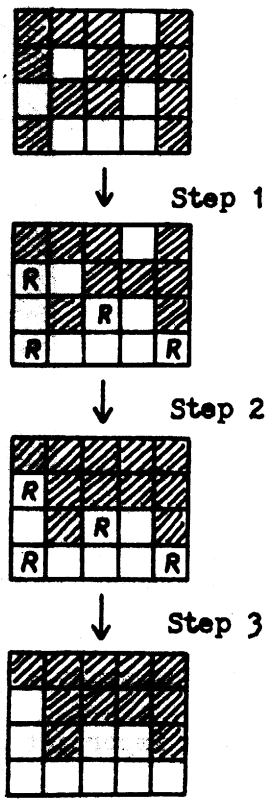
We now study a transformation of black and white figures which will have important applications to the recognition of topological invariants by iterative arrays. By transformation we mean a mapping from black and white figures into black and white figures. The transformation will be studied in its own right in this section and its applications will be discussed in the following section. This transformation was discovered by the author while he was attempting to prove that connectivity could not be recognized in linear time. Since its first application was to show that connectivity could be recognized in linear time, we call it the connectivity transformation and denote it by T. The image of a figure P under T is denoted by T(P).

For heuristic purposes we will describe the transformation as taking place in three steps.

Step 1. Color all southeast corner cells of the black subfigure red. (That is if a cell is black and its eastern and southern neighbors are white, color it red.)

Step 2. Color all southeast corner cells of the white subfigure black. (That is if a cell is white and its eastern and southern neighbors are black and its southeastern neighbor is either red or black, color it black.)

Step 3. Color all red cells white.




Properties of T: We now informally describe the properties of T. The remainder of this section will be devoted to proving these properties in a series of lemmas. If one considers repeated applications of T to a figure, one observes that each component is reduced to an isolated component which then disappears. Distinct components remain distinct and either vanish at different points or at the same point at different times. Similarly, each hole is reduced to an isolated hole which then vanishes with distinct holes remaining distinct and vanishing at different points or different times. It is easy to calculate exactly how many applications of T will be

required to reduce a given component or hole to a single cell and exactly where that cell will be. The entire figure, no matter how complex, will be reduced to the all white background in less than m + n applications of T.

To begin proving the above statements, we need some way of relating the components of P to those of T(P). This is done in the next four lemmas by using the concept of a stationary point.

Definition

A cell (i,j) is called a stationary point of P if it is black in both P and T(P).

Note that the stationary points are exactly those black cells in P which are not southeast corners of the black subfigure of P.

Lemma 1

Every non-isolated component of P contains a stationary point.

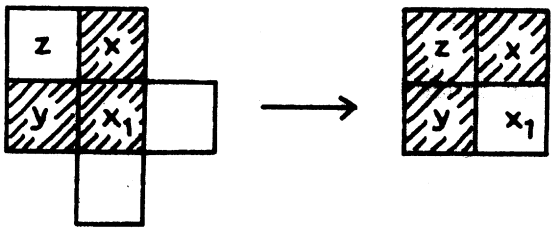
PROOF: Let C be a non-isolated component and let x be a northwest corner of C. Then x must be a stationary point for otherwise it would also be a southeast corner and hence $C = \{x\}$ would be isolated. □

Lemma 2

Two stationary points are connected in P if and only if they are connected in T(P).

PROOF: [$\Longrightarrow$] Let x and y be two stationary points of P which are connected. Then by definition there exists a sequence $x_0, x_1, \ldots, x_n$

of distinct pairwise adjacent black cells such that $x = x_0$ and $y = x_n$. We use induction on n. If n = 1, then x is adjacent to y and we are done. If n = 2, then either $x_1$ is also a stationary point in which case we are done, or $x_1$ is a southeast corner. In the latter case we have the situation depicted in the figure below, possibly with x and y interchanged and one sees that the cell z will be black in T(P) no matter what its color in P. Thus x and y are connected in T(P).

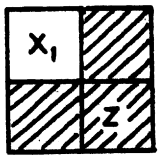



P T(P)

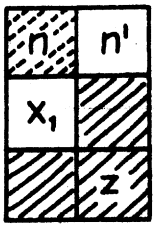
Now assume $n \geq 3$. Observe that any chain of distinct pairwise adjacent black cells cannot contain two consecutive southeast corners. Thus either $x_{n-2}$ or $x_{n-1}$ is a stationary point and we may apply the induction hypothesis to the chain $x_0, \ldots, x_k$ and $x_k, \ldots, x_n$ where k is either n - 1 or n - 2. Thus x is connected to y in T(P) via $x_k$.

$[\Longleftarrow]$ Suppose x and y are connected in T(P) and let $x_0, x_1, \ldots, x_n$ be a sequence of pairwise adjacent black cells in T(P) such that $x = x_0$ and $y = x_n$. Again we use induction and again the cases for n = 1 and n = 2 with $x_1$ a stationary point (of P) are trivial, so assume n = 2 and $x_1$ is not a stationary point. Then $x_1$ must have been white in P since it is black in T(P). Hence $x_1$ must

have satisfied the conditions in step 2 of the description of T and the situation depicted below must have existed in P.




We know that x and y are adjacent to $x_1$ and are stationary points of P. Therefore it will suffice to show that all stationary points of P which are adjacent to $x_1$ are connected in P to the cell labeled z. From the diagram above we see that the eastern and southern neighbors of $x_1$ are indeed stationary points of P and are connected to z. Now consider the northern neighbor of $x_1$ which has been labeled n in the diagram below.




Assume n is a stationary point. Then n cannot be a southeast corner and hence either $x_1$ or n' is black. But $x_1$ is white, thus n' must be black. Therefore if n is a stationary point, it is connected to z via n'. A similar argument holds for the western neighbor of $x_1$. This completes the case for n = 2.

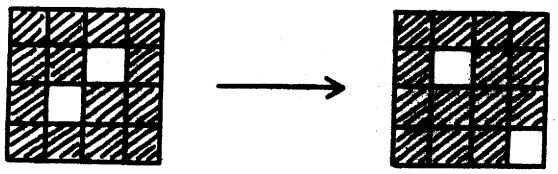
The remaining cases for $n \geq 3$ follow as before from the observation that either $x_{n-1}$ or $x_{n-2}$ is a stationary point, although different reasoning must be used to make this observation now since $x_0, \ldots, x_n$ is a chain in T(P). □

Combining lemmas 1 and 2 with the observation that every black cell in T(P) is either a stationary point of P or is adjacent to a stationary point, we have shown:

Lemma 3

There is a canonical one-to-one correspondence between the non-isolated components of P and the components of T(P).

We now state without proof the corresponding lemma for holes, which can be proved by methods similar to those above. However, a slightly different concept than that of stationary point must be used since some holes such as the one illustrated below have no stationary points.



P T(P)

Lemma 4

There is a canonical one-to-one correspondence between the non-isolated holes of P and the holes of T(P).

The following lemma should be obvious by now.

Lemma 5

If P contains no isolated components or holes,

then P and T(P) are topologically equivalent.

We now show how to compute the number of applications of T required to reduce a component to a single square and where that square will lie. Identical results can be proved for holes using similar arguments.

Definition

Given a component C of a pattern P, let

$T^0(C) = C$

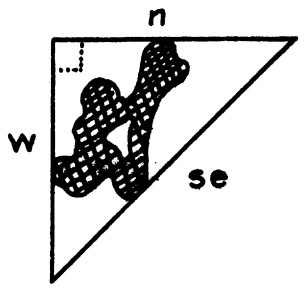
$T^k(C)$ = the canonical image of $T^{k-1}(C)$ under T for k 0 (provided it exists)

$n(C) = \min \{ i \mid \text{row } i \text{ intersects } C \}$

$w(C) = \min \{ j \mid \text{column } j \text{ intersects } C \}$

$se(C) = \max \{ i + j \mid (i,j) \in C \}$

Note that n(C), w(C), and se(C) represent three lines forming a triangle such that C lies within the triangle and touches each line as shown in the figure below.




We will show that the component vanishes at the cell indicated by

the dotted lines and that the number of applications of T required to achieve this is equal to the distance from this cell to the se line.

Lemma 6

If C is a non-isolated component, then

$n(T(C)) = n(C)$

$w(T(C)) = w(C)$

$se(T(C)) = se(C) - 1$

PROOF: $[n(T(C)) = n(C)]$ It is clear that $n(T(C)) \geqslant n(C)$, since each black cell in T(C) is either a stationary point (of P) or is the western neighbor of a stationary point. On the other hand, if (i,j) is the western most point of C which lies in row n(C), then (i,j) must be a stationary point and hence $n(T(C)) \leqslant n(C)$.

$[w(T(C)) = w(C)]$ This result follows immediately from the above and the symmetry of T with respect to north and west.

$[se(T(C)) = se(C) - 1]$ Any cell (i,j) in C such that $i + j = se(C)$ must be a southeast corner of C and hence is adjacent to a stationary point (p,q) such that $p + q = se(C) - 1$. Thus $se(T(C)) \geqslant se(C) - 1$. On the other hand all such cells (i,j) do not appear in T(C), so $se(T(C)) \leqslant se(C) - 1$. □

Lemma 7

If C is a non-isolated component, then

$T^{k(C)}(C)$ is an isolated component located at

$(n(C),w(C))$, where $k(C) = se(C) - n(C) - w(C)$.

PROOF: By lemma 6 we have $k(T(C)) = k(C) - 1$. Thus by induction

$k(T^{k(C)}(C))) = k(C) - k(C) = 0$, which can only hold for an isolated component. That component must be located at $(n(T^{k(C)}(C)), w(T^{k(C)}(C)))$ which is $(n(C), w(C))$ by Lemma 6. □

Lemma 8

If P is an m×n figure, then $T^{n+m-1}(P)$ is the all white figure.

PROOF: Apply Lemma 7 and the fact that $k(C) < n + n - 2$ for any component C of P. □

Lemma 9

If P is an m×n figure containing c components and h holes, then the total number of isolated components appearing in the figures $P, T(P), T^2(P), \ldots, T^{m+n-1}(P)$ is c and the total number of isolated holes is h.

PROOF: Immediate from Lemmas 7 and 8. □

## 3.4 Linear Recognition of Topological Invariants

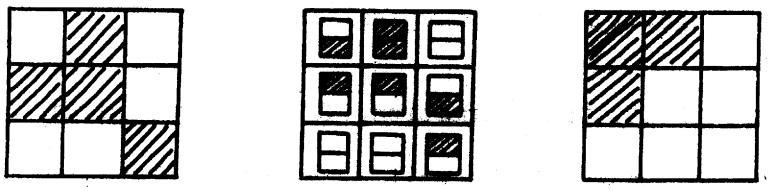
The connectivity transformation, T, described in the previous section forms the basis of the cell types to be presented in this section. These cell types all have two common characteristics:

i) They recognize topologically invariant predicates in linear time.

ii) They consist of two layers, a lower or transformation

layer which carries out successive connectivity transformations on the initial figure, and an upper or observation layer which watches the transformations taking place, gathers and processes information, and finally comes to a decision about the figure.

The transformation layer can carry out successive applications of the connectivity transformation, T, at the rate of one transformation every two units of time as follows. At time $t \equiv 0 \pmod 2$ the latest figure P is represented in the transformation layer. At time $t \equiv 1 \pmod 2$ each cell has entered a state which represents not only its own state at the previous time, but also that of its southern neighbor. Each cell now has access to the information necessary to enter the appropriate state in T(P). By the next unit of time, $t \equiv 0 \pmod 2$, the transformation is complete. The intermediate step is necessary to pass information around the corner so that a cell in the white state can determine the state of its southeastern neighbor. (This step could be eliminated if diagonal connections were allowed.) The process is illustrated below.



$t \equiv 0$ $t \equiv 1$ $t \equiv 0$

The observation layer watches for the disappearance of components or holes in the transformation layer and generates appropriate signals at each such disappearance. These signals are then processed and a

decision is reached. In some cases it is necessary for the northwest corner cell to know that it has received all the information required for a decision. In these cases the southeast corner cell sends out a timing signal which propagates through the array at an appropriate rate. When the timing signal reaches the northwest corner cell, all other signals must have preceded it, and a decision can be made.

We now present some topologically invariant predicates which are linear. In each case the proof that the predicate is invariant rests on the construction of a cell type which recognizes the predicate in linear time. As explained above, all of these cell types operate in two layers with the lower layer being the transformation layer. Thus to describe any given cell type, we need only describe the observation layer.

Theorem 3.1

$\Psi_{CONN}$ is a linear predicate.

PROOF: Signals are only generated by vanishing components. As each signal is generated it heads for the northwest corner cell. The figure is rejected if more than one such signal is received at the corner. If two such signals collide on the way to the corner, they combine to form a reject signal which, when it reaches the corner, will cause the figure to be rejected.

Definition

Let $\Psi_{SC}$ be given as follows

$P \in \Psi_{SC} \iff$ all components of P are simply connected.

**Theorem 3.2**

$\psi_{SC}$ is a linear predicate.

PROOF: Signals are generated by vanishing holes. If the northwest corner receives any such signal, the figure is rejected; otherwise it is accepted. □

The above two predicates are special cases of a more general predicate.

**Definition**

For any $c_1, c_2, h_1, h_2$ such that $0 \leq c_1, c_2, h_1, h_2 \leq \infty$ let $\psi^{c_1, c_2}_{h_1, h_2}$ be given by

$$P \in \psi^{c_1, c_2}_{h_1, h_2} \iff \begin{cases} c_1 \leq c \leq c_2 \text{ and } h_1 \leq h \leq h_2 \\ \text{where c is the number of components} \\ \text{in P and h is the number of holes.} \end{cases}$$

**Theorem 3.3**

For any $0 \leq c_1, c_2, h_1, h_2 \leq \infty$, the predicate $\psi^{c_1, c_2}_{h_1, h_2}$ is linear.

PROOF: This is a simple adaptation of the methods employed in Theorems 3.1 and 3.2. □

In the above theorems we have merely used our transformation to count components and holes. But Lemma 5 indicates that the connectivity transformation preserves additional topological information. By introducing a slightly different mode of operation in the observation layer, we may take advantage of this fact. Consider

the following predicate.

Definition

Let $\psi_{NDC}$ be the predicate given by

$$P \in \psi_{NDC} \iff \begin{cases} \text{No component of P is doubly} \\ \text{connected (i.e. no component} \\ \text{of P has exactly two holes).} \end{cases}$$

Now $\psi_{NDC}$ is not of the form $\psi^{c_1 c_2}_{h_1 h_2}$, but nevertheless we have

Theorem 3.4

$\psi_{NDC}$ is a linear predicate.

PROOF: As before signals are generated by vanishing holes, but unlike previous cases they do not immediately head for the northwest corner. Instead they remain positioned over the component in which they were embedded. As these components gradually shrink and shift under the action of the connectivity transformation, the hole signals shift so as to remain positioned over the components in which they originated. By the time the component is finally reduced to a single cell, all hole signals associated with it have collided and a signal generated by their combination is centered over the now isolated component. In the case of $\psi_{NDC}$ this combined signal would indicate whether the component had originally contained 0, 1, 2, or more than 2 holes. In the case of 2 holes, a reject signal would be generated which would cause the figure to be rejected. Otherwise both the component and the combined hole signal vanish at the next unit of time. □

The method of making a signal generated by a vanishing hole remain above the component in which the hole was embedded can be applied in a dual manner. Thus the predicate "no hole contains exactly two components" could be recognized since it is just the dual of $\psi_{NDC}$. This procedure may be extended as in the following theorem.

Theorem 3.5

Given any figure Q, let $\psi_{\equiv Q}$ be the predicate given by

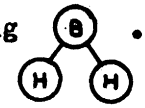
$$P \in \psi_{\equiv Q} \iff \begin{cases} P \text{ is topologically} \\ \text{equivalent to } Q. \end{cases}$$

Then $\psi_{\equiv Q}$ is linear.

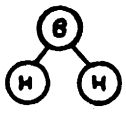
PROOF: Recall that two figures were said to be topologically equivalent if they had isomorphic associated trees. Thus given an input figure P, we can begin computing its associated tree. If at any point in the computation it becomes apparent that the tree of P is not isomorphic to the tree of Q, a reject signal can be generated. Otherwise the computation will continue to completion and P will be accepted. The computation of the tree of P can be carried out in the following manner.

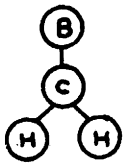
When a hole vanishes (assuming there were no components located in that hole), it generates a signal which represents the subtree

.

This signal floats above the component in which the hole was located. If two such signals should collide, they combine to form a signal representing 

. (Extensions to more holes is clear.) When

eventually the component vanishes, it replaces the signal 

by a signal 

. This signal then floats above the hole (or possibly the background) in which the component was located. The process now continues. In general the signal generated by a vanishing subfigure will represent a tree which is obtained from that subfigure's associated tree by adjoining a node marked "B" to the root. When two such signals collide, they join to form a signal representing the tree obtained by identifying the "B" nodes. By the time P has been reduced to the background and all signals have merged at cell (1,1), the resulting signal will represent the tree associated with P.

The process described above is workable but for one fact. Since there is infinite variety in associated trees and subtrees, we need an infinite number of signals to represent them. In the case of recognizing $\Psi_{\equiv Q}$ this restriction is not effective since we need only enough signals to represent all the possible subtrees which could be formed while constructing the tree associated with Q. Anytime two signals collide or a component vanishes in such a manner that the resulting signal has not been provided for, it indicates that P is not equivalent to Q and a simple reject signal may be produced instead. □

Corollary 3.5.1

Given any figure Q, let $\Psi_{\subset Q}$ be the predicate given by

$$P \in \Psi_{\subset Q} \iff \begin{cases} \text{the associated tree of } P \\ \text{is isomorphic to a subtree} \\ \text{of } Q. \end{cases}$$

Then $\Psi_{\subset Q}$ is linear.

PROOF: A cell type which recognizes $\Psi_{\subset Q}$ may be constructed by a simple adaptation of the method of Theorem 3.5. □

Corollary 3.5.2

Given any finite set $\mathcal{S}$ of figures, let $\Psi_{\mathcal{S}}$ be the predicate given by

$$P \in \Psi_{\mathcal{S}} \iff \begin{cases} \text{there exists an } S \in \mathcal{S} \text{ such that } P \\ \text{is topologically equivalent to } S. \end{cases}$$

Then $\Psi_{\mathcal{S}}$ is linear.

PROOF: The result follows immediately from Theorem 2.12, since $\Psi_{\mathcal{S}} = \bigcup_{S \in \mathcal{S}} \Psi_{\equiv S}$ .

There seems to be an endless variety of topologically invariant predicates to which the connectivity transformation may be applied to obtain linear results. We will conclude with the mention of two predicates which seem interesting and amusing.

Definition

Let $I_4 = \{b,w,s,f\}$ (representing black, white, start, finish). A figure P over $I_4$ is said to be a <u>maze</u> provided it contains exactly one occurence of s (start) and exactly one occurence of f (finish).

A maze P is said to be <u>solvable</u> if the cells containing the s and f are connected by a chain of pairwise connected black cells.

<u>Theorem 3.4</u>

Let $\psi_{MAZE}$ be the predicate over $I_4$ given by

$P \in \psi_{MAZE} \iff$ P is a solvable maze.

Then $\psi_{MAZE}$ is linear.

PROOF: Given a figure P over $I_4$, it can be determined in a linear amount of time whether or not it is a maze. If it is, a two layer connectivity type array can determine solvability in linear time by having the s and f float above the components in which they were embedded. They will collide if and only if the maze is solvable. □

In view of Theorem 2.11, the above result says that we can test whether two given points in a figure are connected in time $(1 + \varepsilon)(m + n) + 2$. But in many figures the shortest path between such points is on the order of $\frac{1}{2}mn$ in length. Thus we can determine that two points are connected in an amount of time which is less than that required to transmit a signal along the shortest path in the component connecting them (!).

<u>An application</u>: The final predicate we consider is included as a highly impractical application of the foregoing methods. Let any figure over $I_4$ represent a map of some islands in a lake with white representing water and all other symbols representing land. Let b represent a bare plot of ground, s a plot on which a sheep is standing,

and f a plot on which a sheep dog (Fido) is standing. Then it will be of relief to shepherds who keep their flocks on islands to know that the following predicate is linear.

$$P \in \Psi_{SHEEPDOG} \iff \begin{cases} \text{every island which has sheep has} \\ \text{a sheep dog.} \end{cases}$$

The proof is omitted.

Higher Dimensions: One might ask for a three-dimensional analog to the connectivity transformation which would allow a three-dimensional iterative cube of automata to test in linear time whether an input figure (input solid?) is connected. No such transformation has been found. Indeed, one can see that any transformation which would handle three-dimensional cases must be somewhat more complicated than a simple shrink-the-components-to-a-single-point approach, since one can have figures such as two interlocking rings which must be unlocked before being shrunk.

As a problem intermediate between two and three dimensions, one might study the problem of recognizing connectivity in multi-level mazes of finite depth. The solution in linear time of multi-level mazes would allow us to relate arrays and two-dimensional finite state automata in an interesting manner.

Theorem 3.5

> If multi-level mazes are solvable in linear time by iterative arrays, then any predicate recognizable by a two-dimensional finite state automaton is recognizable by an array in linear time.

PROOF: Assume that multi-level mazes are solvable in linear time. Let A be a fixed two-dimensional finite state automaton which operates on figures over the set I. Then construct the cell type $M_A$ which operates as follows. At time t = 0 the figure over I is present. At time t = 1 the array has created an s-level maze, where s is the number of states in A. Each level represents a state of A. If cells (i,j) and (p,q) are adjacent and if the initial input at (i,j) would cause the automaton to go from state k to state r and move from (i,j) to (p,q), then a connection between level k at (i,j) and level r at (p,q) is established in the maze. Thus we see that by time t = 1 the array has constructed a map of the movements of A over the figure. This map contains the path actually traced out by A when it is placed on cell (1,1) in its starting state as well as many other paths corresponding to state/location configurations which A would never actually enter. The question "does A accept the figure?" now becomes the question "does the multi-level maze have a solution?" (using the initial state level at (1,1) as the start point and any accept state as a finish point). By assumption this problem is a linear problem.

The author has tried several modifications of the connectivity transformation in an attempt to find a successful method of solving multi-level mazes in linear time, but none have worked. Multi-level mazes can certainly be solved in time proportional to m·n, but their linearity is still an open question. □

## 3.5 Euler Number

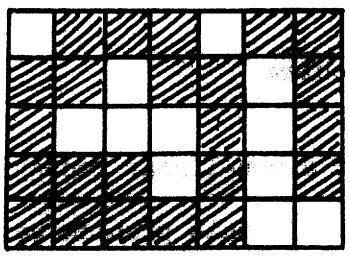
**Definition**

The **Euler number** of a black and white figure is the number of components in the figure minus the number of holes.

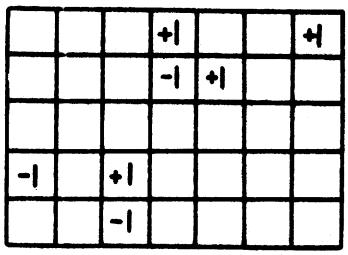
Minsky and Papert[10] show that predicates depending in a reasonable way on the Euler number of a figure are easily recognized by perceptrons. Their method of determining the Euler number is to perform a weighted sum over certain subfigures. Namely, each occurrence of the subfigures and counts +1 and each occurrence of the subfigures and counts -1. They showed by induction how the sum of these integers over a figure is equal to the Euler number of the figure. Their somewhat obscure motivation for this process is that is analogous to a vertex, to an edge, and to a face. Actually there is a quite simple interpretation of their result. If we agree to deposit the count (±1) for each of these subfigures in its southeast corner square and if we agree to sum the counts deposited in any given square, we find that only two cases arise where a non-zero count is attributed to a square. These correspond to the subfigures L and R below.

Now think of an observer who starts walking along the outer boundary of a component, keeping the component on his left, and who eventually returns to his point of departure. During his stroll he will have made a number of left- and right-hand turns. Since he was on an outer boundary, he must have made a total of four more left than right turns in returning to his starting point. If instead of counting all turns, one counts only those left turns which change direction from north to west and only those right turns which change direction from west to north, then one will have exactly one more left than right turn. But these special north-to-west and west-to-north turns occur only at L and R configurations respectively. Thus an outer boundary has exactly one more L than R configuration. Similarly an inner boundary has exactly one more R than L configuration. Hence the sum over a figure of the weighted L and R configurations is equal to the difference between the numbers of outer and inner boundaries. But this is just the difference between the number of components and holes, which in turn is the Euler number of the figure.

As an example, the figure



would produce the following distribution of counts.




We can easily design an array which would by time t = 2 have transformed an original figure into its corresponding pattern of +1's and -1's. This might be a natural first step in setting out to recognize a predicate which depends on the Euler number. Consider for example the predicate, $\psi_{EULER}$, which contains all those figures having positive Euler numbers. After transforming the figure into a distribution of counts, the problem becomes simply to determine whether there are more +1's than -1's.

This can easily be done in time proportional to m·n. It certainly seems that one should be able to detect a surplus of one type of symbol over another in linear time, but surprisingly, this question remains open.

## 3.6 Topological Match Problem

A prime candidate for a non-linear recognition problem is the topological match problem. In this problem we present two black and white figures to the array and ask whether or not the two figures are of the same topological type. The best known times for solving this problem are on the order of $(m \cdot n)^2$. We will mention several methods

of solving this problem below.

Representing pairs of figures: Let us make the following convention for representing pairs of black and white figures. Given two black and white figures we wish to compare, we first adjust them to have the same number of rows by adding extra rows of white cells to the bottom of whichever figure has fewer rows. The figures which now have the same number of rows are placed side by side, separated only by a single column of red cells. The total figure consisting of the two black and white figures and the red divider constitutes the input to our array. We will refer to the two figures as the left figure and the right figure. An array which is to solve the topological match problem will accept only those figures which are in this form and in which the left figure is topologically equivalent to the right.

Converting a figure into tree representation: It is possible to design arrays which solve the topological match problem by dealing directly with the figures as initially presented to the array. It is also possible to first convert the figures into representations of their associated trees and then perform isomorphism checking on their trees. Since topological equivalence of figures was defined in terms of isomorphism of the associated trees, we feel the tree comparison method is conceptually cleaner as well as much easier to describe. The process of converting a figure into a representation of its associated tree appears to require on the order of $(m \cdot n)$ units of time to complete. Since all known methods of comparing figures or trees for isomorphism take on the order of $(m \cdot n)^2$ units of time, the

conversion process does not alter the functional form of our recognition time. By using speed-up, we may compensate entirely for the time spent in conversion.

The conversion process may be thought of as taking place in the following two steps:

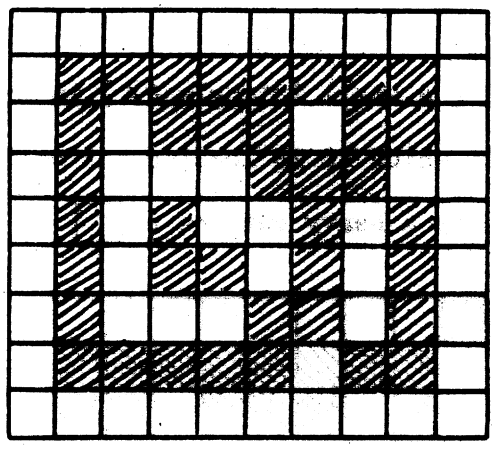
i) one cell in each component or hole is chosen to represent the corresponding node in the associated tree and

ii) the nodes are connected together by a pathway of cells in a manner which represents their connectedness in the associated tree.

We now describe a process for converting a figure into a tree in which these two steps overlap. Since the distinction between background, component, and hole will be relatively unimportant in the following discussion, we will use the term area to refer to any of them. The background is by definition one area, but it may be in fact disconnected by a component which touches the edges of the array in several places. A disconnected background represents a difficulty for the process about to be described, since it recognizes the integrity of an area by its connectedness. We will assume that the figure to be processed has an all white border and hence a connected background. Any figure which does not have such a border may be treated as if it did by having the border cells of the array pretend to be double cells with the outer cell being white.

As an example we will follow the figure given in the following diagram through its conversion into a tree representation.

t = 0



Our process involves a row by row scan of the entire figure by a signal which we will call the scanner. The scanner will travel from west to east along the top row, then move south one row and travel from east to west along that row, move south a row and so on. Each time it strikes an area for the first time, the scanner pauses while that area is processed. The processing of a new area has three points of interest:

i) The cell on which the scanner is sitting becomes marked as the node corresponding to that area;

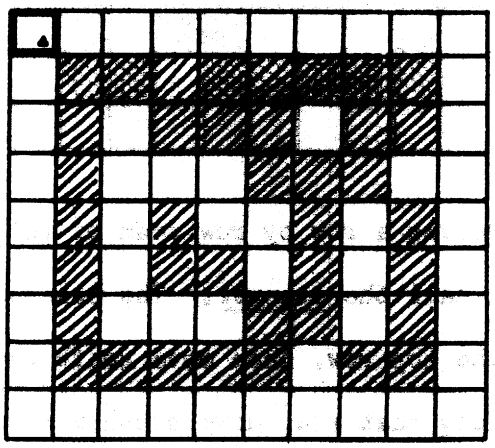
ii) a linking pathway is established between this newly created node and the appropriate node above it in the tree representation; and

iii) the entire area is marked by a contagious process so that it will be recognized by the scanner as having been processed.

Shortly after the scanner has completed its scan, the transformation will be complete.

At t = 1 the scanner is sitting on cell (1,1) and begins waiting for the background to be processed. We will represent the location of the scanner by drawing a small triangle ( ◂ ) in the lower right-hand corner of the cell in which it is located. A cell which has become a node will be denoted by a heavy outline ( ▣ ). Thus we have

t = 1

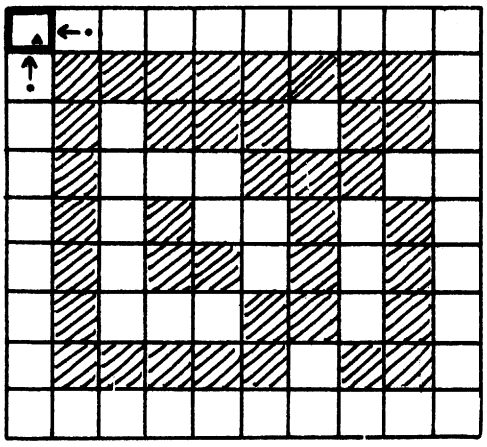


In the case of the background, no connection to another node is made. The contagion which marks the background as having been processed is similar to the contagious erasure used to recognize $\Psi_{CONN}$ in Section 3.2, but with two important differences. First, each cell which catches the contagion keeps a permanent record of one of the directions from which the contagion arrived. We will represent this record by a small arrow within each cell which begins in the center of the cell and points in the direction from which the contagion arrived ( ▣ ). Second a mechanism is provided which allows the scanner to know when the contagion which it has created within an area has completely covered that area. As each new cell catches the contagion, it sends a signal, called a dot, back along the system of arrows which ultimately leads to the scanner. Thus as long as the

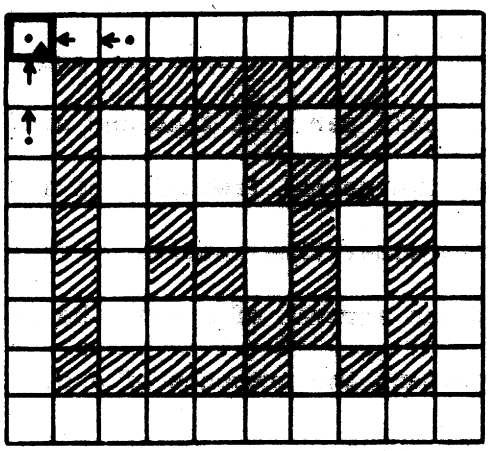
contagion continues to spread, the scanner will continue to receive dots and will not resume its scan. Since the contagion moves away from the scanner at one cell per unit time and since dots travel at one cell per unit time, the normal frequency of arrival of dots at the scanner is one dot every other time interval. Thus if two time intervals pass without the arrival of a dot, the scanner may resume its scan. We will represent the presence of a dot in a cell by a black dot ( ). One might think of the contagion as a spreading grass fire and the dots as particles of smoke drifting back from the fire. The signalling mechanism provided by the dots is necessary since the scanner recognizes new areas by the fact that their cells have not been infected by contagion. If the scanner resumed scanning before the contagion stopped, then it would not be able to distinguish between cells in newly encountered areas and cells in old areas to which the contagion had not yet spread.

Let us now look in on the progress of our example. (Note: the diagrams which follow are intended as an aid to understanding the process and do not necessarily represent all the information present in each cell.)

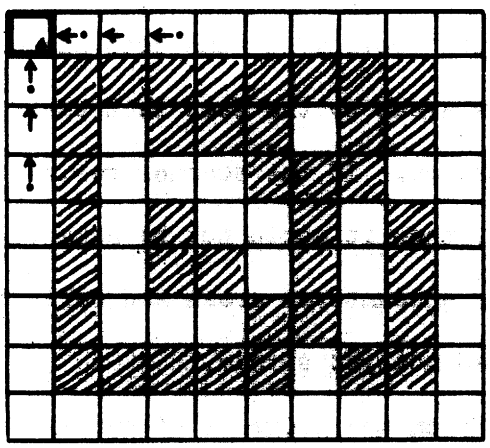
t = 2



t = 3

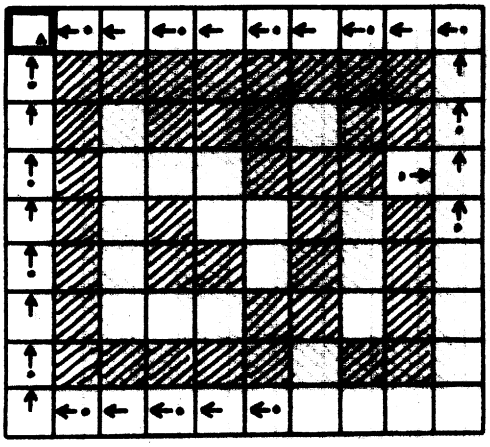


t = 4

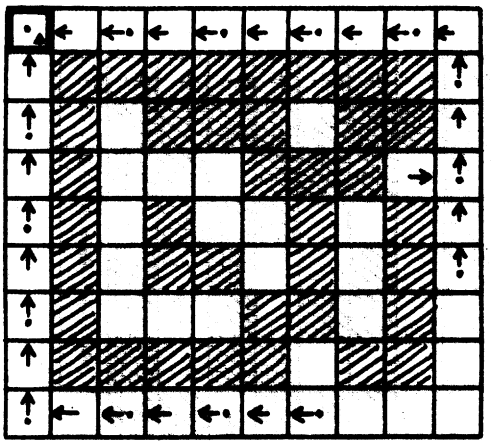


The contagion and signaling process are under way. At t = 14 the contagion branches for the first time.

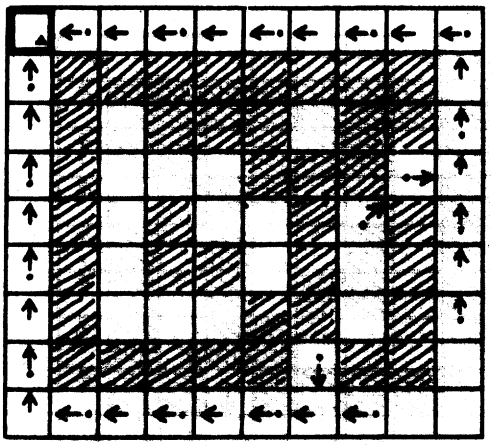
t = 14



t = 15



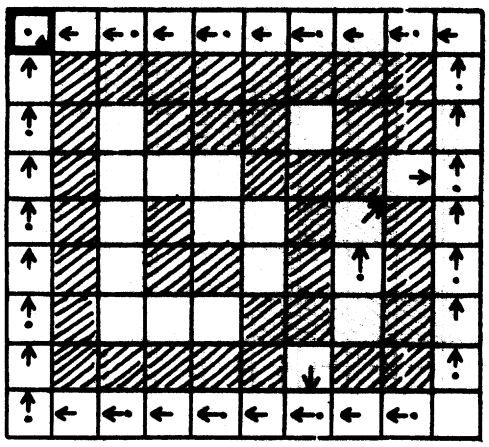
t = 16



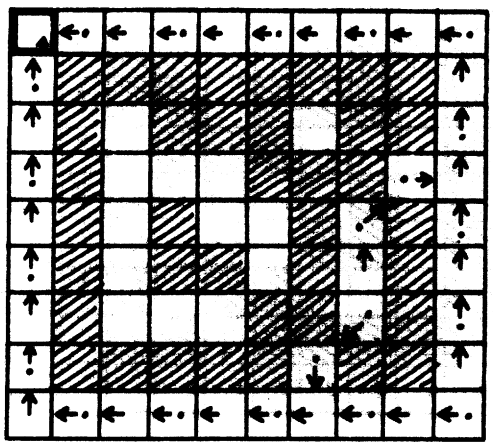
At t = 16 we observe the first case of the contagion spreading through a diagonal connection between white cells (from cell (4,9) to cell (5,8)). Note that the contagion takes two time units to pass through the connection because of our nearest neighbor model. Note also that cell (4,9) has generated a second dot at t = 16 to avoid a gap in the dot sequence which would be created if it simply waited to transmit the dot generated by cell (5,8). In general any white cell which catches the contagion and which has diagonal white neighbors will generate a second dot in this manner.

We now watch the contagion come to an end.

t = 17



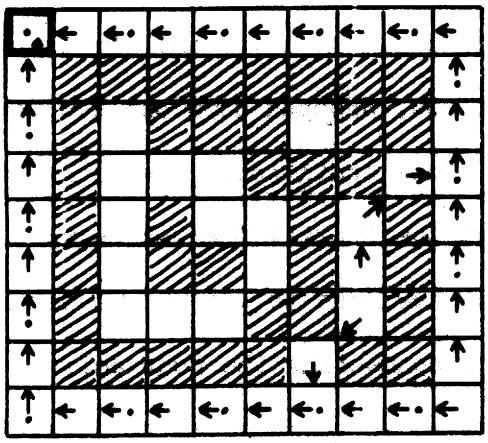
t = 18



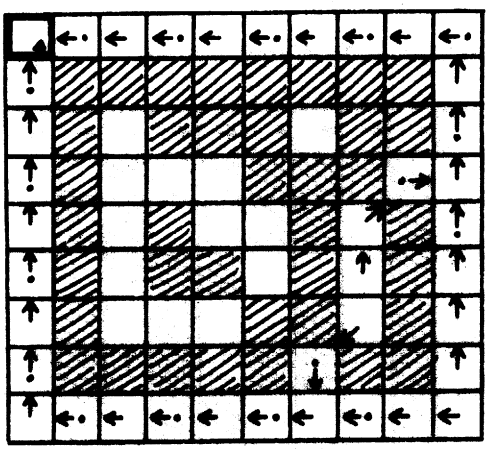
Note that the contagion arrived at cells (7,8) and (9,10) from two directions at once and that these cells made a choice of direction. Any choice will do.

We now see the last of the dots begin to move toward the scanner.

t = 19

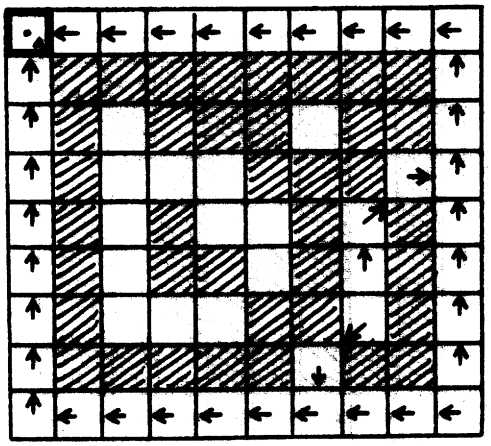


t = 20



By t = 35 the last dot has arrived at the scanner.

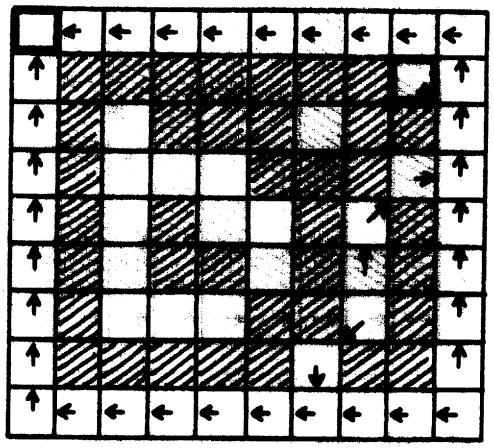
t = 35



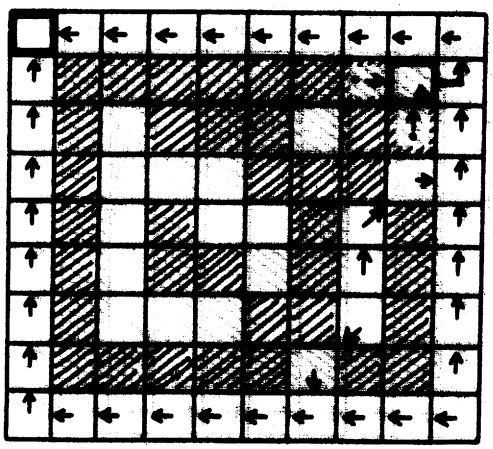
At t = 36 the scanner sees that no nore dots will arrive and by t = 37 it has begun to continue its scan. By t = 47 it has completed its scan of the first row, dropped down one row, and has encountered an unprocessed cell. Once again contagion within the new area begins. In addition we now see the linking process begin to take place. The pathway connecting the new node to its parent in the tree is grown cell by cell simply by connecting the node cell to the cell from which the scanner arrived and continuing to grow the pathway in the direction indicated by the arrows.

The first three steps are shown below.

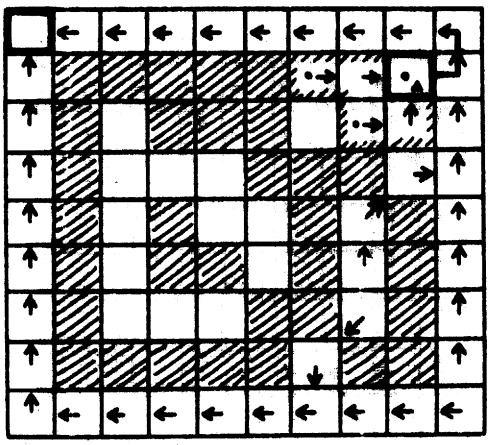
t = 47



t = 48

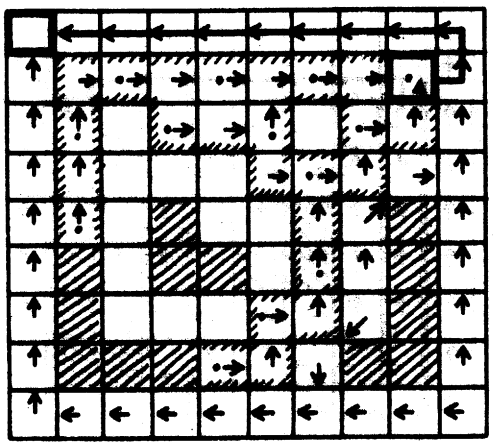


t = 49



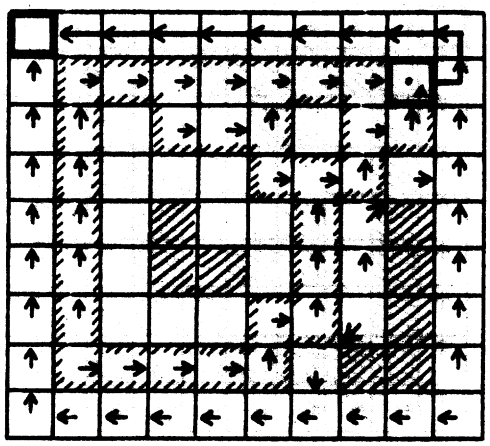
By t = 57 the pathway has been completed, but the contagion is still spreading.

t = 57



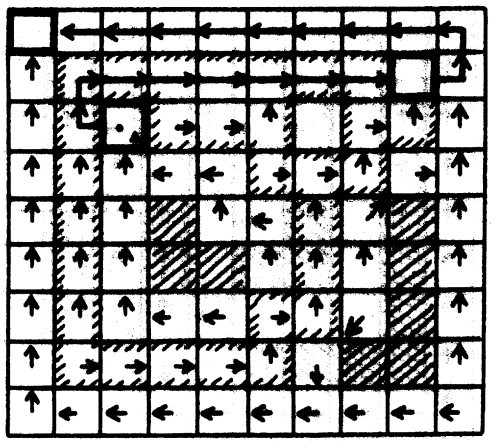
By t = 73 the contagion is dead and the last dot has arrived back at the scanner.

t = 73



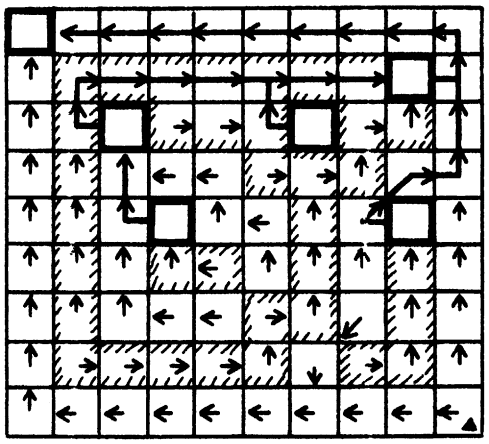
By t = 97 a third area has been processed and linked and the scanner is preparing to continue its scan.

t = 97



At t = 179 the scanner has completed its scan and all processing is complete.

t = 179

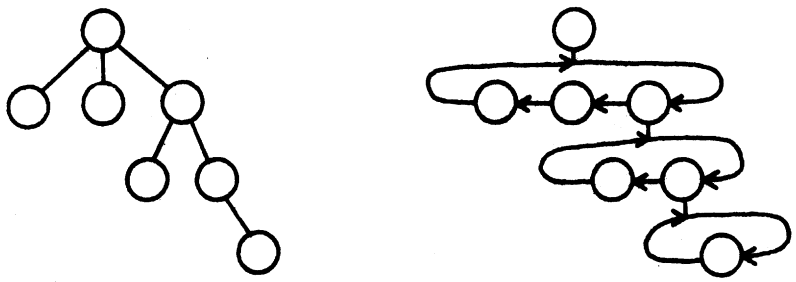


Although the process was complete in our example by the time the scanner finished its scan, this is not always the case. The scanner waits at each node until the contagion is dead, but does not wait until the connecting pathway has been established. Thus it might be the case that some pathways are still not complete by the time the scanner arrives at the end of its scan. Since the longest pathway may be about $\frac{1}{2}m \cdot n$ in length, we must wait this additional amount of time to insure that processing is complete.

The entire process of converting a figure into a tree representation takes time proportional to $m \cdot n$. The scanner is in motion for about $m \cdot n$ units and at rest for a total time less than or equal to twice the sum of the areas. Counting the waiting time for completion of connections, the entire process will be complete in at most $3\frac{1}{2}(m \cdot n)$ units.

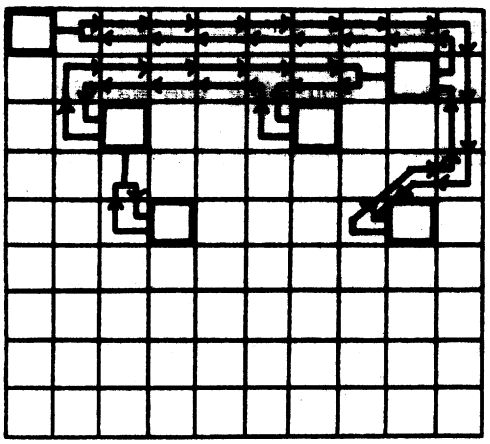
The tree which was produced in our example contains six nodes corresponding to areas and two false nodes created by the confluence of pathways. The false nodes located in cells (2,6) and (2,10) are to

be ignored in our interpretation of the results. The creation of false nodes is a necessary evil since the trees we are trying to represent may have nodes of arbitrarily high degree. One way of ignoring the false nodes is to think of a tree-like structure of loops rather than of an actual tree. In such a loop structure the descendents of a node are threaded on a loop which begins and ends at the node. The diagram below shows a tree T and the corresponding loop structure L.

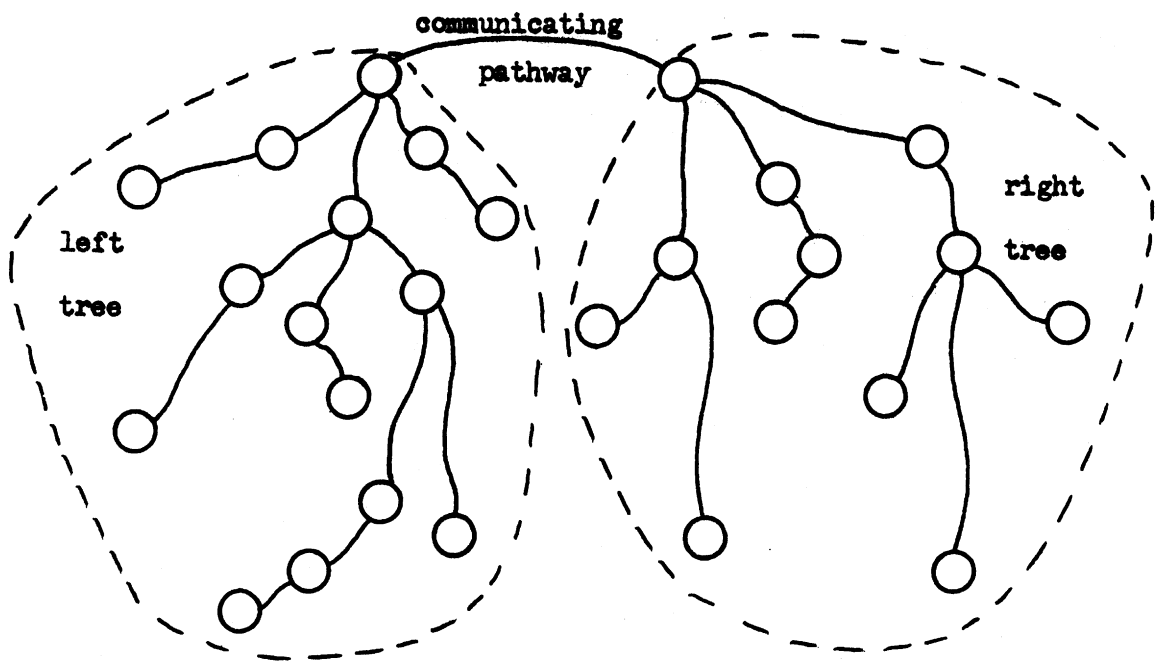


T L

The algorithm for converting figures to trees which was described above can be modified to produce such loop structures simply by making the connecting pathways double. The loop structure corresponding to our example is shown below.



Comparing trees: Having shown how to convert figures to tree representations, we now must describe how two tree representations may be compared for isomorphism. The methods to be described will make use only of the cells which actually form the tree representations for the left and right figures, plus a pathway of cells connecting the background node of the left tree to the background node of the right tree. The pathway connecting the background nodes is used for communication between the cells in the left and right trees. Imagine now that we have deleted all cells from the array except those comprising the trees and the communicating pathway. If we grasp the left background node in one hand and the right background node in the other hand and lift, the trees will untangle and hang down. One has an image of something like the following diagram.

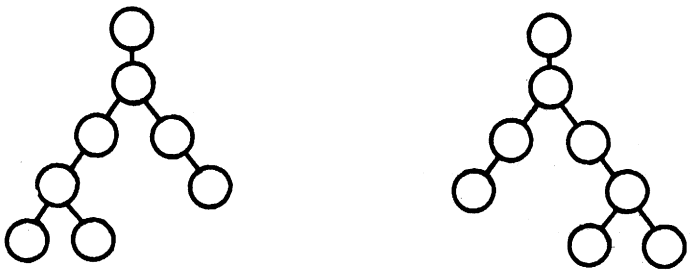


This is the form in which we imagine the cells to be arranged purely to avoid thinking about their actual embedding in the array and to be able to use words such as "above" and "below" in describing the relationship between nodes. Since no path lengths were altered in the lifting process, communication times have not been altered.

Recursive method: The first method of isomorphism-checking to be considered is a straight forward recursive procedure. We think of two observers, one standing on each background node, who can speak to each other via the communicating pathway. Our observers agree to check whether the trees are isomorphic. They do this by first comparing the degrees of the nodes on which they are standing. If the degrees differ, then the trees are certainly non-isomorphic and the process terminates. If the degrees are both zero (ignoring the communication pathway), then the trees are certainly isomorphic and the process terminates. If the degrees are equal but non-zero, then further checking is required. The checking involves comparing each subtree below the node on which the left observer is standing with each subtree below the node on which the right observer is standing. The comparisons are carried out in a recursive manner by the observers themselves. They begin methodically to make all pairwise comparisons of one left subtree with one right subtree. Each time a matching pair is found, they are marked and eliminated from consideration. Eventually either a one-to-one correspondence of subtrees will be found, in which case the trees are isomorphic, or a subtree will be found which has no matching subtree, in which case the trees are non-isomorphic.

Description generation method: The second method of isomorphism checking might be called the description generation method. Only the central idea will be sketched here. Assume that we have established a set of conventions for describing finite rooted trees by strings of symbols. Furthermore assume that our system is canonical in the sense that two trees which are isomorphic have identical descriptions. Such a system could for instance be a parenthesis system in which a tree is described by an opening parenthesis followed by the descriptions of each of its subtrees in lexicographical order followed by a closing parenthesis. Having established such a system, we have the cells in a tree representation carry out organized activity in such a manner that the cell corresponding to the root node emits the canonical description of the tree. Given two trees, we simply compare the descriptions they emit. The trees are isomorphic if and only if the descriptions are identical. A problem which arises with this method is that one tree may emit symbols faster than another, but by organizing additional cells into a variable length stack, such inequities in generation speed may be ignored.

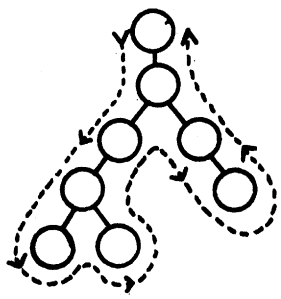
Squad car method: The final method of performing the topological match might be called the squad car method. It is similar to the recursive method described above, but is more highly parallel. It is in its simplest form when applied to the comparison of binary trees, so we will consider this case first. Let $B_1$ and $B_2$ be two rooted binary trees as in the figure on the following page.



$B_1$ $B_2$

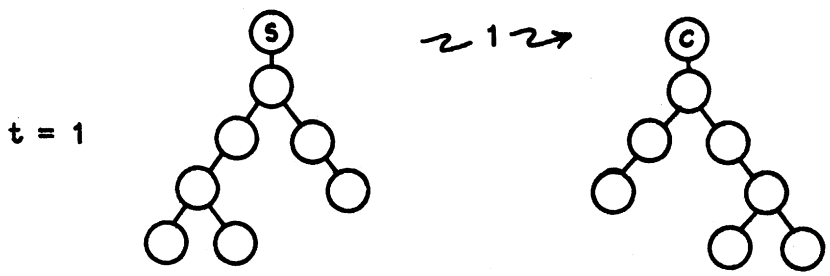
The comparison process goes as follows. A radio equipped squad car, S, will drive over the branches of $B_1$ reporting over its radio a description of its route. Simultaneously a second car, C, will begin to drive over the branches of $B_2$ while listening on its radio to the description of the route taken by S. The car C will attempt to follow an isomorphic route. If C is successful, it will eventually arrive back at the root of $B_2$ and the trees will have been proved isomorphic. If C is unsuccessful, it will, as we shall see, vanish somewhere along its route and hence never return to the root of $B_1$.

The route taken by S in driving over $B_1$ is a simple right-hand turn method for traversing a tree. When S approaches a node of degree three from above, it leaves by the (from S's point of view) right-hand branch. Upon returning to the node from below, S then takes the second descending branch. Upon return to the node again, S leaves by the upper branch. The path of S over $B_1$ is shown in the diagram on the following page.

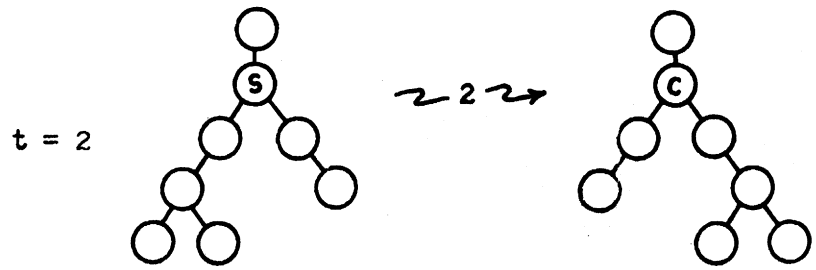


The description which S broadcasts is as follows. Upon entering a node for the first time from above, it broadcasts the number of descending branches leading from the node (0, 1, or 2). On all subsequent visits, it simply broadcasts (N) to indicate that it has returned to a previous node. The complete sequence of symbols as broadcast by S when driving over $B_1$ would be 1, 2, 1, 2, 0, N, 0, N, N, N, 1, 0, N, N, N. (One could use instead a system of parentheses, but the system described above is more attuned to the task at hand.)

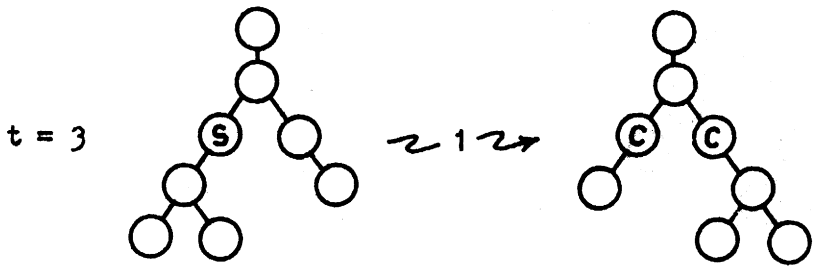
Now let us follow our example as S traverses $B_1$ and C attempts to find an isomorphic path in $B_2$.




At t = 1 both cars are about to begin and are sitting on the root nodes. S has already transmitted its first report (1), but it has not yet been received by C.

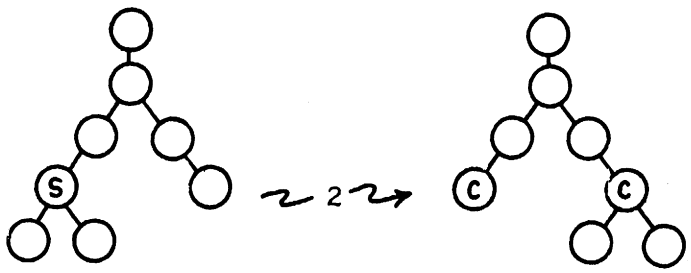



By t = 2, C has received the 1 emitted by S at t = 1, has found its current path to be in agreement with the 1, and both cars have advanced to the next node. (We ignore for the moment the time it takes to transmit messages and the time it takes to move from node to node.) S has transmitted a 2. When C receives the 2, it will find that it is indeed sitting on a node of the appropriate degree. Which branch should it take next? A deterministic car would get one choice and might make the wrong one. A non-deterministic car could be assumed to make the correct choice. Our car however is a parallel car and it makes both choices. That is, it splits into two identical cars and each car takes a different branch leading from the node.



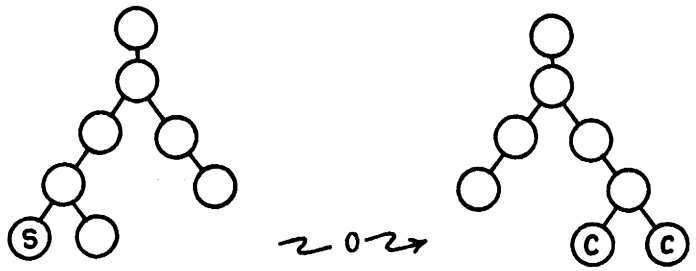

By t = 3, S has moved to the next node and our bilocating car C has moved down to the next two nodes.

t = 4



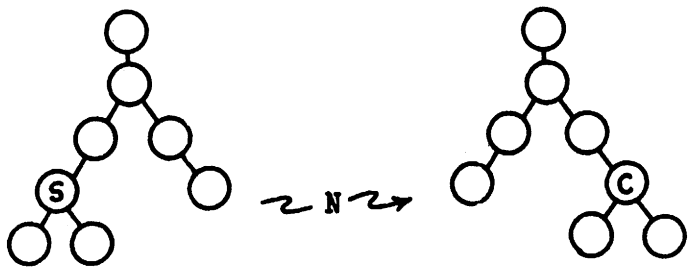

By t = 4, S has arrived at a doubly branching node and has transmitted a 2. Both copies of C have advanced one node. When the left-hand copy of C receives the 2, it will simply vanish since it will then be apparent that it can no longer follow a route similar to that followed by S. The right-hand copy of C will be in agreement with the 2, will again split, and will advance to the next two nodes.

t = 5



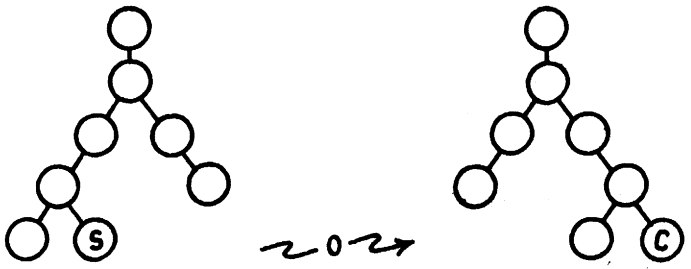

Both copies of C will agree with the 0 now being received and will attempt to return to the previous node. When a car splits, leaves a node in two directions, and returns from both directions, it indicates that both branches lead to isomorphic subtrees. One of the cars can simply vanish, leaving the other car to complete the process.

t = 6




By t = 6, one car remains to continue the processing.

t = 7




By t = 7, C has descended a node and is still in agreement with the description being given by S. From here on out C will simply mirror the movements of S and the trees will be found to be isomorphic. The entire process has taken time proportional to the number of nodes in $B_1$.

A point of interest is that the number of distinct isomorphisms between two trees can be easily obtained from the above method. Each time two cars return to a node and merge, a marker is placed at that node indicating that the two subtrees below are isomorphic. If by the end of the process the trees turn out to be isomorphic, then there will be $2^n$ distinct isomorphisms, where n is the number of marked nodes.

When we come to apply the squad car method of isomorphism checking to the tree representations which have been generated from

figures, we encounter two difficulties. First the trees are not binary, but contain nodes of arbitrarily high degree. Second the internode distances vary widely.

The problem of the existence of nodes of arbitrarily high degree is easily overcome. When S arrives at a new node, it must report the degree of that node. In the binary case this could be done by using a fixed set of symbols, since only three possibilities could arise. In the current case S can report the degree of a newly encountered node in unary by driving along the loop (thinking in terms of a loop structure for a moment) on which the descendents of that node are located and transmitting a "1" every time it passes a node. When S completes its circuit of the loop and arrives back at the original node, it transmits an end of number signal. The degree of the corresponding node or nodes on which C is located is checked by having C traverse the corresponding loop or loops, advancing by one node as each "1" is received. Any copy of C which is not back at its starting node when the end of number signal arrives is on the wrong track and will vanish. Provisions for handling highly multiple splitting and recombination of C in $B_2$ must also be established. This can be accomplished by leaving appropriate markers at the nodes.

The problem of varying distances between nodes appears to be more serious in that the methods of overcoming it seem to boost the total time required for the process from something on the order of the path lengths of the trees to something on the order of the product of the pathlengths of the trees. Consider for a moment what took place in the naive binary picture presented above. The squad car S drove over

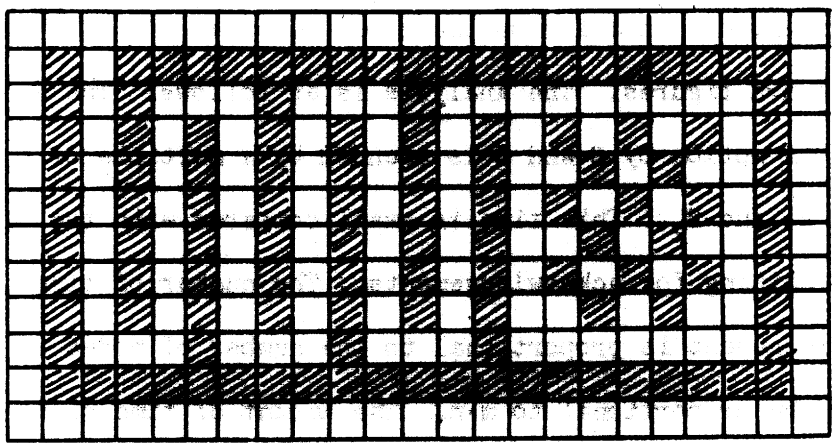
$B_1$ and transmitted a description of its route. Since S received no feedback from C, the rate at which S progressed was independent of any difficulties encountered by C. In particular if S and C drive at the same rate, then the branch lengths in $B_2$ must be shorter on the average than those in S or else C will become overloaded with the information transmitted by S. In the case of the tree representations which we have generated from figures, there is no correspondence between branch lengths in $B_1$ and in $B_2$. One way of circumventing this difficulty is to allow the signals from S to pile up in a queue at C. However, C moves up and down within the tree and there may be multiple copies of C present which may be able to process the signals from S at different rates. By the time the movement and multiplicity of C has been accounted for, one ends up with a process which is proportional to the product of the path lengths of $B_1$ and $B_2$.

A simpler method which yields the same processing time is to have S wait for acknowledgement after each transmission. Thus S makes a unary degree count and then waits for an acknowledgement from C. The signal sequence produced by S travels up the branches of $B_1$, across the communication pathway, and down the branches of $B_2$. As the signals travel down the branches of $B_2$, they split at each node of degree greater than two and a copy goes down each descending branch. Each copy of C will receive the signals, process them, and send back an acknowledgement, "A". The "A"'s travel up the branches of $B_2$. When an "A" arrives at the node of degree greater than two, it waits for the remaining "A"'s to arrive from the other descending branches. When they have all arrived, they combine into a single "A", which

then continues up the tree. It now becomes important that when a copy of C vanishes, something be left behind which can still acknowledge signals, for otherwise no acknowledgement would ever be received from a branch down which a C had ventured and disappeared.

The squad car method as modified by unary degree counting and acknowledgement of transmission takes on the order of $(m \cdot n)^2$ to compare two m*n figures. The analysis is as follows. The amount of time spent in motion by S and C is bounded by a multiple of the path lengths of $B_1$ and $B_2$. This time is small compared with the time spent by S waiting for acknowledgements. The number of times S waits for acknowledgement is proportional to the number of nodes in $B_1$ which in turn may be proportional to m·n. The amount of time spent waiting for a single acknowledgement is primarily a function of the distance between S and the furthest (in terms of path length in the combined trees) copy of C. The latter distance may also be proportional to m·n. Thus the total time S spends waiting for acknowledgements is proportional to $(m \cdot n)^2$. Of course some figure pairs may be found to be non-isomorphic soon after the tree representations have been generated. Other figures however will take much longer.

The diagram below shows a type of figure whose tree representation has a number of nodes proportional to (m·n) and a depth (measured from the root to the deepest node) proportional to (m·n). Such figures will require a long time to check against similar figures.



# CHAPTER 4

# DISPLAY PROBLEMS

The preceding two chapters dealt with the use of arrays as devices whose input was a figure and whose output was a single binary symbol. We now study arrays as devices whose input is a figure and whose output is also a figure.

## 4.1 Introduction

If the arrangement of initial states in an array can be viewed as an input figure, then by allowing the array to run until every cell is in a final state, we may view the arrangement of final states as an output figure. In this way an array or more properly a cell type may be viewed as a transformation from figures over the initial states into figures over the final states. An array operating in this mode is said to be working on a display problem. The following definitions formalize these concepts.

Definition

Given two sets I and F, a figure transformation (or transformation for short), $\mathcal{T}$, of type I/F is a function from the set of figures over I into the set of figures over F, such that the image of an m×n figure is always an m×n figure. If P is a figure over I, we denote its image under $\mathcal{T}$ by $\mathcal{T}(P)$.

**Definition**

Given a transformation $\mathcal{T}$ and a cell type M, we say that M <u>implements</u> $\mathcal{T}$, if

i) $\mathcal{T}$ is a transformation from figures over the set of initial states of M into figures over the final states of M

ii) Given any m×n computation $\mathcal{D} = D^0, D^1, D^2, \ldots$ of type M, there exists an integer k such that $\mathcal{T}(D^0) = D^k$.

**Definition**

Given a transformation $\mathcal{T}$ we say that it is <u>cellular</u> if there exists a cell type M which implements it.

**Definition**

If T(m,n) is a real-valued function, we say that <u>M implements $\mathcal{T}$ within time T(m,n)</u> if M implements $\mathcal{T}$ and for every m×n computation $\mathcal{D} = D^0, D^1, D^2, \ldots$ there exists an integer k such that $\mathcal{T}(D^0) = D^k$ and $k \leq T(m,n)$.

If there exist integers p, q, r such that M implements $\mathcal{T}$ within time pm + qn + r, we say that $\mathcal{T}$ is <u>linear</u>.

Many of the theorems on recognition such as Theorem 2.3, the Minimizing Theorem, have immediate analogs for display. Since nothing

really new is contained in these analogs, we will omit them. The Speed-Up Theorem however is slightly different in form in the case of display.

Theorem 4.1 (Speed-Up)

Let $\mathcal{T}$ be a transformation and let M be a cell type which implements $\mathcal{T}$ within time T(m,n). Then given any positive integer k, there exists a cell type $M_k$ which implements $\mathcal{T}$ within

$$\frac{1}{k}\cdot(T(m,n)) + 2(1 + \frac{1}{k})(m + n) - 2.$$

PROOF: As in Theorem 2.2 and Corollary 2.2.1, we perform packing followed by speeded-up simulation. According to Corollary 2.2.1, we will have all simulated cells in their final states by time

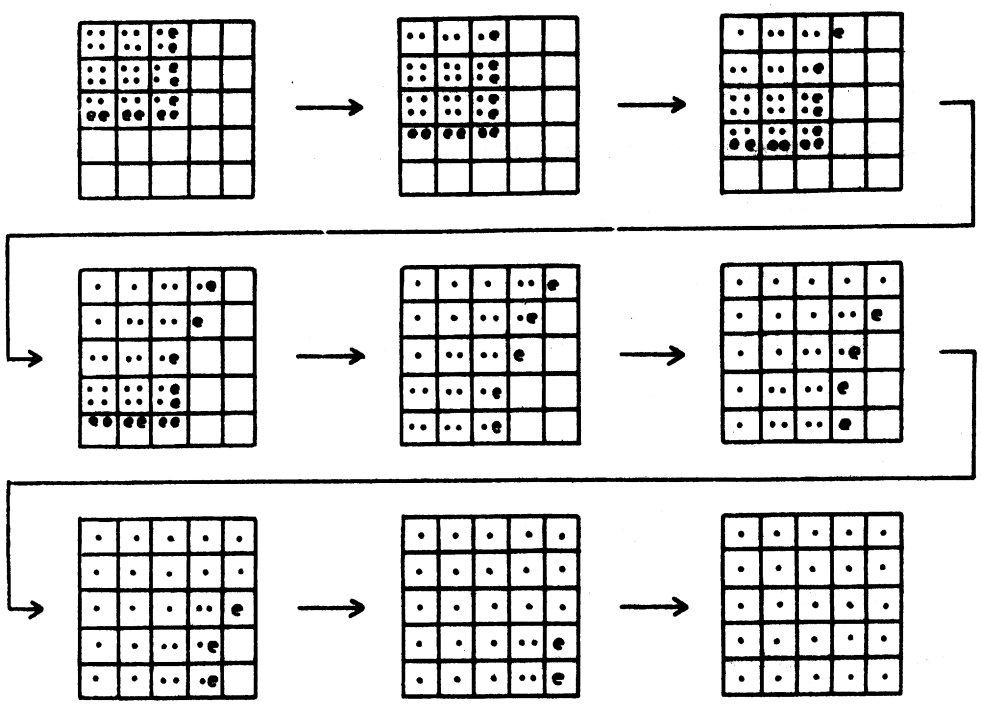
$$t = m + n - \left[\frac{m}{k}\right] - \left[\frac{n}{k}\right] + \left[\frac{T(m,n) - 1}{k}\right] + 2.$$

Unlike the recognition case where no more remains to be done, we must, in the case of display, unpack the results so that the output figure can be displayed by the array.

Unpacking may begin as soon as all the simulated cells have entered their final states. If it were the case that all the simulated cells entered their final state at the same time, we could begin unpacking at that time. However, since different cells may enter final states at different times, we need a method of deciding when the last cell has entered its final state. This is done by having a module generate a completion signal as soon as all of the cells it is simulating have entered their final states. The completion signals originating on the southern edge of the packed

portion of the array flow to the north in the column in which they originated. If a signal arrives at a module, all of whose simulated cells have not entered final states, it waits until they do enter final states before continuing. Thus when a completion signal arrives at a module on the northern edge, it signifies that all of the simulated cells in that column are in final states. In a similar manner each northern edge module passes the completion signal on to its western neighbor as soon as it receives completion signals from its southern and eastern neighbors. Thus in no more than $\left[\frac{m}{k}\right] + \left[\frac{n}{k}\right]$ units after the last simulated cell has entered its final state, the northwest corner module will be aware of this fact. It can then initiate a two-dimensional firing squad in the packed portion of the array. When the firing squad goes off, the unpacking will begin. Assuming that the firing squad takes $2(\left[\frac{n}{k}\right] + \left[\frac{m}{k}\right]) - 4$ units to go off, the unpacking operation will get under way no later than $t = m + n + 2(\left[\frac{m}{k}\right] + \left[\frac{n}{k}\right]) - 2 + \left[\frac{T(m,n) - 1}{k}\right]$. (We will see in Section 4 that this firing squad time can be improved slightly.)

The unpacking process is quite simple and is illustrated on the following page for a 5×5 case with k = 2.



The unpacking process takes $m + n$ steps. (Note information can be packed faster than it can be unpacked.) Thus the entire simulation from the beginning of packing to the end of unpacking takes

$$2(m + n + \left[\frac{m}{k}\right] + \left[\frac{n}{k}\right] - 1) + \left[\frac{T(m,n)-1}{k}\right] \leqslant \frac{1}{k} \cdot T(m,n) + 2(1 + \frac{1}{k})(m + n) - 2$$

steps. □

We also have by a simple construction

Theorem 4.2

The composition of cellular (linear) transformations is a cellular (linear) transformation.

## 4.2 Specific Problems

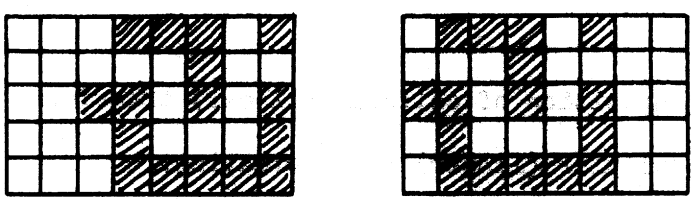
In this section we will consider some specific figure transformations and their implementation by iterative arrays. Some of these transformations are included because they are frequently proposed by people who are hearing about iterative arrays for the first time. Others have been included because they seem to be interesting transformations whose linearity is still open. The discussions are brief and informal.

The first set of transformations we consider are some simple geometric transformations of black and white figures which are usually proposed as "problems" by persons who are experiencing iterative programming for the first time. They are all linear transformations.

$\mathcal{T}_{TRANS}$(TRANSLATE)

This transformation is the result of translating the input figure rigidly to the west so that the westernmost black cell in the figure lies on the western edge of the array.

Example:



P $\qquad\qquad$ $\mathcal{T}_{TRANS}(P)$

$\mathcal{T}_{TRANS}$ may be implemented in linear time as follows:

i) The figure casts a shadow downwards onto the southern edge.

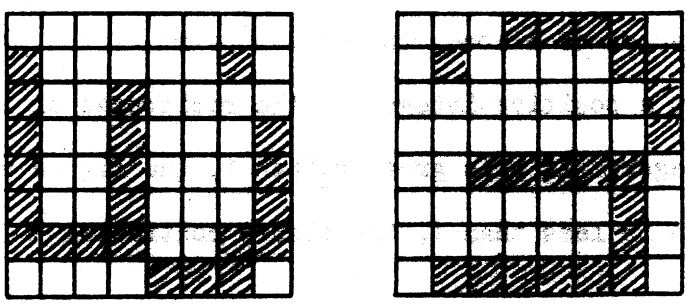
ii) The western-most point of the shadow is detected and a line is drawn to the northern edge from this point. The figure has now been enclosed in a block of cells and is tangent to the western edge of the block.

iii) Using shifting techniques similar to those used in packing and unpacking figures for speed-up (see Theorems 2.2 and 4.1), shift the block to the west so that it is tangent to the western edge.

$\mathcal{T}_{ROT}$ (ROTATE)

This transformation applies only to square ($m = n$) arrays. The transformation is the result of rotating the input figure about its central point by 90° counterclockwise.

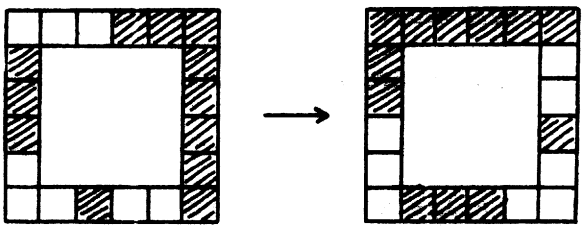
Example:



P $\qquad\qquad$ $\mathcal{T}_{ROT}(P)$

First observe that a hollow square of cells can shift the information it contains by 90° counterclockwise in

time proportional to the edge of the square. Such a shift is indicated below.



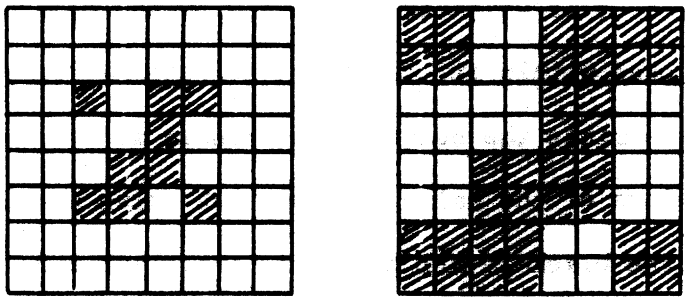
Then $\mathcal{T}_{ROT}$ may be implemented in linear time by first organizing the array into a set of concentric hollow squares and then having each hollow square shift its information by 90° counterclockwise.

One might ask for rotations of other than multiples of 90°. A major difficulty then becomes the question of just what the result of such a transformation should be. This question in turn leads to the area of approximation techniques for figures which is outside the scope of the current work.

$\mathcal{T}_{DIL(n)}$ (DILATE)

This transformation causes a dilation of the figure by a factor of n about the center point of the array, where n is a positive integer or the reciprocal of a positive integer. If n is greater than 1, the figure enlarges; if n is less than 1, the figure contracts.

Example:



P $\qquad\qquad$ $\mathcal{T}_{DIL(2)}(P)$

$\mathcal{T}_{DIL(n)}$ may be implemented in linear time as follows:

i) The center point of the array is determined and each cell determines which of the four rectangular quadrants it lies in.

ii) The portion of the figure within each quadrant is dilated by a factor of n, either away from or toward the central point.

The process of expanding or contracting a quadrant is similar to the process of packing or unpacking for the purpose of speed-up as described in the proofs of Theorems 2.2 and 4.1. When expanding by a factor of k, each cell in the quadrant pretends that it is a module in which a k×k block of cells has been packed. The color of these cells is taken to be the color of the module. When the cells have been unpacked, they will cover a k×k area in the figure. That is, the original cell will have expanded by a linear factor of k. The process of a contraction is similar. After a k×k block

of cells has been packed into a module, that module takes on a color which is determined by the colors of the cells packed within it. In this case one has to fix a rule which determines the color of the module as a function of the colors of the cells. Three such rules are (1) the module turns black if all the cells are black and white otherwise, (2) the module turns black if any of the cells are black and white otherwise, and (3) the module turns black if more than half of the cells are black and white otherwise..

In carrying out a contraction, one always has enough room in the array to represent the image of the transformation. In carrying out an expansion, however, there may not be enough room in the array to contain the resulting figure. (When we say the array contains the resulting figure, we of course mean that the array contains all the black cells in the resulting figure.) We may simply truncate the expanded figure at the boundary of the array or we may use the concept of folding to preserve the entire expanded figure for further processing, although it cannot be displayed. Consider the similarity in structure between a piece of paper which has been folded in half once with the fold on the left and an iterative array of two layers in which we think of the layers as being connected only at the western edge. If the array has dimension m×n, we have in effect an array of dimension m×2n which has been folded over once. This is the concept of <u>folding</u>. By using more layers, more complicated folding may be simulated. In particular, given any positive integer

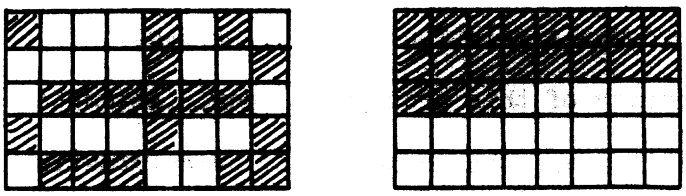
k, we may design an array which, by using folding, acts as though the initial figure is centered on an array of size km×kn. Thus given any positive k, we may use folding to design a cell type which can expand any input figure by a factor of k although it cannot display that figure in the usual sense. Finally given any positive rational number, q, we may perform a dilation by a factor of q by first expanding by the numerator and then contracting by the denominator.

The next set of transformations to be considered are known to be solvable in time proportional to m·n, but their linearity is open.

$\mathcal{T}_{PACK}$ (PACKING)

This transformation accepts black and white figures and produces a figure which has the same number of black cells, but in which these cells are packed into solid rows at the top of the figure.

Example:



P $\qquad\qquad$ $\mathcal{T}_{PACK}(P)$

There are several simple ways of achieving this packing in time proportional to mn. We will leave them for the interested reader to discover.

The packing transformation is related to the recognition of $\Psi_{EULER}$ which was defined in Section 3.5. The recognition of that

predicate had been reduced to determining whether there were more +1's than -1's distributed about the array. If $\mathcal{J}_{PACK}$ is linear, then in linear time the +1's could be packed in one layer, the -1's in another layer, and the majority determined. Thus the linearity of $\mathcal{J}_{PACK}$ implies the linearity of $\Psi_{EULER}$. It would also imply the linearity of $\Psi_{BLACK}$, defined to be the set of all black and white figures which have more black than white cells. Currently the linearity of $\Psi_{BLACK}$ is still open.

$\mathcal{J}_{REP}$ (REPRESENTATIVE)

The problem is to transform a black and white figure into a black, red, and white figure by turning exactly one cell of each component or hole red. This transformation thus corresponds to the process of selecting a representative cell from each hole and component. The tree generation process described in Section 3.6 can be easily modified to produce a solution to the representative problem. The linearity of this problem is open. Attempts have been made to adapt the connectivity transformation to this problem, but without success.

Note: In the above problem as in the following two we do not actually specify the transformation in complete detail. Instead we indicate some general properties which the transformation must have. For example in the problem above we do not care which cell in a given component turns red, so long as exactly one turns red. The problem

is to find a cell type which implements in linear time a transformation having the properties given.

$\mathcal{T}_{SP}$ (SHORTEST PATH)

Given a solvable maze, draw a shortest path solution to the maze in red. Theorem 3.4 states that solvable mazes can be recognized in linear time. Here we ask that a shortest path solution be displayed in linear time. We might be less ambitious and ask can any solution to the maze be displayed in linear time, since this problem is open also. A solution to the shortest path problem can be obtained in time proportional to m·n by a simple adaptation of the contagion process described in Section 3.6.

$\mathcal{T}_{GP}$ (GOLD PLATE)

Given a black and white treasure map (black islands in white water) on which is indicated the location of a chest of gold (by a gold cell), display the map obtained by coloring gold the entire island on which the gold is located. (Or equivalently erase all islands on which the gold is not located.) This process is easily carried out in time proportional to m·n by a contagion process. Its linearity is open.

## 4.3 Two-Dimensional Firing Squad

In this section we consider a two-dimensional analog of the well-known firing squad synchronization problem. (See Section 1.2.) This problem is strictly speaking not a display problem, but is more related to display than recognition, so we have included it in this chapter. As has been seen in the proof of Theorems 2.2 and 4.1, a solution to the firing squad problem is a useful tool in solving other iterative array problems.

The two-dimensional firing squad problem may be stated as follows. Design a cell type M such that the following conditions hold:

i) The initial states of M are g (general) and s (soldier).

ii) The final state of M is f (fire).

iii) The soldier state is dormant relative to soldier and edge states. That is

$$\bar{g}\left(\begin{array}{ccc} & e|s & \\ e|s & s & e|s \\ & e|s & \end{array}\right) = s$$

where $\bar{g}$ is the transition function and e|s represents either an edge state or a soldier state.

iv) Given any m×n array of type M in which the initial configuration consists of one cell in the general state at location (1,1) and all other cells in the soldier state, then there exists an integer k such

that all cells enter state f for the first time at t = k.

A thorough discussion of this problem for the case where m = 1 is given in Balzer[3] where he proves the existence of an eight state solution (nine state solution in our formalism) in which the soldiers fire at time t = 2n - 2. This solution or any solution to the one-dimensional case may be used to construct a solution to the two-dimensional problem as follows. A firing squad activity is organized in row 1 of the array by the general in cell (1,1). At time 2n - 2 each soldier in row 1 instead of entering the firing state, becomes a general. (One of the largest and most drastic field promotions in the annals of military history.) These new generals, together with the old general, now organize firing squad activity in their respective columns. Since this activity begins in each column at the same time (t = 2n - 2) and since the columns are all of length m, we have that the entire array will enter the firing state at time

$$t = 2n - 2 + 2m - 2 = 2(m + n) - 4.$$

The solution to the two-dimensional firing squad problem which was presented in the previous paragraph is not optimal in terms of time. Before considering a faster method, let us find a lower bound on the time required for a solution.

Theorem 4.3

Any solution to the two-dimensional firing squad problem will require at least $m + n + \max\{m,n\} - 3$ units of time to enter the firing state on an m×n array.

PROOF: Let m, n be given and assume without loss of generality that $n \geq m$. We will show that cell (1,m) cannot fire before time $t = m + 2n - 3$. The idea is quite simple. Cell (1,m) cannot fire before it learns of the existence of the eastern edge of the array. The amount of time it takes a signal to travel from the general to the eastern edge and return to cell (1,m) is $m + 2n - 3$.

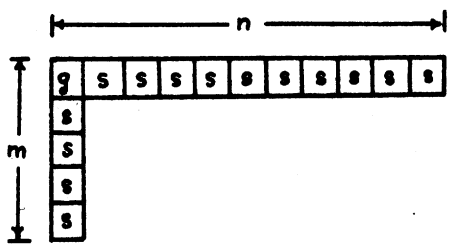
A more formal proof may be given using diagrams and the Interdependence Theorem, but the idea is the same. □

We now find that the lower bound given above is attainable.

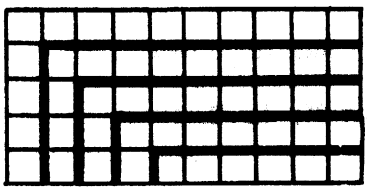
Theorem 4.4

There exists a solution to the two-dimensional firing squad problem which enters the firing state at $t = m + n + \max\{m,n\} - 3$.

PROOF: Moore and Langdon[12] demonstrate the existence of a seventeen state solution to what they term the generalized firing squad problem. This is a one-dimensional firing squad in which the general is located not necessarily at one end of the line of soldiers, but somewhere in the middle. If we bend such a line of soldiers by 90° at the general's location, we end up with an L-shaped firing squad as in the following diagram.



Moore and Langdon's solution, expressed in terms of m and n will cause the soldiers to fire at time $t = m + n + \max\{m,n\} - 3$. Now consider an m×n array which has been partitioned into a set of L-shaped components as in the diagram below. (Assume $n \geq m$.)



We will set up a Moore and Langdon solution to the firing squad problem in each of these L's. Notice that the corners of the L's lie in a diagonal line. The corner cell in the largest L knows it is a general at time t = 0 and can begin activity at once. A signal is made to propagate down the diagonal at the rate of one diagonal cell every three units of time. As the signal strikes each soldier on the diagonal, he becomes a general and initiates firing squad activity in his L. Thus the L whose general is in row i will begin activity at time $t = 3i - 3$ and hence will fire at time

$$t = 3i - 3 + (m - i + 1) + 2(n - i + 1) - 3 = m + 2n - 3.$$

That is all cells in the array will fire at the same time, $t = m + 2n - 3$. □

Letting m = n, we find that the above solution will cause the cells of a square array to fire at time $t = 3n - 3$, which is minimal by Theorem 4.3. Suppose however we ask for a solution to the firing squad problem which works only on square arrays. That is, we don't care what it does on arrays which are not square. Then it turns

out that we can obtain a solution in time 2n - 2 by using the same L partitioning. However instead of treating the entire L as a Moore and Langdon firing squad, we treat each half as a Balzer firing squad. In addition we initiate the squad in row i at time 2i - 2 rather than 3i - 3. This result yielding a time of 2n - 2 for a square (which is provably the best possible) was discovered in parallel by the members of Seymour Papert's class at M. I. T.

BIBLIOGRAPHY

(1) Atrubin, A. J., A Study of Several Planar Iterative Switching Circuits, Masters Thesis in Electrical Engineering, Massachusetts Institute of Technology, February, 1958.

(2) Atrubin, A. J., "A One-Dimensional Real-Time Iterative Multiplier," IEEE Transactions on Electronic Computers, EC-14, 3, June 1965, pp. 394-399.

(3) Balzer, R., "An 8-State Minimal Time Solution to the Firing Squad Synchronization Problem," Information and Control, 10 1, January 1967, pp. 22-42.

(4) Barnes, G. H., et al., "The ILLIAC IV Computer," IEEE Transactions on Computers, C-17, 8, August 1968, pp. 746-757.

(5) Blum, M. and C. Hewitt, "Automata on a Two-Dimensional Tape," IEEE Symposium on Switching and Automata Theory, 1967, pp. 155-160.

(6) Cole, S. N., Real-Time Computation by Iterative Arrays of Finite-State Machines, Doctoral Thesis in Applied Mathematics, Harvard University, August, 1964.

(7) Fischer, P. C., "Generation of Primes by a One-Dimensional Real-Time Iterative Array," J. ACM, 12, 3, July 1965, pp. 388-394.

(8) Hennie, F. C., Iterative Arrays of Logical Circuits, The MIT Press, Cambridge, Massachusetts, 1961.

(9) Lee, C. Y., "An Algorithm for Path Connections and Its Applications," IRE Transactions on Electronic Computers, EC-10, 3, September 1961, pp. 346-365.

(10) Minsky, M. and S. Papert, Perceptrons, The MIT Press, Cambridge, Massachusetts, 1969.

(11) Moore, E. F., Sequential Machines: Selected Papers, Addison - Wesley, Reading, Massachusetts, 1964, pp. 213-214.

(12) Moore, F. R. and G. C. Langdon, "A Generalized Firing Squad Problem," Information and Control, 12, 3, March, 1968, pp. 212-220.

(13) Murtha, J. C., "Highly Parallel Information Processing Systems," Advances in Computers, Vol. 7, Academic Press, New York, 1966, pp. 2-116.

(14) Rosenfeld, A. and J. L. Pfaltz, "Sequential Operations in Digital Picture Processing," J. ACM, 13, 4, October 1966, pp. 471-494.

(15) Unger, S. H., "A Computer Oriented Toward Spatial Problems," Proceedings of the IRE, 46, 10, October 1958, pp. 1744-1750.

(16) von Neumann, J. (ed. Burks), Theory of Self-Reproducing Automata, University of Illinois Press, Urbana, Illinois, 1966.

(17) Waksman, A., "An Optimal Solution to the Firing Squad Synchronization Problem," Information and Control, 9, 1, February 1966, pp. 66-78.

# INDEX


Accept a predicate 20
Accepted figure 19
Adjacent 65
Area 94
Array 13
  iterative -- 13
  m×n -- 17
  -- of type M 17
Associated tree 67
Atrubin, A. J. 8-9, 22
Automaton
  linear bounded -- 50
  pebble -- 50

Background 67
Balzer, R. 8, 33, 133
Barnes, G. H. 9
Blum, M. 11, 50, 65
Boolean/Finite Theorem 46
Burks, A. W. 7

Cartesian product, two-dimensional 16
Cell 13, 32
  primary -- 42
  secondary -- 42
  -- state 16
  -- type 16
Cellular
  -- predicate 46
  -- space 14
  -- transformation 120
Cole, S. N. 8-9, 31, 64
Complement of a predicate 20
Computation 19
Connected 66
Connectivity transformation 71
Components 66

Description 17
  initial -- 17
  -- generation method 109
Display problem 119
Distance 26
Don't care symbol 17
Dot 96

Edge state 14, 16
Effectively recognizable 58
Equivalent
  -- cell types 60
  -- in power 51
  topologically -- 68
Euler number 90

Figure 19
  accepted -- 19
  left -- 93
  rejected -- 19
  right -- 93
  -- transformation 119
Final state 15-16
Finite predicate 46
Firing squad 132
Fischer, P. C. 8-9, 31
Function, transition 16

Given 58

Hennie, F. C. 7-8, 22, 31, 50, 60
Hewitt, C. 11, 50, 65
Holes 66

Implement a transformation 120
Initial
  -- description 17
  -- state 15-16
Iterative array 13
Interdependence Theorem 28
Invariant, topologically 68
Isolated 67

Langdon, G. C. 8, 134
Layers 45
LBA 52
Lee, C. Y. 9
Left figure 93
Linear
  -- bounded automaton 50
  -- predicate 61
  -- transformation 120

Maze 86
multilevel -- 88
solvable -- 87
Methods of tree comparison
description generation 109
recursive 108
squad car 109
Minimizing Theorem 45
Minsky, M. 10, 21, 49, 65, 90
Module 32
Moore, E. F. 8
Moore, F. R. 8, 134
Murtha, J. C. 9
Myhill, J. 8

Neighborhood, d- 27
-- function of type M 27
Neighboring 65
Non-isolated 67
Non-uniform simulation 37
Non-Universality Theorem 51

Open questions
connectivity 88
construction of M' 62
linear transformations 129
linearity of predicates 64
multilevel mazes 89
topological matching 92

PA 52
Papert, S. 10, 21, 49, 65, 90
Paterson, Mike 62-63
Pebble automaton 50
Pfaltz, J. L. 9
Predicate 20
cellular -- 46
finite -- 46
linear -- 61
Primary cell 42
Problem
display -- 119
firing squad -- 132
recognition -- 15
topological match -- 92
Programming techniques
folding 128
layers 45
non-uniform simulation 37
progressive synchronization 35
Progressive synchronization 35

Recognition problem 15
Recognizable
effectively -- 58
within time T(m,n) 26
Recognize
-- a predicate 21
-- within time T(m,n) 25
Reject a predicate 20
Rejected figure 19
Right figure 93
Rosenfeld, A. 9
Scanner 95
Secondary cell 42
Solvable maze 87
Space, cellular 14
Speed-Up Theorem 38, 121
Squad car method 109
State
cell -- 16
edge -- 14, 16
final -- 15-16
initial -- 15-16
-- in a description 18
-- of cell at time t 19
Stationary point 73
Strictly more powerful 51
Successor 18
Synchronization
firing squad -- problem 132
progressive -- 35

Theorem
Boolean/Finite -- 46
Interdependence -- 28
Minimizing -- 45
Non-Universality -- 51
Speed-Up -- 38, 121
Topological match problem 92
Topologically
-- invariant 68
-- equivalent figures 68
Transformation
cellular -- 120
connectivity -- 71
implement a -- 120
linear -- 120
Transition function 16
Tree, associated 67
Two-dimensional
-- cartesian product 16
-- automaton/computer 50

Type, cell 16

Unger, S. H. 9
Universal computer 50

Von Neumann, J. 7

Waksman, A. 8

**CS-TR Scanning Project**
**Document Control Form**

Date : 2/2/96

**Report #** LCS-TR-66

Each of the following should be identified by a checkmark:
Originating Department:

☐ Artificial Intellegence Laboratory (AI)
☒ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

## Document Information

Number of pages: 142 (147 images)
Not to include DOD forms, printer instructions, etc... original pages only.

Originals are:
☐ Single-sided or
☒ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☒ Unknown ☐ Other:_____

Check each if included with document:

☐ DOD Form ☐ Funding Agent Form ☒ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number):_____

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description : Page Number:

IMAGE MAP (1-142) UNTITLED TITLE PAGE, 2-142
(143-147) SCANCONTROL COVER TRGT'S (3)

Scanning Agent Signoff:

Date Received: 2/2/96 Date Scanned: 2/14/96 Date Returned: 2/15/96

Scanning Agent Signature: Michael W. Cook

# Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

Scanned
Date: 2/14/1996
M.I.T. Libraries
Document Services