ACM MONOGRAPH SERIES

Published under the auspices of the Association for Computing Machinery Inc.

Edited by ROBERT L. ASHENHURST *The University of Chicago*

A. FINERMAN (Ed.) University Education in Computing Science, 1968
A. GINZBURG Algebraic Theory of Automata, 1968
E. F. CODD Cellular Automata, 1968

In preparation
G. ERNST AND A. NEWELL GPS: A Case Study in Generality and Problem Solving

Previously published and available from The Macmillan Company, New York City
G. SEBESTYN Decision Making Processes in Pattern Recognition, 1963
M. YOVITS (Ed.) Large Capacity Memory Techniques for Computing Systems, 1962
V. KRYLOV Approximate Calculation of Integrals (Translated by A. H. Stroud), 1962

CELLULAR AUTOMATA

E. F. Codd

Research Laboratory
IBM Corporation
San Jose, California

ACADEMIC PRESS, Inc. New York and London 1968

ACADEMIC PRESS, INC.
111 Fifth Avenue, New York, New York 10003

United Kingdom Edition published by
ACADEMIC PRESS, INC. (LONDON) LTD.
Berkeley Square House, London W.1

LIBRARY OF CONGRESS CATALOG CARD NUMBER: 68-23486

PRINTED IN THE UNITED STATES OF AMERICA

PREFACE

There is a growing interest in large computing systems which operate in a highly parallel manner, in computing machines which can construct copies or variants of themselves, and in models for biological reproduction. Cellular automata provide a common basis for investigations in all these areas. The major pioneering work in cellular automata is that of von Neumann and is published elsewhere. In this book extensions of von Neumann's work are presented in a self-contained way which assumes no prior knowledge of cellular automata.

The material in this book should be helpful to computer designers and programmers who want a better understanding of the principles of homogeneous cellular systems; to automata theoreticians who may be interested in taking this work to a new level of generality; and to biochemists interested in the possibility of biochemical computers with self-reproducing capability.

The treatment is partly mathematical and partly logical (in the computer design sense of the word). Both microprogramming and programming appear in a primitive form. Physical realizations of cellular automata are not discussed.

Some theorems are discussed which deal with conditions under which universal computation and construction can be exhibited in cellular spaces. A design is presented for a machine embedded in a cellular space – a machine which can compute all computable functions and construct a replica of itself in any quiescent, accessible, and sufficiently large region of the space. Simulation of one cellular space by another is also discussed. This is a problem of considerably greater difficulty than simulating a cellular space by means of a general purpose digital computer. Finally, the use of a digital computer for research in cellular automata is described.

The material in the first three chapters can be understood without any special aids, although the reader who is not mathematically oriented

may wish to skip parts of Chapter 3. However, to obtain a full understanding of Chapter 4, and to a lesser extent later chapters, the reader is advised to use a digital computer for which interactive access is available. A very simple program will permit the reader to observe for himself the behavioral consequences of the various transitions which make up the transition function of the 8-state cellular space described in that chapter.

I wish to express my gratitude to Professor Arthur W. Burks of the University of Michigan for introducing me to von Neumann's work in cellular automata; to Professors J. H. Holland, B. A. Galler, and N. R. Scott of the same university for their keen interest in this work, to Dr. Stephen Hedetneimi and Mr. Richard Laing for technical assistance (especially checking); to Mrs. Ann Jacobs for editorial assistance; to Mrs. M. Elizabeth Brandt for drawing the figures; to Professor Ward Edwards for permission to make on-line use of a general-purpose digital computer which is entrusted to his care in connection with Air Force project AF19(628)-2823. The author is indebted to the IBM Corporation for providing the principal support for this work and to the National Institutes of Health (Grant No. GM-12236-01) for providing additional assistance.

This research was conducted in 1964–1965 while the author was participating in the activities of the Logic of Computers Group at the University of Michigan. Lectures on this work were delivered in 1966 at the University of Michigan, Syracuse University, and Uppsala University in Sweden.

San Jose, California
July 1968

E. F. CODD

CONTENTS

Chapter 5. **Components**

Chapter 6. **A Self-Reproducing Universal Computer-Constructor**

Chapter 1

Introduction

1.1 Nature of Investigation

The subject of cellular automata deals with large collections (usually infinite in order to avoid boundary problems) of interconnected finite automata, each finite automaton being thought of as a cell. The kinds of things investigated include conditions under which it is possible to exhibit

1. The computation of all computable functions.
2. The construction of automata by automata, the offspring being at least as powerful (in some well-defined sense) as the parent.

What relevance do these studies have for the development of computers in the future?

Much of our present computer design is aimed at homogeneity of components to facilitate mass production, and at incrementable power to satisfy a great variety of rapidly changing needs. Two levels at which these objectives are apparent are the circuit level and the processing unit level. The user who wishes to boost the internal computing power of his system usually replaces the whole system or a major processing unit. For a small (but increasing) number of system types, he has the option of adding a processing unit. In any event, there is very little choice as to the quality or quantity of the increment in power. Moreover, the user has virtually no means of reorganizing the additional components in a problem-dependent way so as to obtain better performance.

If an economical realization can be found, cellular automata provide the capability of extending the computing power of a system in small or large increments and of reorganizing these increments to suit various special needs.

The study of cellular automata may also prove useful in biochemical research by providing conditions and properties which will aid in the search for macromolecules or microorganisms exploitable in biochemical computers.

The foundation upon which this work is based is John von Neumann's *The Theory of Automata: Construction, Reproduction, Homogeneity*, edited by Arthur W. Burks [6]. In this treatise von Neumann introduces the notion of cellular space and discusses universal computing and constructing automata embedded in such a space. His discussion centers upon a *particular* cellular space having the following characteristics:

1. An infinite plane is divided up into squares.
2. Each square contains a copy of the same finite automaton (the square together with this automaton is called a *cell*).
3. Associated with each cell is its neighborhood, consisting of itself together with its four immediate, nondiagonal neighbors.
4. The state of a cell at time $t + 1$ is uniquely determined by its *neighborhood state* at time t, together with the transition function f of the finite automaton which is associated with every cell.
5. The finite automaton associated with each cell possesses a distinguished state v_0 called the quiescent state, such that

$$f(v_0, v_0, \ldots, v_0) = v_0.$$

6. At each time step all but a finite number of cells are in the quiescent state.
7. The number of distinct states for the finite automaton associated with each cell is 29.
8. A particular transition function f is specified and shown to yield certain computation and construction properties discussed below.

An initial state is specified for every cell in the space. This initial configuration (or pattern of states) together with the local properties of the space listed above uniquely determines the configuration of the

cellular space at each subsequent time step. Configurations with computing ability can be interpreted as automata embedded in the cellular space. The questions posed by (and answered affirmatively by) von Neumann were:

1. Can a universal Turing machine be embedded in the space?
2. Is it possible to embed in the space an automaton A with the property that, given the specifications of any constructible automaton B (in embedable form), A can build B and then set B free to work independently of A?
3. Can an automaton A be exhibited which satisfies the second requirement above and is itself one of the automata which it can construct?

One of von Neumann's interests in studying these questions was in the development of organization in biological systems under the adverse conditions of homogeneity and isotropy.

Thatcher contributed some useful definitions and formalizations in the first chapter of his paper "Universality in the von Neumann Cellular Model" [5]. Many of our definitions (see Chapter 2) are based on his. The major part of his work dealt specifically with the von Neumann 29-state cellular space and provided a new set of constructions to demonstrate the computation and construction universality of that space.

Lee also gave a demonstration of the computation universality of von Neumann's space in a paper entitled "Synthesis of a Cellular Universal Machine using the 29-State Model of von Neumann" [3]. However, his approach departed from the assumptions of von Neumann in regard to cellular spaces (and of Turing in regard to Turing machines) by resorting to configurations in which an infinite number of cells are in the nonquiescent (nonblank) state.

In a paper entitled "Variants of Thatcher's Algorithm for Constructing Pulsers" [1], Hedetniemi studied some design aspects of a class of components introduced by von Neumann in connection with his 29-state space.

Upon surveying this research, virtually all of which is concerned with von Neumann's 29-state cellular space, it appeared timely to ask if similar kinds of behavior could be exhibited in other cellular spaces:

for example, spaces with considerably fewer states per cell. We rejected at the outset an approach in which von Neumann's 29-state space is taken as the starting point and an attempt is made to eliminate states one by one.

Instead, we started with the most primitive class of spaces which have the same neighborhood as von Neumann's—spaces with just two states per cell. Inspection of the pattern propagation properties of these spaces led directly to a proof of their nonuniversality with respect to computation in the von Neumann sense (see Chapter 3).

Two directions of further investigation were immediately apparent:

1. Enlarging the neighborhood.
2. Assuming more than two states per cell.

Successive attempts at finding computation-universal 2-state, 9-neighbor and 3-state, 9-neighbor spaces failed. These attempts were made with the aid of a general-purpose, digital computer and entailed a high degree of interaction between the computer on the one hand and the experimenter on the other. This interaction rapidly developed an intuitive feeling in the experimenter for the kinds of local behavior which would be compatible with one another and, when combined appropriately, would yield the very complex behavior associated with universal computation and construction.

This experimentation also led to a change in tactics. A shift was made to investigating state-rich, neighborhood-poor spaces with the intention of returning later (via simulation) to the state-poor, neighborhood-rich spaces. After some initial reverses, this new approach proved successful.

An 8-state, 5-neighbor space was discovered which is capable of supporting not only the computation and construction behavior sought by von Neumann, but also certain reading and copying behaviors which probably cannot be exhibited in his space. A self-reproducing, programmable computer was designed to exhibit these behaviors. The principles of design, the components, layout, and metalanguages in which the computer is defined are all in sharp contrast to von Neumann's.

Finally, a technique was discovered for defining a 2-state cellular space capable of simulating an arbitrary 8-state space with the von Neumann neighborhood. Coupling this with the demonstrated uni-

versality of an 8-state space proved the existence of a computation construction-universal 2-state cellular space.

In passing, let us note a relationship between the iterative circuit computers described by Holland [2] and the cellular automata introduced by von Neumann. The von Neumann cells in a given cellular space all possess the same neighborhood, which does not vary with time. On the other hand, cells in Holland's iterative circuit computers have neighborhoods which can vary from cell to cell and from time to time. In fact, these neighborhoods can become arbitrarily large. Further, the Holland cell necessarily possesses a large number of states, because it is not far short of being a universal computer itself.

1.2 Outline

We begin with formal definitions of cellular space and associated notions in Chapter 2. Here we introduce definitions for computation universality, construction universality, self-reproduction, and various kinds of symmetries. In Chapter 3 the notions of bounded and boundable propagation are defined and related to universality. Several propositions are stated and proved. The chapter concludes with a proof of the existence of a computation-construction-universal, 2-state cellular space. Included in this proof is a description of a technique for simulating one cellular space by another.

Chapters 4, 5, and 6 are all concerned with showing the existence of a computation-construction-universal, 8-state, 5-neighbor cellular space. The space is defined in Chapter 4, and related to behavioral subgoals in such a way that the reader should be able to gain an appreciation for the method of discovery of this space.

As a stepping stone to the design of a universal computer-constructor (UCC), Chapter 5 introduces in a schematic manner classes of configurations which are later used as components in designing such a UCC. The actual design given in Chapter 6 is expressed partly in terms of these components, partly in block diagram form, and partly in terms of a microprogram. Of particular interest is the set of commands which this computer-constructor can execute—a set which is

universal with respect to computation and construction, but which does not depend upon the space having a distinguished direction.

Chapter 7 describes how discovery of cellular spaces with various properties may be expedited by on-line interaction with a general-purpose digital computer. It further describes some of the tests made on the cellular space defined in Chapter 4 to check that it does behave as claimed.

The entire work is summarized in Chapter 8 and some open questions are listed.

Chapter 2

Basic Definitions

2.1 Introductory Remark

In this chapter our aim is to provide precise definitions for the notion of cellular space, various symmetries which such a space may possess, and various behaviors which it may or may not be able to support. The principal behaviors to be discussed are computation, construction, and self-reproduction.

Ultimately, we wish to know necessary and sufficient conditions for a cellular space to be capable of supporting these behaviors. At the present time we appear to be far from realizing this goal, but it is hoped that the propositions presented in Chapter 3 will provide a step along the way.

Many of the definitions given in this chapter are based heavily upon those proposed by Thatcher [5]. Some important modifications have, however, been made, and these will be pointed out when we come to them. The remaining definitions concerning symmetries and certain compound types of universality are new.

2.2 Cellular Space

Let I denote the set of integers. To obtain a cellular space we associate with the set $I \times I$:

1. The *neighborhood function* g: $I \times I \to 2^{I \times I}$, defined by
$$g(\alpha) = \{\alpha + \delta_1, \alpha + \delta_2, \ldots, \alpha + \delta_n\} \qquad \text{for all} \quad \alpha \in I \times I$$
where $\delta_i (i = 1, 2, \ldots, n) \in I \times I$ is fixed.
2. The finite automaton (V, v_0, f), where V is the set of *cellular states*, v_0 is a distinguished element of V called the *quiescent state*, and f is the local transition function from n-tuples of elements of V into V. The function f is subject to the restriction,
$$f(v_0, v_0, \ldots, v_0) = v_0.$$

It is convenient to think of the cellular space as a plane assemblage of a countable number of interconnected *cells*. The location of each cell is specifiable by its Cartesian coordinates with respect to some arbitrarily chosen origin and set of axes. Each cell contains an identical copy of the finite automaton (V, v_0, f), and the state $v^t(\alpha)$ of a cell α at time t is precisely the state of its associated automaton at time t. Each cell α is connected to the n neighboring cells $\alpha + \delta_1, \alpha + \delta_2, \ldots, a + \delta_n$. In all that follows we shall assume that one of the neighbors of α is α itself and, in line with this assumption, we adopt the convention that $\delta_1 = 0$.

The *neighborhood state function* h^t: $I \times I \to V^n$ is defined by
$$h^t(\alpha) = (v^t(\alpha), v^t(\alpha + \delta_2), \ldots, v^t(\alpha + \delta_n)).$$
Now we can relate the neighborhood state of a cell α at time t to the cellular state of that cell at time $t + 1$ by
$$f(h^t(\alpha)) = v^{t+1}(\alpha).$$
In Chapters 3 and 4 we shall make use of the term *transition*, which is an ordered pair $(h^t(\alpha), v^{t+1}(\alpha))$. Whenever convenient, we omit the time superscript t from h^t.

2.3 Configurations

Following von Neumann we restrict our attention to the case in which all cells except a finite number are initially in the quiescent state. The restriction on f,
$$f(v_0, v_0, \ldots, v_0) = v_0$$

means that a cell whose neighborhood is entirely quiescent remains quiescent itself. (Remember that the neighborhood of any cell includes that cell itself.) Thus, at every time step all cells except a finite number are in the quiescent state.

An allowable assignment of states to all cells in the space is called a *configuration*. Thus, a configuration is a function c from $I \times I$ into V, such that

$$\{\alpha \in I \times I \mid c(\alpha) \neq v_0\}$$

is finite. Such a function is said to have *finite support relative to* v_0, and the set above is denoted by $\sup(c)$.

We are now in a position to define the *global transition function* F. Let C be the class of all configurations for a given cellular space. Then F is a function from C into C defined by

$$F(c)(\alpha) = f(\dot{h}(\alpha)) \qquad \text{for all} \quad \alpha \in I \times I.$$

Given any initial configuration c_0, the function F determines a sequence of configurations[1]

$$c_0, c_1, \ldots, c_t, \ldots$$

where

$$c_{t+1} = F(c_t) \qquad \text{for all } t.$$

We now wish to distinguish between configurations which change with time and those which do not. A configuration c' is a *subconfiguration* of c if

$$c \mid \sup(c') = c' \mid \sup(c').$$

A configuration c is called *passive* if $F(c) = c$, and *completely passive* if every subconfiguration of c is passive. If W is the set of states which occur in the configuration c, we say that W is the *alphabet* of c and c is a *configuration over* W. A subset W of the set V of cellular states is called a *passive* or *completely passive* set if all configurations over W are passive or completely passive respectively.

In the definitions which follow we attempt to make precise what it means for configurations to interact with one another. We begin with two definitions concerned with putting configurations together in a

[1] We shall have occasion in Chapter 3 to call such a sequence a propagation.

space. The configurations c, c' are disjoint if $\sup(c) \cap \sup(c') = \varnothing$. If c, c' are disjoint configurations, their *union* is defined by

$$\begin{aligned}(c \cup c')(\alpha) &= c(\alpha) && \text{if} \quad \alpha \in \sup(c)\\ &= c'(\alpha) && \text{if} \quad \alpha \in \sup(c')\\ &= v_0 && \text{otherwise.}\end{aligned}$$

Let us denote the result of t applications of the global transition function F to configuration c by $F^t(c)$. Let c, c' be disjoint configurations. We say that c *passes information* to c' if there exists a time t, such that

$$F^t(c \cup c')|Q \neq F^t(c')|Q$$

where

$$Q = \sup(F^t(c')).$$

Our introduction of this notion and its use in a number of subsequent definitions constitute an initial point of departure from the definitions of Thatcher.

Occasionally it is convenient to refer to the restriction of a configuration as though it were a configuration. Thus, the configuration $c|Q$ means the configuration which is everywhere quiescent outside Q and identical to $c|Q$ throughout Q. Similarly, if c is a subconfiguration of d, the configuration $d - c$ means the configuration e, such that $c \cup e = d$. The symbol $\overline{\sup}(c)$ means the complement with respect to the entire space of $\sup(c)$.

Configuration s is a *translation* of configuration s' if there exists an element $\delta \in I \times I$, such that

$$S'(\alpha) = S(\alpha - \delta)$$

for all $a \in I \times I$, where the operation is component-wise subtraction.

2.4 Computation

A reasonable way of studying computation in cellular spaces is to set up a correspondence between Turing machines on the one hand and cellular configurations on the other. Such a correspondence should

preserve the distinction between tape and control unit, since this is so vital for the concept of a universal Turing machine.

Because we are dealing with 2-dimensional cellular spaces, it seems appropriate to assume that the Turing machines, with which a correspondence is to be set up, can handle 2-dimensional tapes. Let T be a set of (2-dimensional) tapes each having only a finite number of nonblank symbols. Let W be the set of states which occur in T. A partial function ψ from T into T is *Turing-computable* if there exists a Turing machine with symbol alphabet W which computes ψ.

In a cellular space a 2-dimensional tape is represented by a completely passive configuration, and the blank symbol by the quiescent state. We are now in a position to say what it means for ψ to be computable in a cellular space Z.

First we require that all tapes in T over the alphabet W be completely passive configurations in Z, whether taken individually or collectively. This means that, if we take any finite subset of tapes in T, then, for every set of translations which yields disjoint translates, the union of these translates is completely passive.[1] The function ψ is *computable in* Z if there exists a configuration c, a cell $\alpha \in \sup(c)$, and a nonquiescent state v (which we call the stop state), such that, for any configuration $d \in T$, $\psi(d)$ is defined if there exists a time t such that

$$F^t(c \cup d) \mid \sup(T) =\cdot \psi(d)$$

where

$$\sup(T) = \bigcup_{d \in T} \sup(d)$$

and $F^t(c \cup d) \mid \overline{\sup}(T)$ does not pass information to $F^t(c \cup d) \mid \sup(T)$ and

$$F^t(c \cup d)(\alpha) = v$$

and

$$F^{t'}(c \cup d)(\alpha) \neq v \qquad \text{for all} \quad t' < t.$$

In such a case we say that c *computes* ψ.

Note that this definition does not require the computer c ultimately to become passive. It does, however, require a prespecified cell α to enter a prespecified state v in order to provide an effective method of determining when a computation is completed.

[1] Note that the union of two or more completely passive configurations is *not necessarily* completely passive.

Let us call an infinite set of tapes, which is in effective one-to-one correspondence with the set of nonnegative integers, a *Turing domain.* As previously, we require these tapes to be completely passive whether taken individually or collectively. A cellular space is *computation-universal* if there exists a Turing domain T and if, for any Turing-computable partial function ψ from T into T, there exists a configuration c disjoint from T such that c computes ψ. In short, a space Z is computation-universal if (in an appropriate sense) every Turing-computable partial function is computable in Z.

Let Z be a cellular space which has a Turing domain T. Suppose there exists a configuration c disjoint from T such that, for any Turing-computable partial function ψ from T into T, there exist a tape $d \in T$ and a translation δ such that d_δ is disjoint from T and from c, and furthermore $c \cup d_\delta$ computes ψ. Then c is called a *universal computer* with domain T.

Clearly, the existence of a universal computer in a cellular space Z implies the computation universality of Z. The converse appears to be an open question. In Chapter 6 we demonstrate the computation universality of a certain 8-state cellular space by exhibiting a universal computer in that space.

2.5 Construction

In regard to construction in cellular spaces, our primary concern is with the generation of configurations which, in the sense defined in the last section, possess the ability to compute. Our interest in the generation of completely passive configurations is incidental, being confined to their use as tapes for configurations which compute and as tapes for configurations which construct.

This was the kind of emphasis which von Neumann placed on construction when he asked: "Can an automaton [*sic*] be constructed . . . by another automaton?" After all, the generation of completely passive configurations, taken by itself, can be handled by the notions of computation just introduced.

We begin with the following broad definition. The configuration c *constructs* the configuration c' if there exists a time t, such that

1. c' is a subconfiguration of $F^t(c)$ disjoint from c.
2. $F^t(c) - c'$ does not pass information to c'.

Note that the requirement that c' be disjoint from c eliminates the trivial case of a completely passive configuration "constructing" itself at every time step and also the case in which the only activity is decay (i.e., conversion of nonquiescent cells to the quiescent state). The second requirement provides that c' shall be the *end result* of construction—any further change which takes place in c' would take place if c' were alone in the space. Simple forms of pattern propagation are not eliminated by this definition.

The notion of construction universality which we are about to formalize demands of a space the existence of configurations with the ability to construct a rich enough set of computers such that with this set any Turing-computable partial function on a Turing domain can be computed in the space. We shall not require that *all* configurations, which can be interpreted as having computing ability, be constructible. That would be a tall order.

Suppose Z is a computation-universal cellular space and T is a Turing domain over which this universality can be exhibited. Let C be a class of configurations, each of which computes some Turing-computable partial function on T, and for any such function, at least one member of C computes it. Then, C is called a *complete set of computers* with domain T.

Note that a complete set of computers is always an infinite set. In the case of a universal computer, we must take it infinitely many times with as many different tapes as are necessary for the computation of all the Turing-computable partial functions on the pertinent domain. Given a computation-universal cellular space having a complete set C of computers with domain T, such that every member of C is constructible by some configuration disjoint from C, then the space is said to be *construction-universal.* Of particular interest is the case in which a *universal constructor* exists—i.e., a configuration which, when augmented by a suitable tape (disjoint from C), constructs an arbitrary member of C.

A further special case of interest is that in which both a universal computer and a universal constructor exist and the set of all tapes required by the universal constructor is included in the Turing domain T. For in this case it is possible to present in coded form the specifications

of configurations to be constructed and have the universal computer decode these specifications (providing that the decoding transformation is a Turing-computable one). Then the universal constructor can implement the decoded specifications.

A single configuration which is both a universal computer with domain T and a universal constructor with domain T' is called a *universal computer-constructor* if $T' \subseteq T$. A space in which such a configuration exists is said to be *computation-construction universal.*

The following additional notions related to construction can be readily formalized, but are merely mentioned in passing due to the marginal relevance to the propositions of Chapter 3. They are, however, relevant to the universal computer-constructor exhibited in Chapter 6.

A desirable property of a constructor is the ability, when supplied with a suitable tape, to locate the configurations it constructs almost anywhere in the space, provided only that the site is quiescent and accessible. Note that the universal computer-constructor described in Chapter 6 possesses this property. However, a simulated version in 2-state space (see Section 3.3) does not have the same degree of arbitrariness in location, because of the many-to-one correspondence between cells in the 2-state space and cells in the 8-state space.

Another desirable property of a constructor is the ability to preserve its constructing power (i.e., the set of configurations which it can construct) whenever it effects a construction. A further question which may arise is whether, and under what circumstances, these various properties are inherited by the configurations which are constructed.

2.6 Self-Reproduction

The natural way to define self-reproduction in cellular spaces is as a special case of construction. Thus, the configuration c is *self-reproducing* if there exists a translation δ such that c constructs c_δ. Again, we may specialize the various forms of construction—the arbitrary location type, the type in which the constructor preserves its ability to construct, and so on—to obtain the corresponding types of self-reproduction.

Casual reading of the definition of computer-constructor may give one the impression that there necessarily exists a tape which makes it

self-reproducing. In fact, this need not be so, because the complete set of computers which it can construct need not be composed of that universal computer coupled with appropriate tapes. The universal computer-constructor described in Chapter 6 happens to be self-reproducing with the arbitrary location property. It also has the ability to retain its self-reproducing property after any number of self-reproductions.

2.7 Symmetries of Cellular Spaces

We treat symmetries of cellular spaces in two categories: those pertaining to the neighborhood function g and those to the transition function f. We begin with g.

Let $\delta = (x, y) \in I \times I$. Then, we adopt the following notation:

$$\delta' = (-y, x)$$
$$\delta'' = (-x, -y).$$

A weak form of symmetry which g may possess is expressed by

$$\alpha \in g(\beta) \leftrightarrow \beta \in g(\alpha).$$

This is equivalent to

$$\alpha + \delta \in g(\alpha) \leftrightarrow \alpha + \delta'' \in g(\alpha),$$

and when this holds we shall say g is *180°-rotation-symmetric*. Since we are interested in isotropic cellular spaces, we need the stronger property,

$$\alpha + \delta \in g(\alpha) \leftrightarrow \alpha + \delta' \in g(\alpha),$$

and any g satisfying this condition is called *90°-rotation-symmetric*.

In addition to the rotation symmetries defined above we can similarly define reflection symmetries with respect to the two coordinate axes. Note that the von Neumann neighborhood

$$g_1(\alpha) = \{\alpha, \alpha + (1, 0), \alpha + (0, 1), \alpha + (-1, 0), \alpha + (0, -1)\}$$

is reflection-symmetric with respect to both axes, as well as being rotation-symmetric.

Assume g is 90°-rotation-symmetric. The local transition function f is *rotation-symmetric* if there exists a permutation p acting on the set V of cellular states such that

1. $p(v_0) = v_0$
2. if $f(v^t(\alpha), v^t(\alpha + \delta_2), \ldots, v^t(\alpha + \delta_n)) = v^{t+1}(\alpha)$ then

$$f(p(v^t(\alpha)), p(v^t(\alpha + \delta_2')), \ldots, p(v^t(\alpha + \delta_n')) = p(v^{t+1}(\alpha)).$$

In the special case of p being the identity function, we say f is *strongly rotation-symmetric*. For convenience we also apply the terms rotation-symmetric and strongly rotation-symmetric to cellular spaces which have transition functions with the corresponding symmetries.

In passing we note that the von Neumann 29-state, 5-neighbor space is rotation-symmetric, but not strongly so. This is curious, since von Neumann placed considerable emphasis on isotropy.

In a similar way we can define reflection symmetries for the transition function. These appear to be less useful than rotation symmetry. In fact, we shall exploit reflection *a*symmetry in developing a universal 8-state space in Chapter 4.

Chapter 3

Propagation and Universality

3.1 Preliminary Definitions

A natural metric to associate with any cellular space based on $I \times I$ is the so-called city-block metric ρ, defined by

$$\rho(\alpha, \beta) = |x_\alpha - x_\beta| + |y_\alpha - y_\beta|$$

where

$$\alpha = (x_\alpha, y_\alpha), \qquad \beta = (x_\beta, y_\beta)$$

The metric ρ is extended to finite sets of cells by

$$\rho(P, Q) = \max_{\alpha \in P, \beta \in Q} \rho(\alpha, \beta)$$

where P, Q are any finite sets of cells. We call $\rho(P, P)$ the diameter of P and abbreviate it to dia P. Note that the extended function is not a true metric, since in general $\rho(P, P) \neq 0$.

The extended function can now be applied to configurations as follows:

$$\rho(c, d) = \rho(\sup(c), \sup(d))$$

and

$$\text{dia } c = \text{dia}(\sup(c))$$

where c, d are any configurations.

Using this quasi-metric we can define various notions related to propagation. Given a cellular space $(I \times I, N, S, s_0, f)$ we have

observed in Chapter 2 that an initial configuration c_0 determines a sequence of configurations,

$$c_0, c_1, c_2, \ldots, c_t, \ldots$$

where for all t

$$c_{t+1} = F(c_t)$$

and F is the global transition function. We shall call such a sequence a *propagation* and denote it by $\langle c_0 \rangle$.

A propagation $\langle c \rangle$ is *bounded* if there exists an integer K such that for all t

$$\rho(c, F^t(c)) < K.$$

Otherwise, $\langle c \rangle$ is an *unbounded propagation.*

Suppose propagation $\langle c \rangle$ is unbounded. It may be possible to check its growth by adjoining to configuration c at time 0 some other (disjoint) configuration. Accordingly, we define a *boundable propagation* $\langle c \rangle$ as one for which there exists a disjoint configuration d such that $c \cup d$ is bounded. In this case we say that *d bounds c.* If no d bounds a given c, $\langle c \rangle$ is said to be an *unboundable propagation.*

As a preliminary step to discussing propagation in 2-state and 3-state cellular spaces, some remarks about neighborhoods are in order. Given a cellular space with origin θ and neighborhood function g such that

$$g(\alpha) = \{\alpha, \alpha + \delta_2, \ldots, \alpha + \delta_n\}$$

for all $\alpha \in I \times I$, let us treat $g(\theta)$ as the *neighborhood template* which can be translated to any position $\alpha \in I \times I$ to define the neighborhood at that position.

We can associate with the neighborhood template a minimal circumscribing rectangle R_g with the following properties:

1. Its edges are parallel to the coordinate axes.
2. All cells belonging to $g(\theta)$ lie within R_g.
3. Touching each edge of R_g there is at least one cell belonging to $g(\theta)$.

Identifying the positive y axis with the direction north, we call the set of cells in R_g with maximum abscissas the eastern perimeter of R_g. The northern, western, and southern perimeters are similarly defined (see

Fig. 3.1). The following cells are called extremal elements of the neighborhood template:

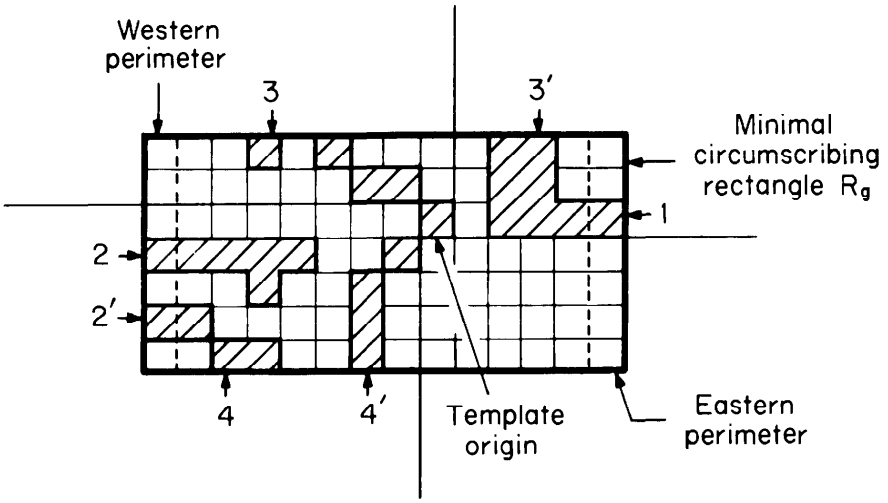


FIG. 3.1. Extremal elements of neighborhood template.

1. The northernmost and southernmost template cells in the eastern perimeter if this perimeter is strictly east of the template origin (cell 1 in Fig. 3.1).
2. The northernmost and southernmost template cells in the western perimeter if this perimeter is strictly west of the template origin (cells 2, 2′).
3. The westernmost and easternmost template cells in the northern perimeter if this perimeter is strictly north of the template origin (cells 3, 3′).
4. The westernmost and easternmost template cells in the southern perimeter if this perimeter is strictly south of the template origin (cells 4, 4′).

As many as eight cells, or as few as one (if we exclude the trivial neighborhood), are identified in this way.

3.2 Propagation in Certain 2-State Cellular Spaces

In the cellular spaces discussed below we assume that the quiescent state is labeled 0, and the remaining states are 1, 2, . . . as needed.

State v_1 always refers to the state of the template origin (the "center" cell of the neighborhood).

PROPOSITION 1. In any 2-state cellular space with local transition function f, if there exists an integer $j \neq 1$ such that

$$f(v_1, v_2, \ldots, v_n) = 1$$

and

$$v_i = 0 \text{ for } i \neq j$$
$$1 \text{ for } i = j$$

and element j of the neighborhood template is extremal, then every configuration in that space yields an unboundable propagation.

Proof. Suppose the hypothesis is satisfied by an element j with template coordinates (x_1, y_1) and, let us say that it is the northernmost in the eastern perimeter of the minimal rectangle for the template. The argument is similar if it happens to be any other extremal element of the template.

Let c be any configuration in the space at time t and R_c its minimal circumscribing rectangle: i.e., $\sup(c) \subseteq R_c$ and, touching each edge of R_c there is at least one cell in state 1. We identify extremal cells for $\sup(c)$ in R_c just as we did for $g(\theta)$ in R_g, except that we drop the restrictions related to the origin. Select that extremal cell of $\sup(c)$ which is the southernmost in the western perimeter of R_c and let its coordinates be (x, y)—see Fig. 3.2.

Then, the cell $\alpha = (x-x_1, y-y_1)$ has the neighborhood state,

$$h(\alpha) = (v_1, v_2, \ldots, v_n)$$

where $v_i = 0$ for all $i \neq j$, $v_j = 1$, and the element j is the extremal one for which

$$f(v_1, v_2, \ldots, v_n) = 1$$

Thus, at time $t + 1$ cell α is in state 1. Now x_1 is strictly positive, because j is extremal in the eastern perimeter of R_g. Hence, the western perimeter of the configuration at time $t + 1$ is strictly to the west of the western perimeter of the configuration at time t. Since this applies at all times t,

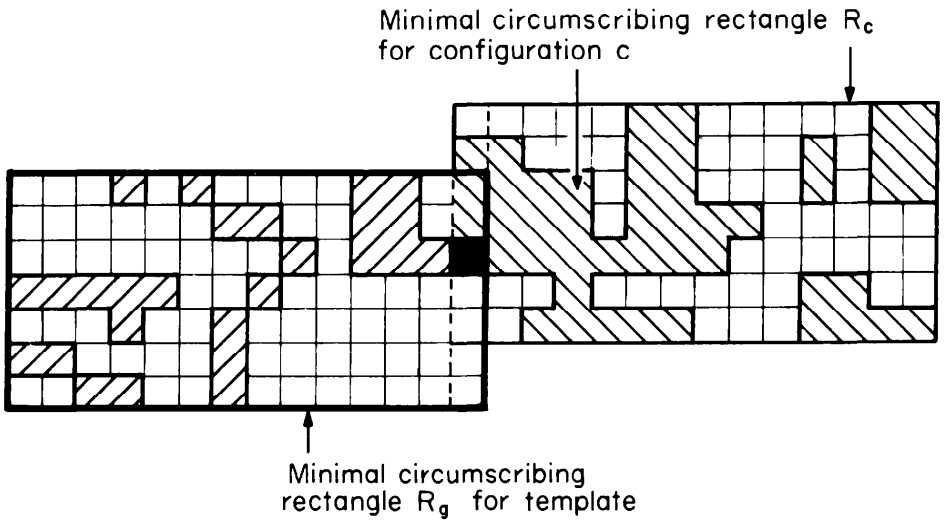


FIG. 3.2. Matching extremals of template and configuration.

the propagation $\langle c \rangle$ is unbounded. Finally, we observe that c was an arbitrary configuration. Thus, for all c, $\langle c \rangle$ is unbounded and therefore, for all c and all d, $\langle c \cup d \rangle$ is unbounded; whence, for all c, $\langle c \rangle$ is unboundable.

PROPOSITION 2. In every 2-state, 5-neighbor cellular space with 90°-rotation-symmetric neighborhood either every configuration yields an unboundable propagation or every configuration yields a bounded propagation. In the latter case, the diameter of any initial configuration cannot expand by more than a factor of two.

REMARK. According to this proposition, unbounded but boundable propagation cannot be exhibited in this class of cellular spaces.

Proof. Consider any configuration c. The neighborhood template has four elements excluding the template origin. Because the neighborhood is 90°-rotation-symmetric, either all four lie on the template axes or all four do not.

Case 1: all four noncenter neighborhood elements lie on the axes. Any cell α outside R_c has at most one neighbor belonging to sup(c). If no neighbor belongs to sup(c), the cell α must remain quiescent due to the

requirement for all transition functions that $f(0, 0, \ldots, 0) = 0$. If one neighbor of α belongs to sup(c), the neighborhood state is given by

$$h(\alpha) = (0, v_2, v_3, v_4, v_5)$$

where one and only one of the v_i, say v_j, has the value 1.

Either $f(h(\alpha)) = 1$, in which case (since j is extremal) we can apply Proposition 1 and conclude that every configuration yields an unboundable propagation; or $f(h(\alpha)) = 0$, in which case every cell outside R_c remains quiescent, the propagation $\langle c \rangle$ is contained within R_c for all time and $\langle c \rangle$ is therefore bounded. In the latter case the diameter of the initial configuration is no smaller than the length[1] of the larger edge of R_c and the final diameter is no larger than the sum of the lengths of the edges of R_c. Thus, the diameter expands by no more than a factor of two.

Case 2: all four noncenter neighborhood elements lie off the axes. Any cell α outside R_c has 0, 1, or 2 neighbors belonging to sup(c). However, we can always find a cell α outside R_c with precisely one neighbor belonging to sup(c). We simply match the northernmost template cell in the eastern perimeter of R_g with the southernmost cell of sup(c) in the western perimeter of R_c. Thus, the neighborhood state $h(\alpha)$ as defined above arises once again.

Case 2.1: $f(h(\alpha)) = 1$ yields unboundable propagation as previously.
Case 2.2: $f(h(\alpha)) = 0$.

Those cells outside R_c which have just one neighbor in sup(c) do not contribute to the growth of c beyond R_c. We must therefore look at the only remaining possibility for such growth: namely, those cells outside R which have two neighbors in sup(c). If no such cell exists, then the propagation $\langle c \rangle$ is clearly contained in R_c.

Consider a cell α outside R_c with two neighbors in sup(c)—say β, γ. Treating the three cells as points, we observe that the angle subtended at α by β, γ is 90°, by the assumption that the neighborhood is 90°-rotation-symmetric. Further, if the first-quadrant element of the neighborhood template has coordinates (x, y) one of the lines $\alpha\beta$, $\alpha\gamma$ makes an

[1] The length of an edge of R is taken as the distance from one vertex to the other, and is therefore one less than the number of cells in the corresponding perimeter.

angle arctan(y/x) with the x axis, and the other an angle arctan(x/y) with that axis (neither x nor y is zero, because of the assumption for Case 2).

Thus, the first time step—and all subsequent time steps—cannot result in growth of c outside a rectangle R_c' (see Fig. 3.3), which minimally encloses R_c and whose edges make an angle arctan(y/x) with

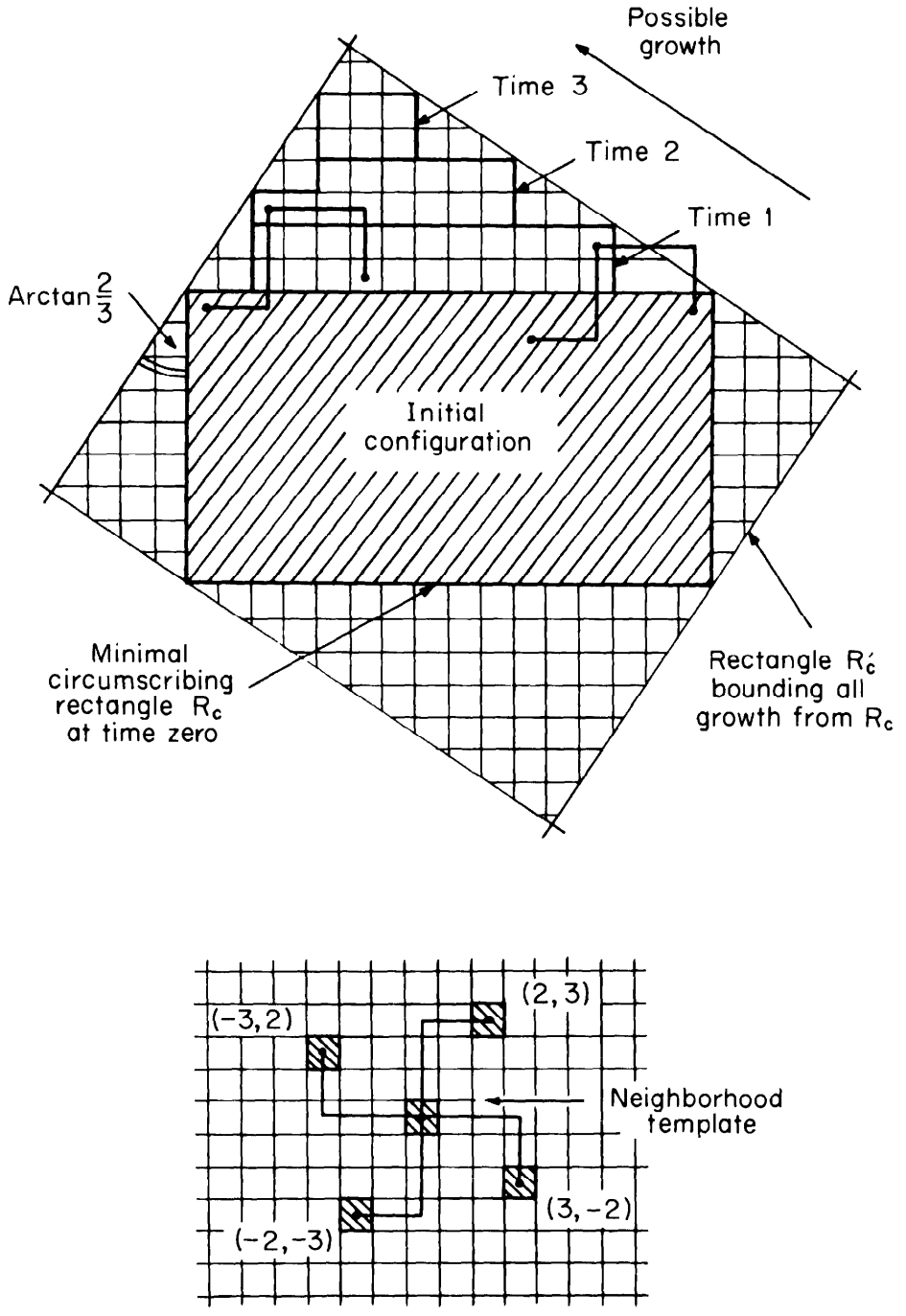


FIG. 3.3. Rectangle bounding growth in Case 2.2.

those of R_c. Hence, $\langle c \rangle$ is bounded. Its initial diameter is no smaller than the length of the larger edge of R_c. Its final diameter is no larger than twice that value.

Since, in each of the above cases, the choice of c was arbitrary, the results apply to all configurations. Note that, when every propagation is unbounded, every propagation is unboundable.

PROPOSITION 3. There exists a strongly rotation-symmetric 2-state, 9-neighbor cellular space in which unbounded but boundable propagation is realizable.

Proof. Consider the neighborhood defined by the following set of template coordinates δ_i $(i = 1, 2, \ldots, 9)$:

$$(0,0), (-1, 1), (0,1), (1, 1), (1, 0), (1, -1), (0, -1), (-1, -1), (-1, 0)$$

We specify the transition function f to the extent desired by stating that it is strongly rotation-symmetric and listing some seven constraints which it must satisfy.

Each constraint is coded to take advantage of the rotation symmetry and accordingly represents four transitions belonging to f. Thus, the leftmost digit of the neighborhood state is the state of the center cell of the neighborhood. The next eight digits are the states of the eight (noncenter) neighbors taken clockwise and so as to yield the smallest eight-digit integer. The constraints on f are as follows:

$$\begin{aligned}
f(0, 0, 0, 0, 0, 0, 0, 1, 1) &= 1 \\
f(0, 0, 0, 0, 0, 0, 1, 1, 0) &= 1 \\
f(1, 0, 0, 0, 0, 1, 1, 1, 0) &= 1 \\
f(1, 0, 1, 0, 1, 0, 1, 0, 1) &= 1 \\
f(0, 0, 0, 0, 0, 0, 0, 0, 1) &= 0 \\
f(0, 0, 0, 0, 0, 0, 0, 1, 0) &= 0 \\
f(0, 0, 0, 0, 0, 0, 1, 1, 1) &= 0.
\end{aligned}$$

The propagation $\langle c \rangle$, where c is defined by

$$\begin{aligned}
c(\alpha) &= 1 \quad \text{if} \quad \alpha = (0, 0), (0, 1) \\
&= 0 \quad \text{otherwise},
\end{aligned}$$

is unbounded. However, c is bounded by the configuration d, where

$$\begin{aligned} d(\alpha) &= 1 \quad \text{if} \quad \alpha = (-1, 0), (0, -1), (1, 0) \\ &= 0 \quad \text{otherwise.} \end{aligned}$$

PROPOSITION 4. There exists a strongly rotation-symmetric 3-state, 5-neighbor cellular space in which unbounded but boundable propagation is realizable.

Proof. We take the von Neumann neighborhood defined by the following set of template coordinates δ_i $(i = 1, 2, \ldots, 5)$:

$$(0, 0), \quad (0, 1), \quad (1, 0), \quad (0, -1), \quad (-1, 0).$$

Once again we shall define a transition function f which is strongly rotation-symmetric:

$$\begin{aligned} f(h(\alpha) &= 2 \quad \text{if} \quad \text{the state of cell } \alpha \text{ is } 2 \\ &= 1 \quad \text{if} \quad h(\alpha) = 00001, 00010, 00100, 01000 \\ &= 0 \quad \text{otherwise.} \end{aligned}$$

The configuration c, consisting of a single 1 at the origin and all other cells quiescent, yields an unbounded propagation. However, any configuration consisting simply of a ring of cells in state 2 which surrounds the origin bounds c. For example, the configuration d bounds c, where

$$\begin{aligned} d(\alpha) &= 2 \quad \text{if} \quad \alpha = (-1, 1), (0, 1), (1, 1), (1, 0), \\ &\qquad\qquad (1, -1), (0, -1), (-1, -1), (-1, 0) \\ &= 0 \quad \text{otherwise.} \end{aligned}$$

3.3 Universality

PROPOSITION 5. Given a cellular space Z, if there exists an integer K such that, for every configuration c,

$$\forall t\,[\rho(c, F^t(c)) < K \operatorname{dia} c],$$

then Z is not computation-universal.

Proof. Suppose the hypothesis is satisfied for some Z, and Z is computation-universal. Then, there exists a Turing domain (say T) in Z. Define a function $E: T \to 2^T$ as follows:

$$E(d) = \{e \in T \mid \rho(d, e) > n_d \rho_\theta(d)\} \qquad \text{for any} \quad d \in T,$$

where n_d is the nonnegative integer (called the index of d) which is associated with d by the effective one-to-one correspondence which underlies the Turing domain T, and $\rho_\theta(d)$ is the quasi-metric distance of d from the origin θ.

The function $\psi: T \to T$ is defined by

$$\psi(d) = d'$$

where d' is that element of $E(d)$ whose index is least. Clearly, ψ is Turing-computable. Since Z is computation-universal, there exists a configuration c disjoint from T which computes ψ. Note that ψ is total.

Choose a tape $d \in T$ such that

$$n_d > \frac{K(\rho_\theta(c) + \rho_\theta(d))}{\rho_\theta(d)}.$$

Such a tape can always be chosen because $\rho_\theta(c)$ is fixed and there are infinitely many tapes for which

$$\rho_\theta(d) > K\rho_\theta(c)$$

Consider the configuration $c \cup d$. Since c computes ψ, there exists a time t such that

$$F^t(c \cup d) \mid \sup(T) = \psi(d) = d'.$$

Now,

$$\begin{aligned} \rho(c \cup d, F^t(c \cup d)) &\geqslant \rho(c \cup d, d') \\ &\geqslant \rho(d, d') \\ &> n_d \rho_\theta(d) && \text{(by definition of } \psi) \\ &> K(\rho_\theta(c) + \rho_\theta(d)) && \text{(by choice of } d) \\ &\geqslant K \operatorname{dia}(c \cup d) \end{aligned}$$

which contradicts the hypothesis.

REMARK. This proposition states that in a computation-universal cellular space it must be possible, by choice of initial configuration, to

obtain propagation as far out from this configuration as desired. It does not state that there must exist a configuration which yields unbounded propagation. We conjecture, however, that the existence of unbounded but boundable propagation is a necessary condition for computation universality.

PROPOSITION 6. There does not exist a computation-universal 2-state, 5-neighbor cellular space with a neighborhood which is 90°-rotation-symmetric.

Proof. By Proposition 2 either
1. Every configuration yields an unboundable propagation, or
2. Every configuration yields a bounded propagation.

Case 1: Every configuration yields an unboundable propagation. Therefore there do not exist any completely passive configurations. Hence, there is no Turing domain in the space. Thus the space is not computation-universal.

Case 2: Every configuration is bounded and its diameter cannot expand by more than a factor of 2. Therefore, by Proposition 6 with $K = 2$, the space is not computation-universal.

PROPOSITION 7. There exists a strongly rotation-symmetric 8-state, 5-neighbor space (with the von Neumann neighborhood) which is computation-construction universal.

Proof. See Chapters 4, 5, 6.

PROPOSITION 8. There exists a 2-state cellular space which is computation-construction universal.

Proof. Our method of proof is as follows. We show that *any* 8-state, 5-neighbor cellular space with the von Neumann neighborhood can be simulated by a suitably chosen 2-state space. Then we appeal to Proposition 7.

Let Z_8 be the 8-state space to be simulated. Our task is to define a 2-state space, say Z_2, which simulates Z_8. Accordingly, we must find a suitable neighborhood function—or equivalently a neighborhood template—and a suitable transition function for Z_2. In selecting the neighborhood template for Z_2 we are more concerned with simplicity of exposition of the simulation principles and ease of generalization than with minimality of the template size. The resulting template is quite large, having 85 elements.

Let us impose on the entire Z_2 plane a uniform grid Z_8^* with a square mesh, each mesh consisting of a 3-by-3 array of Z_2 cells. The set of 3-by-3 arrays in Z_2 defined by this grid is now placed in one-to-one correspondence with the set of cells in Z_8 preserving the neighborhood relation in the obvious way.

Figure 3.4 shows a configuration in Z_8. Figure 3.5 shows how this configuration is coded when it is transformed into Z_2. In both figures a blank square denotes a cell in the quiescent state 0. The state 0 is explicitly indicated when it is one of the three bits used for representing a state of a cell in Z_8.

Denote the image of cell k_8 belonging to Z_8 by k_8^* (an element of the grid Z_8^* on the space Z_2). We need only three of the nine Z_2 cells in k_8^* to represent the state of k_8. A diagonal strip of Z_2 cells in k_8^* is selected

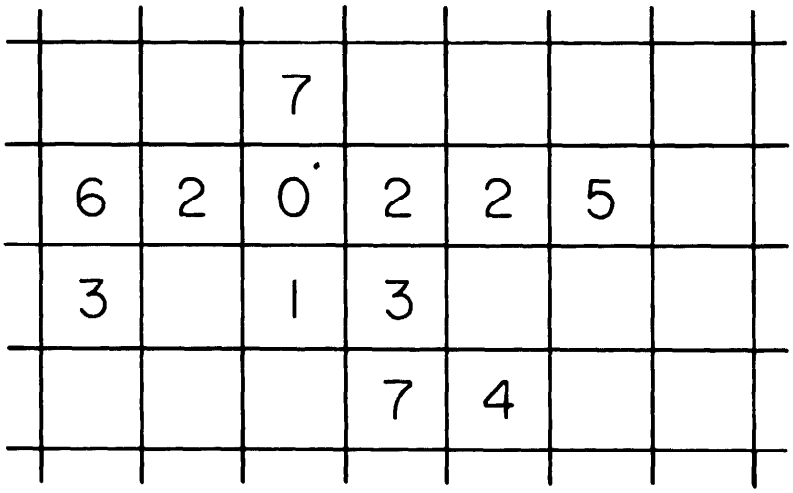


FIG. 3.4. Configuration in Z_8.

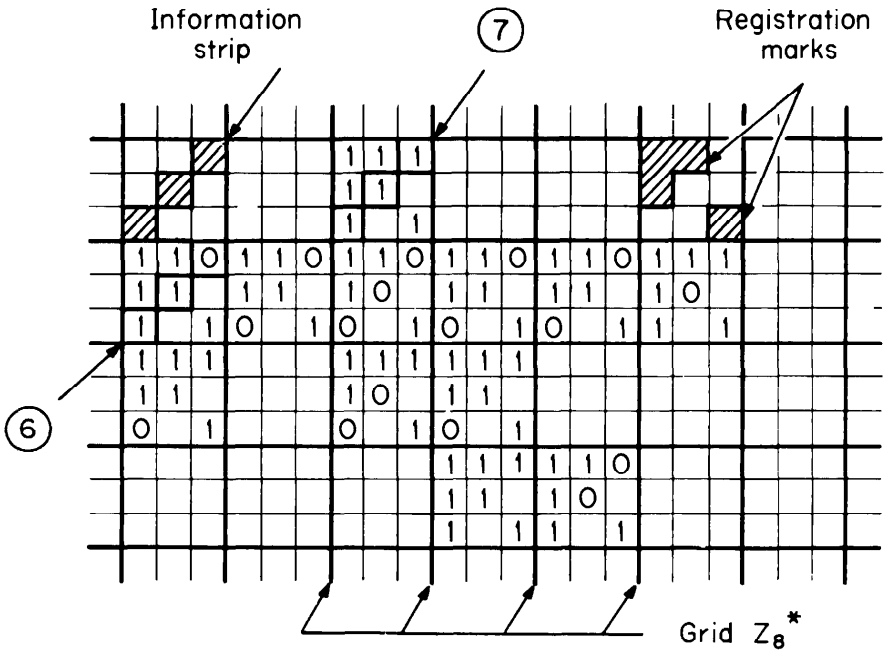


FIG. 3.5. Configuration in Z_2.

for this purpose, and we shall call this the *information strip*. Two of the information strips highlighted in Fig. 3.5 are labeled with the Z_8 states represented in them.

We adopt the binary (i.e., radix 2) code for the eight states of Z_8. The high-order bit is allocated to the bottom cell of the information strip, the next bit to the middle cell, and the low-order bit to the top cell of the strip.

Suppose we consider that the one-to-one transformation which takes Z_8 onto Z_8^* acts also on the template T_8 in space Z_8 to produce the *image template* T_8^*, which is therefore similar in shape and size to the original template. A distinction now arises which is crucial to understanding the method of simulation: the template T_2 to be defined for space Z_2 is not coincident with the image template T_8^*. In fact, T_2 is considerably larger.

Figure 3.6 shows template T_2 enclosed in a heavy black line and template T_8^* cross-hatched. The size and shape of T_2 are such that, when its center C is placed on any one of the nine Z_2 cells of the cell k_8^*, all five neighbors of k_8^* are fully covered by T_2 and no other Z_8^* cells are fully covered (in Fig. 3.6 the center C is placed on the top left cell of k_8^*).

This coverage by T_2 together with a set of *registration marks* (see Fig. 3.5) provides a means of determining which of the nine Z_2 cells in k_8^*, the template T_2 is centered on. Such location sensitivity is necessary

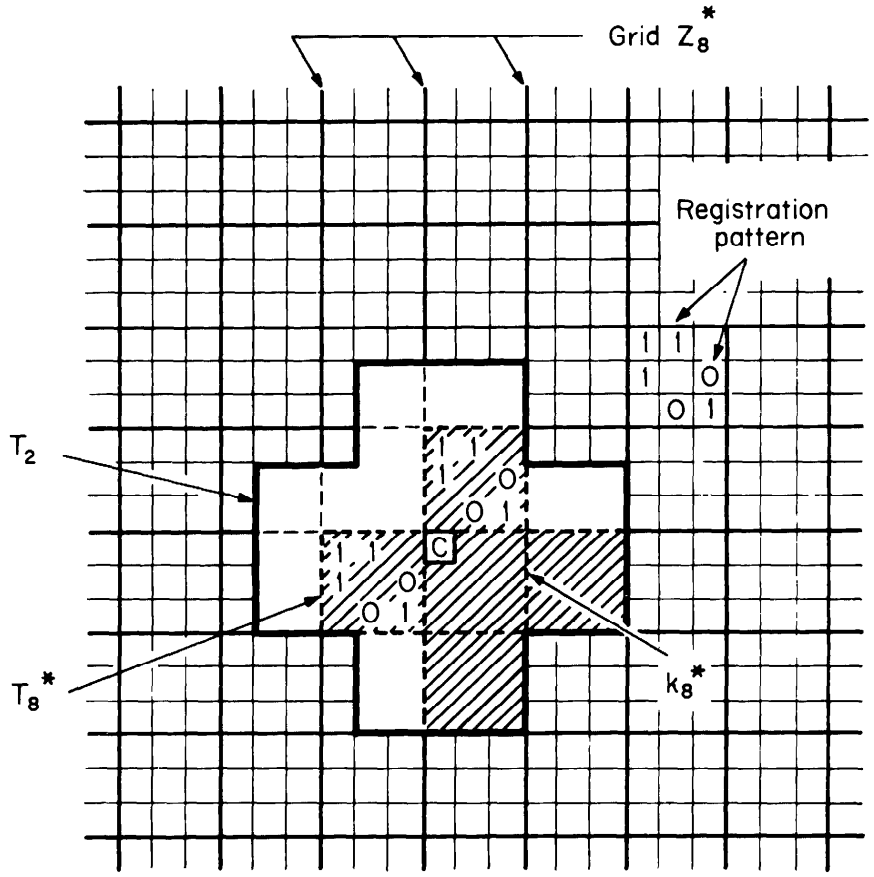


FIG. 3.6. Templates T_2 and T_8^*.

in the first place because the three bits in the information strip of k_8^* depend in three distinct ways on all 15 bits in the five information strips in the T_8^* neighborhood of k_8^*.

Initially, registration marks are placed in those Z_8^* cells which have nonquiescent information strips. If we designate by (i, j) the Z_2 cell in row i, column j, of a given Z_8^* cell, the marks are assigned to cells (1, 1), (1, 2), (2, 1), (3, 3) in any Z_8^* cell which is to be marked.

Cells (2, 3), (3, 2) are permanently quiescent. The three remaining cells (1, 3), (2, 2), (3, 1) constitute the information strip.

As time progresses the configuration in Z_8 may grow (i.e., cells initially quiescent may become nonquiescent). The configuration in Z_2 must grow in a corresponding manner, and as it does, new registration marks must be generated. Remember that we cannot assume every Z_8^* cell is marked initially, since this would violate the assumption underlying our whole work that only a finite number of cells can be in a nonquiescent state.

If we permitted the unconditional generation of new registration marks, there would result permanent activity on the periphery of configurations in Z_2. Now, if the simulation is to preserve computation universality, passive configurations in Z_8 must be represented by passive configurations in Z_2. Thus, we permit the generation of new registration marks in any cell k_8^* *if and only if* its information strip is about to pass from the quiescent state to some nonquiescent one.

On the other hand, for our present purposes we do not need to provide for the decay of registration marks at all, and therefore do not do so. As a result, a Z_8^* cell may be marked, even though its information strip is quiescent. It cannot, however, be unmarked if its information strip is nonquiescent. Note that at all times only a finite number of Z_8^* cells are marked.

Since the simulation is real-time (i.e., one time step in Z_2 corresponds to one in Z_8) the generation of new registration marks must take place *at the same time step* that the information strip passes from the quiescent state to a nonquiescent state. Hence, the new state of each *registration cell* in k_8^* is dependent upon all five information strips in the neighborhood of k_8^*. This is a second reason for T_2 being so large.

Having discussed the principles of simulation, the mapping from initial configurations in Z_8 into configurations in Z_2, and the specification of the neighborhood template T_2, we turn our attention to details of the transition function f_2. We consider f_2 in three parts:

1. Determining template location with respect to the grid Z_8^*.
2. Transforming information strips.
3. Preserving registration marks and, where necessary, generating new ones.

Suppose template T_2 is placed with its center on some cell α in Z_2. Let us use template coordinates to refer to each Z_2 cell sensed by T_2 in this position. Thus α itself is (0, 0), the square directly above it is (0, 1), and so on. Consider a pattern of six cells: (x, y), $(x + 1, y)$, $(x, y - 1)$, $(x + 2, y - 2)$ are all in state 1, while $(x + 2, y - 1)$, $(x + 1, y - 2)$ are in state 0. The sensing of such a pattern somewhere within T_2 implies that (x, y) is the top left Z_2 cell in some Z_8^* cell.

For a given α there are precisely five registration cells (x, y) for which such a pattern can be detected—one for each of the five Z_8^* cells which

are fully covered by T_2. Denote the registration cell of the center Z_8^* cell by (x_α, y_α). Then the five registration cells for the given α are

$$(x_\alpha, y_\alpha), \quad (x_\alpha, y_\alpha + 3) \quad (x_\alpha + 3, y_\alpha), \quad (x_\alpha, y_\alpha - 3), \quad (x_\alpha - 3, y_\alpha).$$

Write $s(x, y)$ for the state of the Z_2 cell with template coordinates x, y. Note that $s(x, y)$ has only two possible values. Write also:

$$P(x, y) = s(x, y) \wedge s(x + 1, y) \wedge s(x, y - 1) \wedge s(x + 2, y - 2) \wedge \neg s(x + 2, y - 1) \wedge \neg s(x + 1, y - 2)$$

$$Q(x, y) = P(x, y) \vee P(x, y + 3) \vee P(x + 3, y) \vee P(x, y - 3) \vee P(x - 3, y)$$

Then, a registration pattern is sensed if and only if there exists a cell (x, y) for which

$$Q(x, y) = 1.$$

In fact, only nine cells (x_α, y_α) need be considered—one for each of the nine possible template positions α of the template center within a Z_8^* cell. These nine cells are

$$\begin{array}{ccc} (0, 0), & (0, -1), & (0, -2) \\ (-1, 0), & (-1, -1), & (-1, -2) \\ (-2, 0), & (-2, -1), & (-2, -2). \end{array}$$

Thus, if $Q(i, j) = 1$, the template T_2 is centered on the Z_2 cell in row $1 - i$, column $1 - j$ of the central Z_8^* cell covered by T_2. If all nine $Q(i, j)$ are zero, the Z_8^* neighborhood is quiescent and hence the central Z_8^* cell remains quiescent.

Whenever the location-sensing test defined above indicates that T_2 is centered on a Z_2 cell belonging to an information strip, the transformation to be applied is directly related to the transition function f_8 in Z_8 as follows. For any state v in Z_8, let $p(v)$, $q(v)$, $r(v)$ denote the high-order, middle-order, and low-order bits of the binary code for v. Suppose that the neighborhood state of cell β in Z_8 is given by

$$h_8(\beta) = (v_1, v_2, v_3, v_4, v_5)$$

and the transition function f_8 satisfies $f_8(h_8(\beta)) = v'$.

Further, suppose template T_2 is centered on the top bit of the information strip in cell β^* (the image of β in the grid Z_8^*). Then, we have (at time t, say)

$$\begin{aligned} s(-2, -2) &= p(v_1) \\ s(-1, -1) &= q(v_1) \\ s(0, 0) &= r(v_1) \end{aligned}$$

for the center information strip;

$$\begin{aligned} s(-2, 1) &= p(v_2) \\ s(-1, 2) &= q(v_2) \\ s(0, 3) &= r(v_2) \end{aligned}$$

for the top information strip; and similar statements for the other three information strips. With states and template location thus defined, f_2 yields at time $t + 1$

$$s(0, 0) = r(v').$$

Similar remarks apply to the middle and bottom bits of the information strip.

Whenever the location-sensing test indicates that T_2 is centered on one of the permanently quiescent cells (2, 3), (3, 2), the transition function f_2 must leave it in the quiescent state. The remaining four cells (1, 1), (1, 2), (2, 1), (3, 3) are either all marked or all quiescent. If marked, f_2 leaves them marked. If not marked, all five information strips are sensed and f_8 applied in the simulated manner described above. If $p(v') = q(v') = r(v') = 0$, the cell upon which T_2 is centered is left in the quiescent state. Otherwise, it is transformed to state 1.

REMARK. This simulation procedure can be readily generalized to the case in which a cellular space with m states per cell is to simulate a cellular space with the von Neumann neighborhood and n states per cell. Further investigation would be needed for other types of neighborhood in the space to be simulated.

Chapter 4

A Universal 8-State, 5-Neighbor Cellular Space

4.1 Introductory Remark

In this chapter our aim is to provide the reader with an *understanding* of the transition function. For this purpose a bare definition of the function (e.g., by a listing of neighborhood states and their images) would be entirely inadequate.

A precise definition is of course provided, but first we stress the objectives and motivation. Then we proceed to discuss various behavioral subgoals and the resulting constraints on the function. These subgoals represent the few which were actually realized during the experimental work associated with the development of this function. As pointed out in the chapter on methodology most of the experiments were unsuccessful.

It would result in an excessive amount of detail if every constraint on the transition function were to be motivated or interpreted. However, the exposition which follows should be ample for the reader to understand the reason(s) for any transition not treated here specifically.

The particular neighborhood selected for the 8-state, 5-neighbor space is the von Neumann neighborhood, which for any cell c consists of c itself together with its four immediate nondiagonal neighbors. This neighborhood is of course rotation-symmetric. In contrast with von Neumann, we shall require the transition function to be strongly rotation-symmetric.

It is important to note that the transition function f which is described and defined in this chapter is a partial function. That is to say, it is defined on some but not all neighborhood states. Its domain of definition is, in fact, a very small subset of the set of all possible neighborhood states.

Why is f partial? First, the given partial function f is adequate to yield a set of behavioral properties which are sufficient to demonstrate the universality of the space in all of the senses currently of interest. Actually, of course, a partial transition function defines a whole class of spaces[1] and we shall, in one stroke, demonstrate the universality of every one of them.

Second, other types of universality are conceivable and some of them may become important. We accordingly wish to leave f in its partially defined state in case there should exist an extension of it which gives rise to a space with these other types of universality in addition to those treated here.

4.2 Principal Objectives

A space is required in which it is possible to embed a self-reproducing, universal computer-constructor as defined in Chapter 2 (see Section 2.5). Accordingly, our principal objectives are as follows:

1. There shall be a set P of completely passive states with the property that all other completely passive sets of states are subsets of P.
2. All configurations over P shall be constructible, readable, and erasable by some configuration in the space.
3. *Every* Turing-computable function from P^* (the set of all configurations over P) into P^* shall be computable by some configuration in the space.
4. At least one of the configurations in P^* shall upon stimulation become a machine capable of all the constructing, reading, and erasing operations specified in (2) above.

[1] In spite of this, we shall continue to refer to *the* space, as though there were just one defined by f.

4.3 Subordinate Objectives

In establishing the universality of the space to be defined we wish to show that all the desired computing, constructing, reading, erasing, and stimulating operations can be carried out in a space which, except for the computer-constructor and the configuration it is operating upon, is entirely quiescent. Thus, it will be helpful to assume in what follows that this condition holds.

Acceptance of this condition implies that, for example, in planning the construction of any desired configuration over (0, 1), we cannot assume that the *site*[1] is staked out in advance or that there are configurations already in the *construction region*[2] which are capable of receiving messages from the computer-constructor.

On the contrary, the computer-constructor must construct a path of some kind from itself to the construction site. Then, whatever information is needed to proceed with the construction can be sent to the site over this path.

Pathless propagation would, on the other hand, require a receiver of some kind to be at the site from the beginning. It is for this reason that pathless propagation (while interesting in itself) was deemphasized early in the experimental work. Instead, paths of various kinds were investigated with respect to how information might be propagated along them and how readily they might be extended in and retracted from various directions.

Here it must be remembered that the space—being strongly rotation-symmetric—does not possess a distinguished direction. Thus it is necessary to deal in *changes of direction* rather than in absolute directions—a factor which accounts for some of the more subtle of the differences between this work and the work of von Neumann, Burks, Thatcher, and Lee. (Note for example the relative directions "left," "right," and "ahead" which appear in the set of commands for the universal computer-constructor introduced in Chapter 6 and compare

[1] The construction site is the finite area in which a particular structure is to be erected or erased or read.

[2] The construction region, on the other hand, is the infinite area in which the construction site may be arbitrarily located.

them with the absolute directions "left," "right," "up," and "down" which appear in Thatcher's set of commands.)

The requirement that all configurations over some nontrivial[1] subalphabet of states be readable has a very important implication: either it must be possible to read a row of cells whose states belong to this subalphabet without requiring access to both sides of the row, or if access to both sides is used, it must be possible to move a path into a 2-dimensional configuration in such a way that the part of the configuration which is overwritten is "remembered" by the overwriting transformation.

At this time in the von Neumann 29-state space no method is known for

1. Reading an arbitrary linear configuration over the maximal completely passive subset of states.
2. Reading an arbitrary *2-dimensional* configuration over any nontrivial subalphabet of states.

It will be demonstrated that both of these operations are feasible in the 8-state space to be defined.

The operations *and* and *or* of logic are taken by von Neumann as primitives (they are associated with the confluent states and the transmission states respectively). On the other hand, he synthesizes the operation of *negation*. In our space we shall synthesize all the logical operations—none of them will be taken as a primitive.

The transition function f is complicated—as might be expected of an 8-state, 5-neighbor space which exhibits the kind of universality in which we are interested. In order to make the description of a universal computer-constructor in this space intelligible (let alone the problem of creating the design in the first place), it is necessary to provide a direct and simple means of "escaping" from the complexities of f and arriving at a higher level of organization.

This is done by developing a basic set of *components* (see Chapter 5) from which a universal computer-constructor can be built and explained without referring back to the transition function again. For

[1] A nontrivial alphabet contains two or more letters.

similar reasons further escapes are provided to higher levels of organization—for example, the signal sequence level (see latter portion of this chapter), the microprogram level, and the program level (see Chapter 6).

4.4 The Eight States

The eight states are labeled 0, 1, . . . , 7 and as in earlier chapters we shall take 0 as the quiescent one. It is convenient to divide the states into two disjoint subsets: one for representing structure and one for communication purposes.

The first subset (0, 1, 2, 3) is called the structure subset and its elements are called the *structure states*. The second subset (4, 5, 6, 7) is called the *signal subset* and its elements are called *signal states*.

Almost every configuration over (0, 1, 2, 3) is passive—we shall see later why a few exceptions exist. By contrast, every configuration which contains one or more signal states is active.

The structure subset can also be subdivided into two disjoint subsets: (0, 1) is the maximal completely passive subset for this space and is therefore the alphabet P which is crucial to each one of the four principal objectives (q.v.); (2, 3) can be thought of as the developed structure subset, since these states are developed during the transition phase in which a machine initially constructed in completely passive

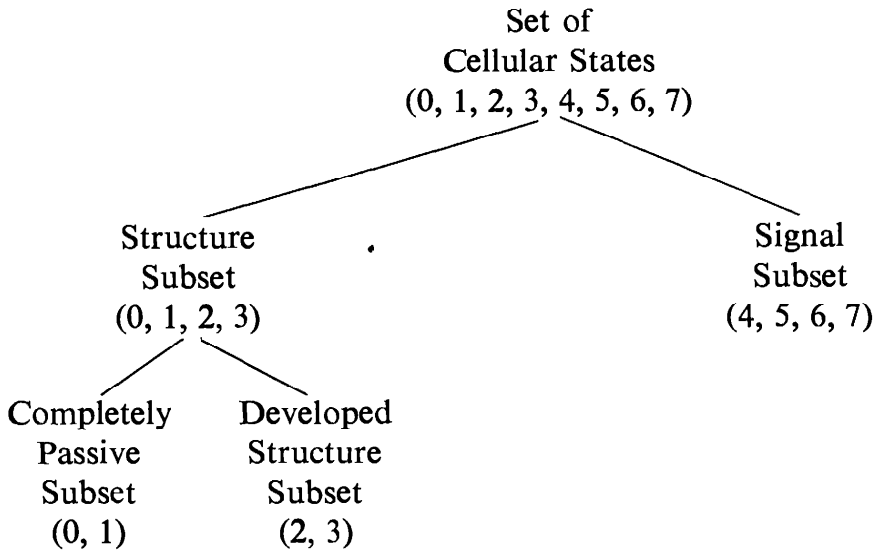


FIG. 4.1. Classification of cellular states.

form is transformed into a fully active form. Figure 4.1 summarizes the classification of states.

4.5 The Definition of the Transition Function f

The transition function f is defined in terms of two tables. One of these, the Short Table, lists those transitions which are exceptions to the rule that the state of the center cell remains unchanged in any neighborhood whose alphabet is (0, 1, 2, 3). The other, the Long Table, lists transitions for neighborhoods whose alphabet is not (0, 1, 2, 3).

Many of the transitions are invoked not in one but in several different types of behavior. Accompanying each transition is a reference to a sample of behavior in which this transition is invoked.

With a suitable computer and computer program, the interested reader can load the two tables together with sample initial configurations and observe in more detail than can be given here how all the various kinds of behavior are realized. The tables appear in Section 4.27.

4.6 Paths and Signals

The simplest form of path is a row of adjacent cells all in state 1. In a strongly rotation-symmetric space such a path is inherently bi-directional, a property that is highly desirable in view of our goal of reading all possible 2-dimensional configurations over (0, 1). Note also that such a path is completely passive so long as it is not acted upon by states other than 0, 1 in neighboring cells.

In order to make a signal propagate down such a path in one direction only, the signal must be a configuration which possesses a distinguished direction. The simplest form such a signal can take is a sequence of just two states, neither of which is state 1, and one of which is a signal state. Let **s** denote a signal state (i.e., 4, 5, 6, or 7). Then the four types of signals have the form 0**s**, and they all propagate so that **s** leads and 0 trails.

To the simple linear path we add right-angled corners and T-junctions so that signals may be dispatched in various directions to possibly multiple destinations (using the junctions as fan-outs) and from possibly multiple sources (using the junctions as fan-ins). A machine in completely passive form consists of nothing more than a network of such paths.

4.7 The Sheathed Path

To send information out from a computer-constructor to an arbitrary location in the construction region, we have seen that a path of some kind must be extended out into a quiescent area. It is highly desirable that the extension—in whatever direction required—be feasible using only the path to be extended as the carrier of signals which put the extension into effect. In other words we shall rule out the use of a second path to supply information for the extension of the first path.

We are accordingly faced with the problem of constructing left turns and right turns in a path by means of signals propagated down the path itself. In order to distinguish left from right in this strongly rotation-symmetric space, a path consisting solely of a string of cells in state 1 is inadequate.

The solution to this problem adopted here is to introduce a second kind of path which we shall call a *sheathed path*. A sheathed path consists of an unsheathed path—which constitutes its *core*—augmented by a layer of cells on both sides of the core, each of these cells being in state 2.

The idea of sheathing paths arose partly from considering the myelin coating of the axons of neurons (we do not claim, however, that any close correspondence exists here) and partly from early experiments in a 3-state space wherein much trouble was encountered with signals "leaking" into a partially completed structure instead of performing the desired construction operation at the end of the construction path—the sheath acts as an insulator and prevents such leakage.

Incidentally, the sheath does not alter the inherently bidirectional nature of the path.

4.8 The Three Phases of Construction

The construction of a machine at a site known to be totally quiescent is carried out in three phases. The first phase consists of building a completely passive network of unsheathed paths. This configuration over (0, 1) is developed step by step using the construction path to carry the signals for converting appropriate cells from state 0 to state 1.

The second phase consists of injecting at an appropriate entry point (or entry points) a signal which propagates throughout the entire network of paths, sheathing each path which it traverses. The signal which performs this transformation is 06 and, for this reason, it is called the *sheathing signal.* The structure which results from the sheathing operation—a network of sheathed paths—is a configuration over (0, 1, 2) and is still passive (but not completely passive).

The third and final phase consists of injecting at one or more of the entry points used in phase 2 a signal which activates the structure. This signal—the *activating signal*—is 07 and from it all other signals can be generated. It is capable (under appropriate conditions) of introducing state 3 into the sheath of a path and then a *node* is said to be formed. We shall see later that nodes provide the key to all the logical operations and to many of the construction operations.

4.9 Propagation of Signals

Since only the sheathing signal needs full mobility over unsheathed paths, it is the only signal for which the transition function makes such provision. Signal 05 is given restricted mobility in connection with the delicate operation of injecting the sheathing signal (see Section 4.25).

On the other hand, all four signals propagate down sheathed paths in a uniform manner. So we shall discuss this propagation first.

4.10 Propagation down Sheathed Paths

Let us consider some of the transitions necessary for the propagation of signals down sheathed paths.

Figure 4.2 shows the signal 0s propagating down a linear sheathed path. Note how in the interval of one time step the signal has shifted by one cell in the direction defined by the signal itself. To put this behavior

TABLE 4.1
TRANSITIONS FOR LINEAR PROPAGATION

Neighborhood state[a]	Image	Comment
10212	1	Stability of core
012s2	1	Regeneration of core
s0212	0	Propagation
112s2	s	Propagation
20212	2	Stability of sheath
20202	2	Stability of sheath
202s2	2	Stability of sheath
00002	0	Nonradiation of sheath

[a] The leftmost digit of the neighborhood state is the state of the center cell of the neighborhood. The next four digits are the states of the four (noncenter) neighbors taken clockwise and so as to yield the smallest 4-digit integer.

into effect we need transitions as listed in Table 4.1. Justification for each transition is given in the comment column of the table.

So that we can dispense with detailed specification of many of the constraints on f needed for stability and nonradiation reasons, we shall henceforth assume that all configurations over (0, 1, 2, 3) are passive

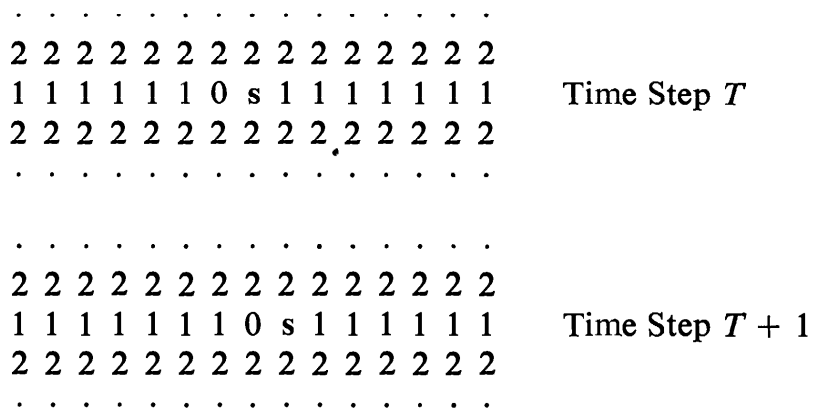


FIG. 4.2. Propagation down linear sheathed path.[1]

[1] The numeral 0 and middle dot · both represent state zero.

unless otherwise noted. Thus, in general, the image of any neighborhood state over (0, 1, 2, 3) is equal to the state of the center cell of that neighborhood.

4.11 Corners and Junctions

Additional constraints on f are needed to enable signals to negotiate the corners and junctions which they may encounter. Figure 4.3 shows one signal negotiating a left-hand corner and another negotiating a right-hand corner. Table 4.2 shows transitions needed for this behavior.

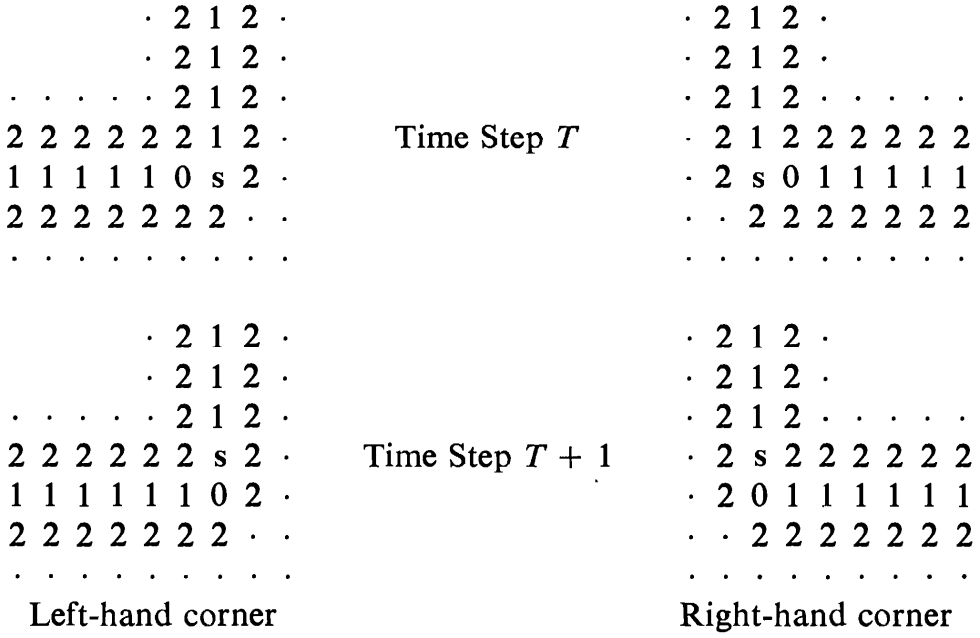


FIG. 4.3. Signals cornering.

A lone signal encountering a T-junction, no matter upon which arm of the T it makes its approach, splits into two copies, one of which

TABLE 4.2
TRANSITIONS FOR SIGNALS CORNERING

Neighborhood state	Image	Comment
s0122	0	Left corner at T
s0221	0	Right corner at T
01s22	1	Left corner at $T + 1$
0122s	1	Right corner at $T + 1$

proceeds along one exit, and the other along the remaining exit. Thus, a single signal can be dispatched to many destinations. Later, we shall see how this behavior at a T-junction is exploited to provide fan-in as well as fan-out (Section 5.3).

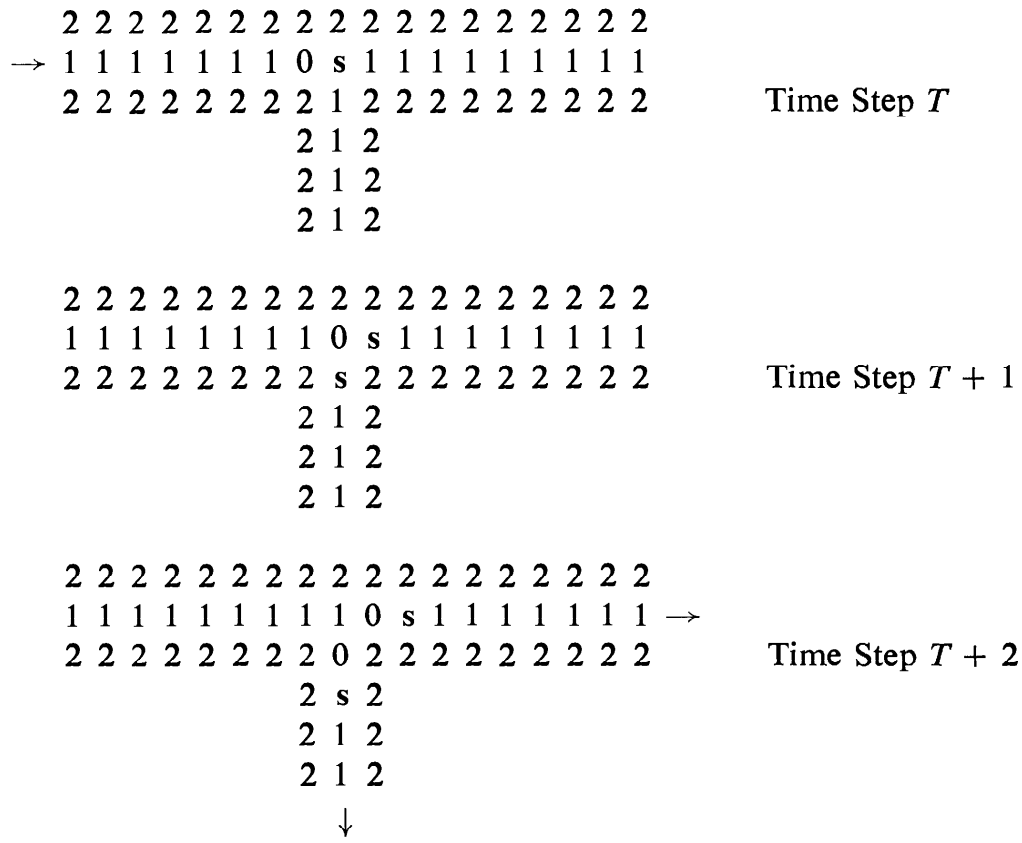


FIG. 4.4. Duplication of signal at T-junction.

Figure 4.4 shows the duplicating behavior for a signal approaching on the left arm. Table 4.3 lists corresponding transitions. Additional transitions are, of course, needed for the other two directions of approach.

TABLE 4.3

TRANSITIONS FOR SIGNAL AT T-JUNCTION

Neighborhood state	Image	Comment
111s2	s	Step $T - 1$ (not shown above)
s0211	0	Step T
012ss	1	Step $T + 1$

4.12 Collision of Signals

Suppose that the initial configuration placed in the space has the property that each signal state is an even distance with respect to the city-block metric from every other signal state (see Section 3.1 for the definition of this metric). Then the present transition function (in contrast to certain extensions of f) preserves this property. It does not, however, preserve the corresponding odd-distance property. Moreover, it contains no provision for handling the collision of signals whose signal states are an odd distance apart. This is so because all the required behavior can be obtained with only one signal state in the initial configuration, and then the even-distance constraint is automatically met. (In this connection note that the duplication of signals which takes place at T-junctions generates copies which are an even distance apart.)

The collision of two similar signals which were previously an even distance apart results in their mutual annihilation unless the collision takes place at a T-junction. The process of annihilation is shown in Fig. 4.5. Corresponding transitions are listed in Table 4.5.

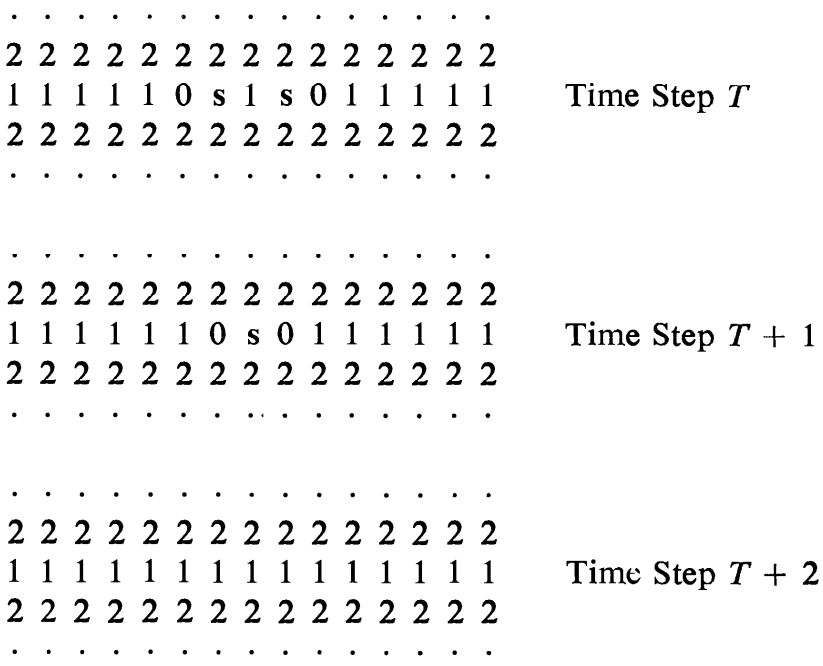


FIG. 4.5 Collision of signals on linear path.

At a T-junction the two colliding signals combine to form a third which we shall call the *product signal.* Table 4.4 is the multiplication table for this product.

TABLE 4.4
SIGNAL MULTIPLICATION TABLE

$07 \times 07 = 04$
$04 \times 04 = 05$
$05 \times 05 = 06$
$06 \times 06 = 06$

The first three entries in this table permit the generation in a simple manner of the signals 04, 05, 06 starting from a single signal 07. The given 07 is fanned out and then fanned in again as in Fig. 4.6. Provided that the two branches are of the same length, the two copies of the original 07 signal will arrive simultaneously at the second T-junction, and thus form the product signal 04. A similar network of paths converts 04 to 05, 05 to 06 (see Section 5.7). The purpose of the last entry in Table 4.4 will become evident in the next section.

TABLE 4.5
TRANSITIONS FOR COLLISION

Neighborhood state	Image	Comment
12s2s	s	Time Step T
s0202	1	Time step $T + 1$

No provision is made for threefold or fourfold collisions, since we can obtain all the required behavior:

1. Without arranging for threefold collisions at T-junctions.
2. Without using X-crossings at all.

4.13 Sheathing an Unsheathed Path

A single 06 signal injected at some point into a network of unsheathed paths is to traverse every part of the network which is accessible

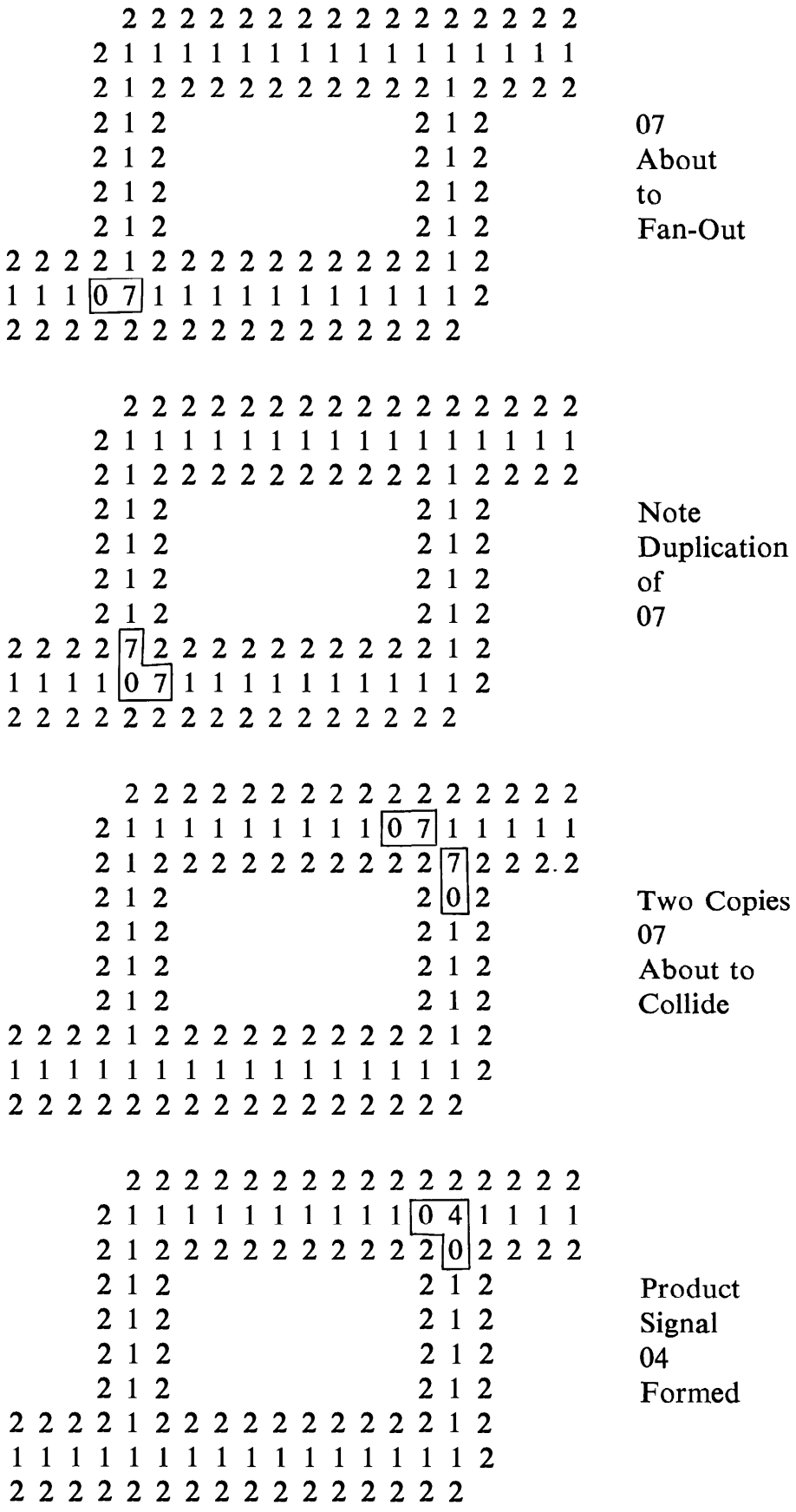


FIG. 4.6. Formation of 04 from 07.

from this point, duplicating itself at all T-junctions except those at which two copies of the signal arrive simultaneously (note that no provision is made for X-crossings). The process ends when all copies of the sheathing signal have been annihilated either by pairwise collision or by reaching path ends—the behavior of 06 at path ends is discussed below.

Each path in the network is traversed exactly once. Considerable complications would arise if this were not so. To see how this is achieved consider the looped path shown in Fig. 4.7.

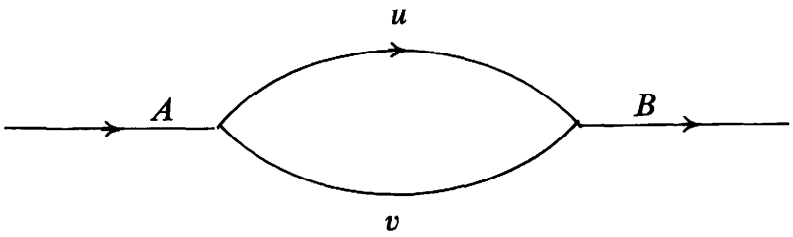


FIG. 4.7. Looped path.

First suppose the two branches u, v of the loop are of equal length. A sheathing signal entering at the left duplicates at the first junction A. The two copies of the signal arrive simultaneously at the second junction B. Accordingly the product signal is formed. Now we see why Table 4.4 stipulates that 06 shall be idempotent. A single copy of the signal 06 therefore emerges at the right leaving the loop entirely sheathed and free of signals.

The second case to consider is that of unequal length branches. Let u denote the shorter branch and v the longer. As before the signal 06

TABLE 4.6
TRANSITIONS FOR SHEATHING

Neighborhood state	Image	Comment
00026	2	Sheathing right
00062	2	Sheathing left
00006	2	Sheathing at corner
20612	2	Stability of sheath
20216	2	Stability of sheath
10106	6	Linear propagation
10621	6	Cornering left
10126	6	Cornering right

entering from the left duplicates at A. Now, however, one copy—the one traversing branch u—reaches B before the other and accordingly duplicates itself there. Thus, we obtain a copy of the signal emerging at the right, another headed toward B on branch v and a third headed toward A on branch v. The two copies on branch v inevitably collide and annihilate one another. The final state of the loop is therefore as in Case 1 above, namely, completely sheathed and free of signals.

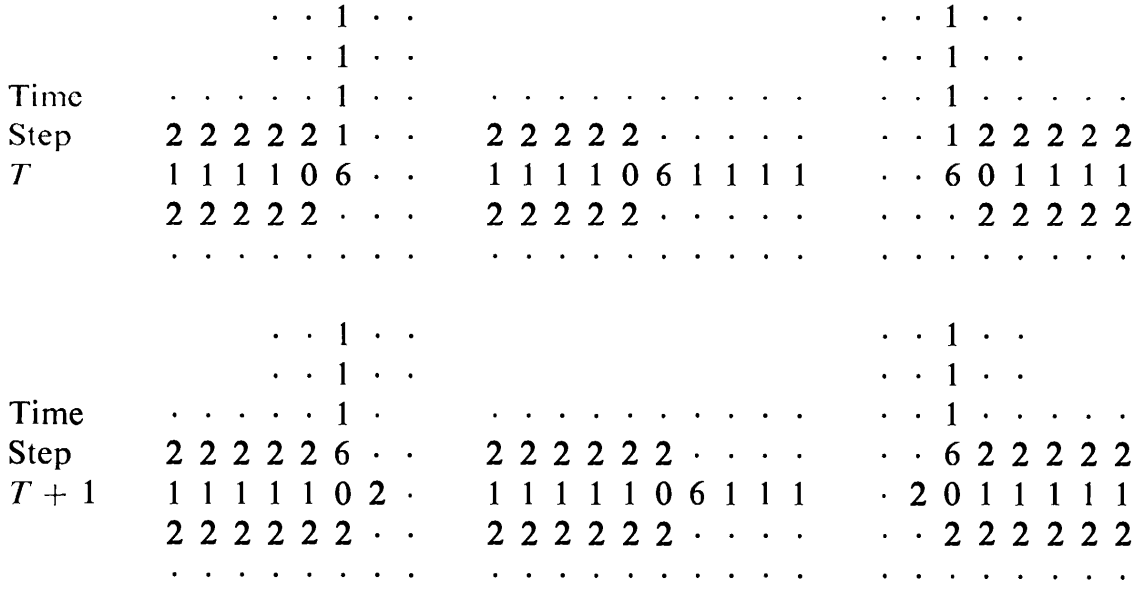


FIG. 4.8. Various sheathing situations.

Figure 4.8 shows various sheathing situations and Table 4.6 some of the associated state transitions.

4.14 The Cap on a Path End

An unsheathed path ends at a cell in state 1 which has just one other cell of its neighborhood in that state and the three remaining cells quiescent. These three quiescent cells are labeled clockwise p, k, q (see Fig. 4.9).

When the sheathing signal reaches the end of the path, it causes all three of the cells p, k, q to go into state 2. So long as k is in state 2 the path end is said to be *capped*. We shall regard the sheathed and capped state to be the standard one for path ends.

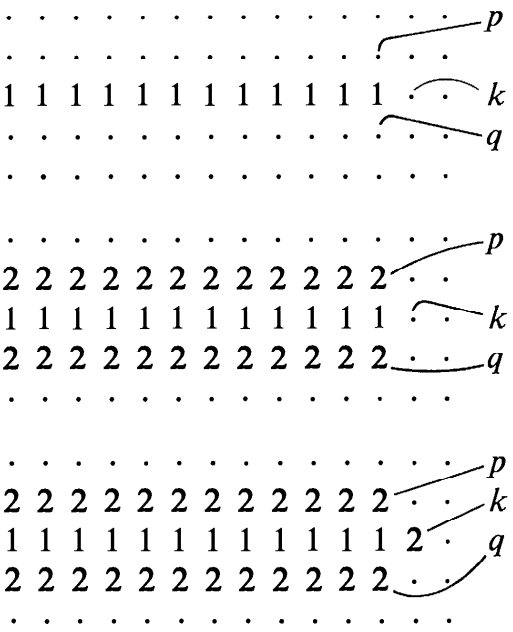


FIG. 4.9. Path ends.

A sheathed and capped path end may be *decapped*—i.e., its cell k returned to the quiescent state—by sending another signal 06 to the end.

As will be seen in later sections cell k plays a crucial role in reading, writing, and erasing operations, while cells p, q are important in extending paths left and right and in retracting them.

4.15 Gates and Gating

When the cap of one path (say A) is in contact with the sheath of a second (say B), we refer to the local configuration as a *gate*, to path A as the *control path* for this gate, and to path B as the *subordinate path* (see Fig. 4.10). The cell in B's sheath which is in contact with the cap of A is called the *gate node*. A gate node is never at a corner, junction, or end of a path.

It is by changing the state of gate nodes from 2 to 3 or vice versa that all the necessary logical operations and control over transmission of information can be realized.

The activating signal 07 is the only one which can turn a gate on (i.e., change the state of its gate node from 2 to 3). Both signals 06, 07

can turn gates off (3 to 2). The sheathing activity of 06 would be crippled if 06 were allowed to turn gates on. Signals 04, 05 are not sent over control paths because of their marking properties, discussed below.

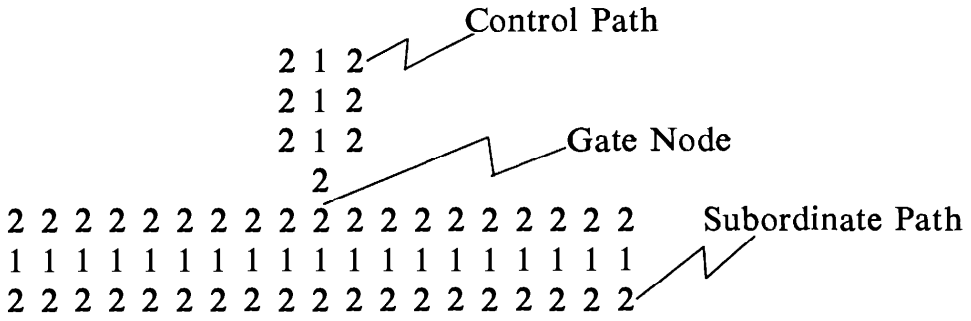


FIG. 4.10. Gate in off state.

The action of signal 07 upon reaching the end of the control path of a gate is shown in Fig. 4.11. If the gate is off, it turns it on; if the gate is on, it turns it off. When not acted upon by a signal on the control path, the state of a gate node remains unchanged. Gates therefore have the latch property.

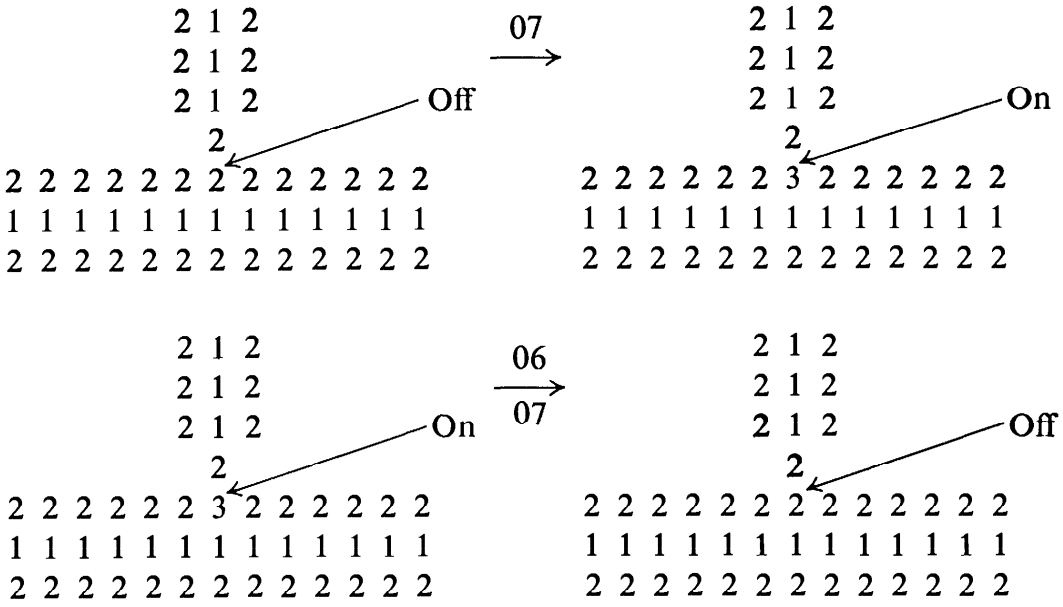


FIG. 4.11. Turning a gate on and off.

The effect of a single gate in the *on* state upon the subordinate path is to convert it from a bidirectional path into a unidirectional one. Thus a signal (whatever its type) proceeding along the subordinate path in a

direction such that the node is located on the left-hand side of the signal as it approaches is allowed to pass (unchanged in form), while any signal going in the opposite direction is annihilated.

Two gates in the *on* state, one on one side of the path, the other on the other (but not directly opposite each other), have the effect of completely blocking that path to all signals. Two gates in the *on* state, one on each side of the path but, now, directly opposite each other act as a signal transformer (see Section 5.8), which allows signals to pass in either direction, but in so doing changes them to the activating signal 07. This function plays an important role in achieving a crossover scheme which permits paths carrying any of the four signals effectively to cross (see Section 5.9).

4.16 Path Extension and Signal Sequences

Assume we have a sheathed and capped path terminating in a quiescent area. The operation of extending this path ahead by one unit entails the use of a simple sequence of signals: 07 followed by 06. Signal 07 converts the cap to state 1, thus extending the core by one unit. Signal 06 then sheathes and caps the exposed core, yielding the desired result (see Fig. 4.12).

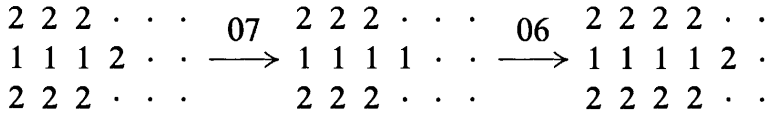


FIG. 4.12. The *extend* operation.

In this operation of extending a path we encounter for the first time a signal sequence: i.e., a sequence of signals traveling along a common path in the same direction. The only constraint upon the separation of these signals is that the signal states of successive signals must be separated by at least four units. The transition function does not make provision for them being any closer.

4.17 The Marking Signals

In preparation for understanding the processes of extending a path left and right and of retracting a path left, right, and astern, we consider certain marking operations, which change the state of cells p, q at the end of a path.

Signal 04 is called the *left marking signal* and 05 the *right marking signal*, since 04 affects the state of p but not q, while 05 affects the state of q but not p (an example, by the way, of the exploitation of reflection asymmetry).

4.18 Changing the State of Cells p and q

Almost all of the operations at path ends have as suboperation a change in the state of p or q or both. Due to the fact that the space is reflection-asymmetric the neighborhood states of p, q are usually distinguishable even if state p equals state q. For example, when signal 0s (s = 4, 5, 6, 7) arrives at the path end (see Fig. 4.13), the neighborhood

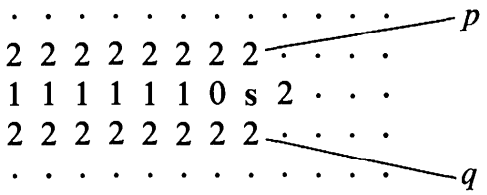


FIG. 4.13. Arrival of signal at path end.

states of p, q, are p00s2, q002s respectively (where, for convenience, we have used p to denote the state of cell p and similarly with q).

It is therefore possible to arrange for signal 04 to affect p but not q and for signal 05 to affect q but not p. The transition function, in fact, does just this. State diagrams for p, q are given in Fig. 4.14.

Each of the signals 06, 07 affects the state of *both* p, q; the corresponding state diagram is included in Fig. 4.14. The action of 07 on p when p is zero is undefined. A similar remark applies to q.

In passing let us take note of one of the transitions which is an exception to the passivity of the set (0, 1, 2, 3) and which arises indirectly as a result of the marking properties of signals 04, 05. Associated with every simple corner (not junction) are two cells in the sheath (the two outer-

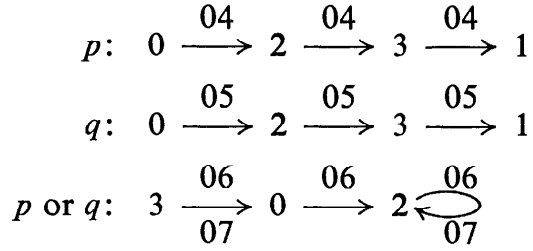


FIG. 4.14. State diagrams for cells p, q.

most cells in state 2), one of which "looks like" a p-cell to a passing 04 signal, and the other a q-cell to a passing 05 signal. Accordingly, when such a signal reaches a corner, one or other of the two cells is transformed to state 3. Its state is restored by means of the transition

$$f(3, 0, 0, 0, 2) = 2$$

which takes advantage of the zero belonging to the signal.

4.19 Path Extension Left and Right

Figure 4.15 shows the steps signal by signal by which a path is extended left, assuming its end is in standard form initially. Note how the core is extended by a single 1, while the cap makes a knight's move.

The first two signals of the sequence 04–04–05–06 convert cell p successively to states 3, 1 thus taking care of the left extension of the core. Now two different cells are the p, q cells of the path end; the new p, q states are 2, 0 respectively. The third signal of the sequence changes the state of q to 2 and the fourth simply caps the end of the path.

Figure 4.16 shows the steps by which a path is extended right. The symmetry between the operations *extend left* and *extend right* obviates the need for further explanation of the latter operation.

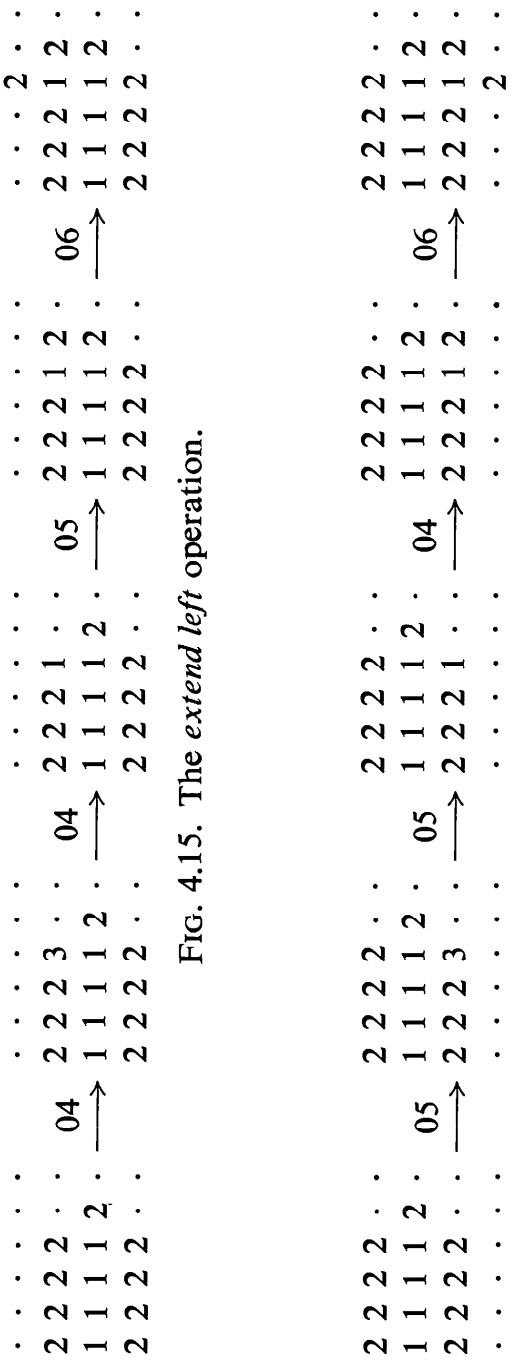


FIG. 4.15. The *extend left* operation.

FIG. 4.16. The *extend right* operation.

4.20 Path Retraction

The operation *retract* is the inverse of the operation *extend.* The signal sequence for this operation is 04-05-06-06 and the successive effects of these signals on the path end are shown in Fig. 4.17. Note that the signal sequence for retract should not be applied to a path end which is in either the left-extended state or the right-extended state shown as the final state in Figs. 4.15 and 4.16, respectively. The first signal (04) of the sequence converts p to 3; the second (05) converts q to 3; the third (06) converts both p, q to 0, decaps the path and sets the terminal core bit to 0; the fourth (06) finally caps the retracted core.

4.21 Path Retraction Left and Right

The operation *retract left* is the inverse of the operation *extend left.* The signal sequence for this operation is 05–06–06–06 and the signal-by-signal effects are shown in Fig. 4.18. The operation *retract right*, the inverse of *extend right*, is similarly depicted in Fig. 4.19.

4.22 Operations upon (0, 1) Configurations

The operations *mark*, *erase*, *sense and wait*, and *cap after sense* are concerned with the construction, destruction, and reading of arbitrary (0, 1) configurations. An important consideration common to these operations is the angle of approach which is employed in bringing the construction path up to, and in contact with, the perimeter cells of the (0, 1) configuration. Three possible approaches are shown in Fig. 4.20.

Whatever approach is used, a 2-dimensional (0, 1) configuration cannot be read without (at least temporarily) erasing some of it. We shall require, however, that it be possible always to avoid erasing any

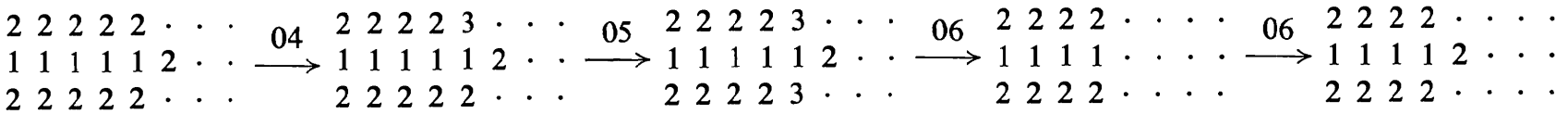


FIG. 4.17. *Retract.*

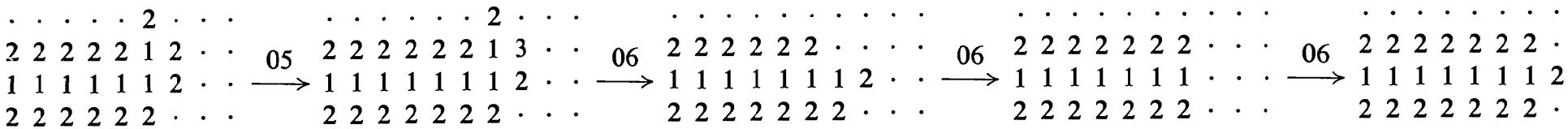


FIG. 4. 18. *Retract left.*

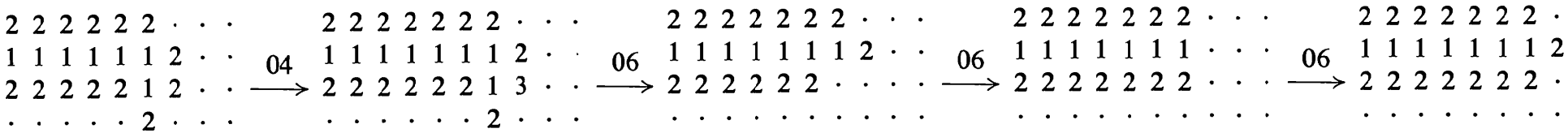


FIG. 4.19. *Retract right.*

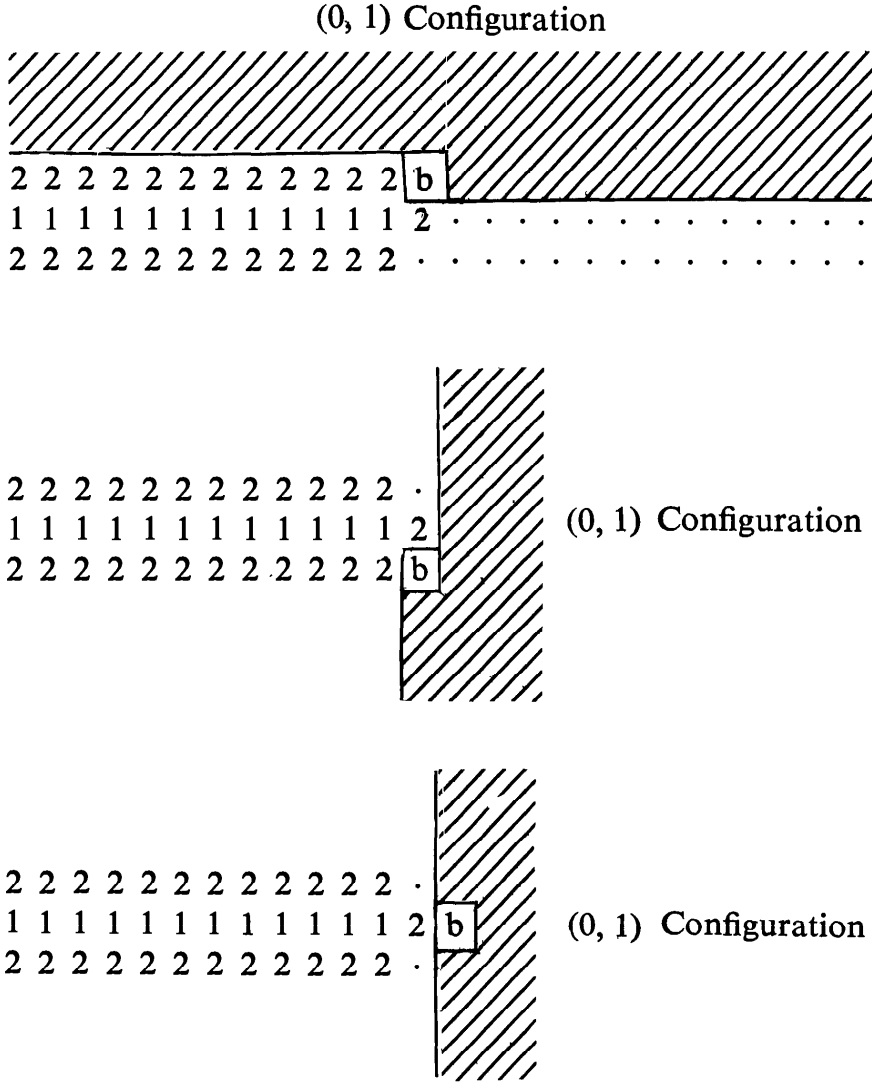


FIG. 4.20. Three approaches to construction.

part which has not yet been read. All three approaches shown in Fig. 4.20 satisfy this criterion.

The third approach was adopted for the following reasons:

1. It permits access by the construction path to an arbitrary cell on the perimeter of a (0, 1) configuration if that perimeter is linear.
2. It permits more symmetric treatment of the cells p, q and, in fact, yields a viable transition function.

One result of item (1) above is that, in contrast to the von Neumann space, no special design is needed for a tape unit. With regard to item (2) it is not known whether either of the first two approaches are viable in an 8-state, 5-neighbor space, but several attempts with the first failed.

4.23 Marking and Erasing

The operation *mark* applies to cell u under the path cap (see Fig. 4.21). It is assumed that u is in state 0 initially—whether this is actually the case can be determined by use of the *sense and wait* operation. Excluding the path cap (state 2), the neighbors of u are assumed to be in state 0 or state 1.

The signal sequence for this operation is 07–06–04–05–07–06 and the signal-by-signal effects are shown in Fig. 4.21.

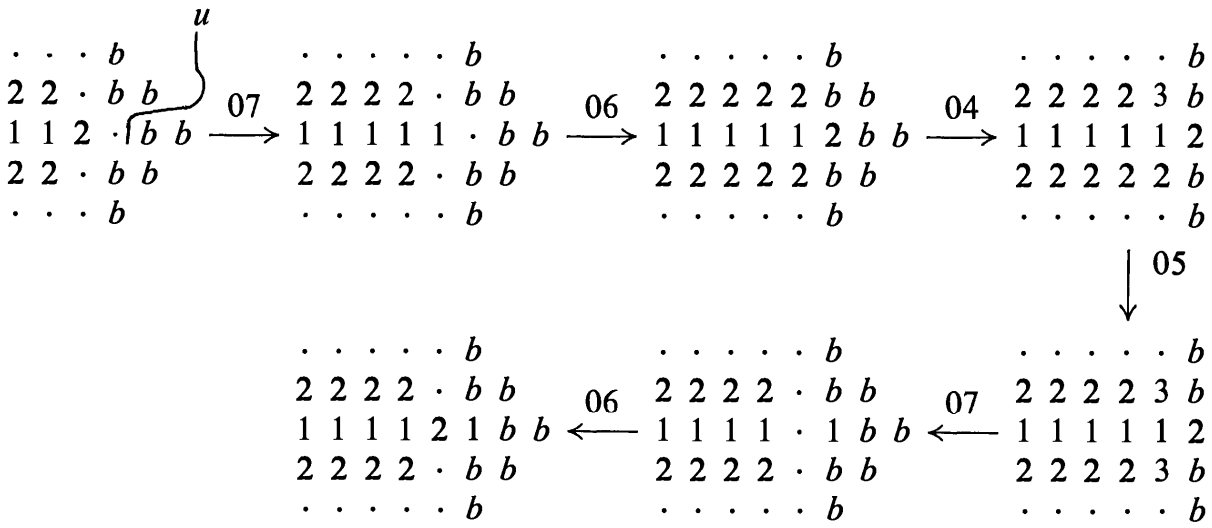


FIG. 4.21. The *mark* operation.

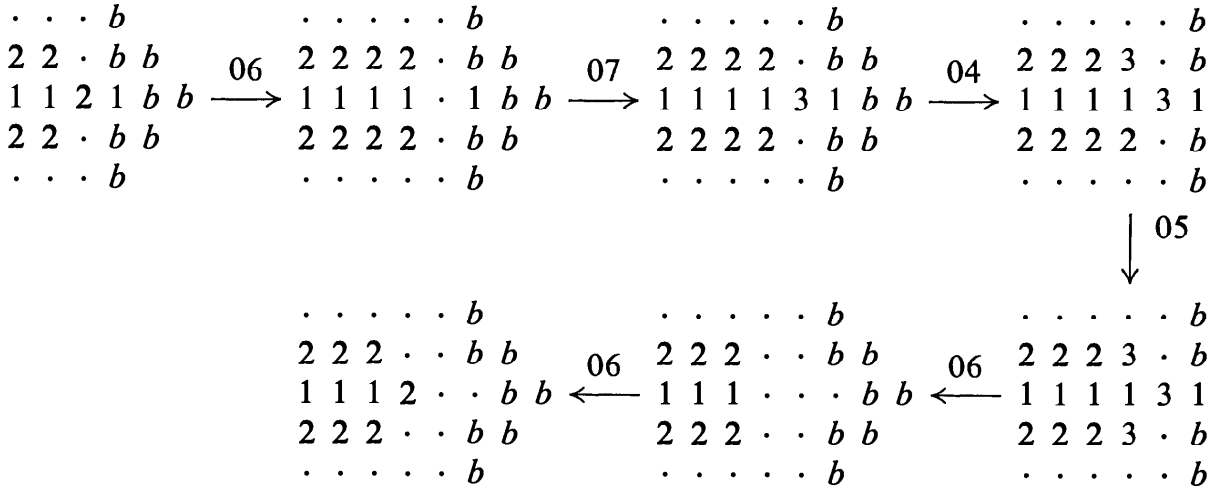


FIG. 4.22. The *erase* operation.

The operation *erase* is the inverse of *mark* (except for the final condition of the construction path) and is predicated on the same assumptions, except that u must be in state 1 initially. Among the operations on (0, 1) configurations, *erase* is exceptional in that at the end of the operation the path has effectively been retracted by one unit. This property places a constraint upon the closeness with which a construction path C may run parallel to the perimeter of a (0, 1) configuration before turning in toward that perimeter. If C is to be used for erasing, the minimum allowable distance between the perimeter and the core of C (when running parallel to the perimeter) is four units. If no erasing is required, the minimum allowable distance is three units.

The signal sequence for *erase* is 06–07–04–05–06–06 and the steps are shown in Fig. 4.22.

4.24 Sensing

Sensing whether cell u (the cell under the cap of the construction path) is in state 0 or state 1 is carried out by a combination of two operations: the *sense and wait* operation and the *cap-after-sense* operation. The first of these operations causes an *echo signal* to be generated.

The echo signal is either 06 or 07 according to whether cell u is in state 0 or state 1 respectively. The signal travels back along the construction path, presumably to the computer-constructor at the other end of the path.

Generation of the echo, which is put into effect by the signal sequence 07–07, leaves the end of the construction path C in a nonstandard condition. The process of standardizing the construction end of C must wait until the echo signal has arrived at the computer end of C and been switched on to some other path (see Section 5.11). When C is clear of signals once again, the signal sequence for *cap-after-sense*, namely 04–06, should be sent to standardize the construction end of C.

The steps for the sensing operations are shown in Figs. 4.23 and 4.24 for the two cases: u in state 0; u in state 1. Some extra detail is given to show how the different echoes arise.

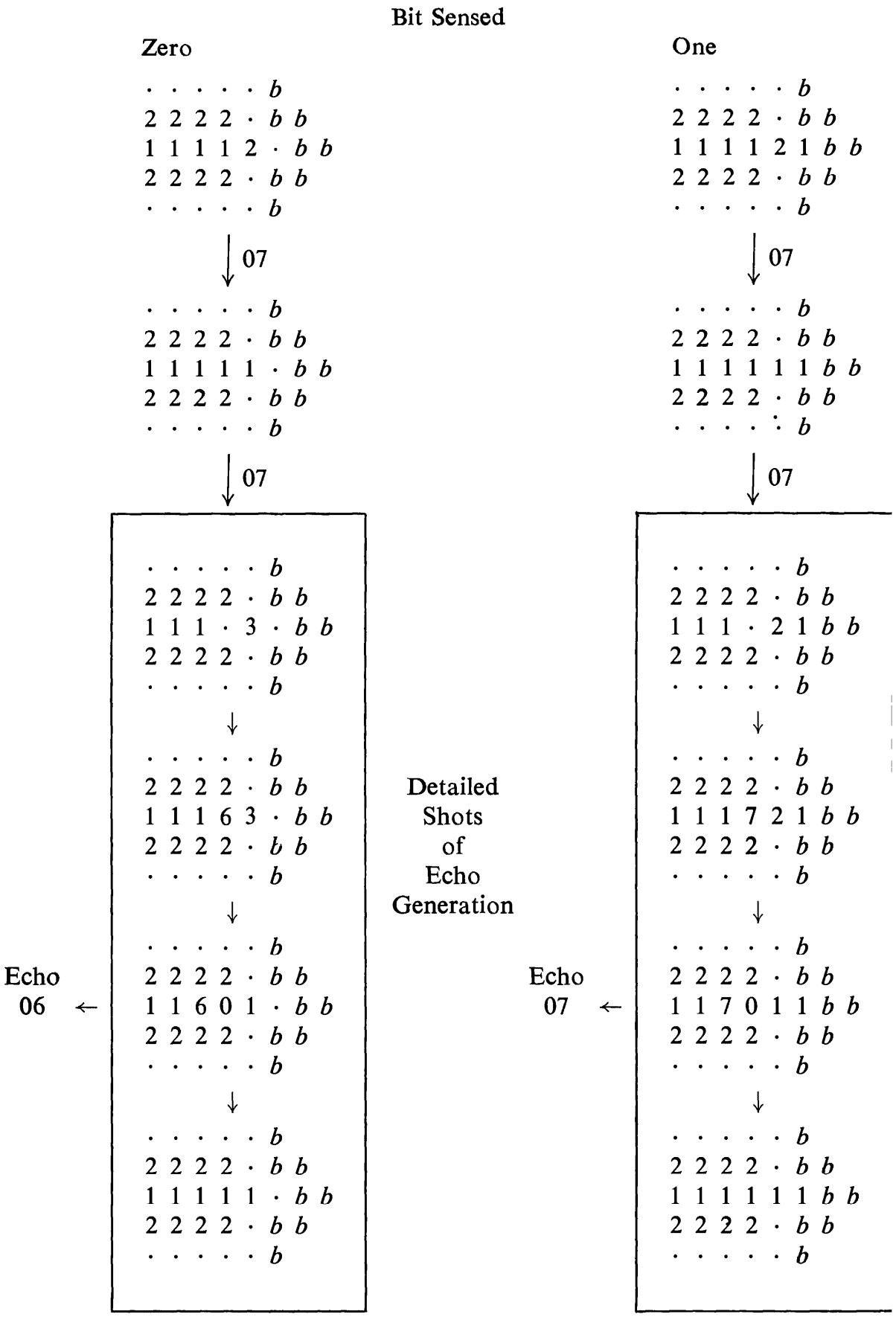


FIG. 4.23. The *sense* operation.

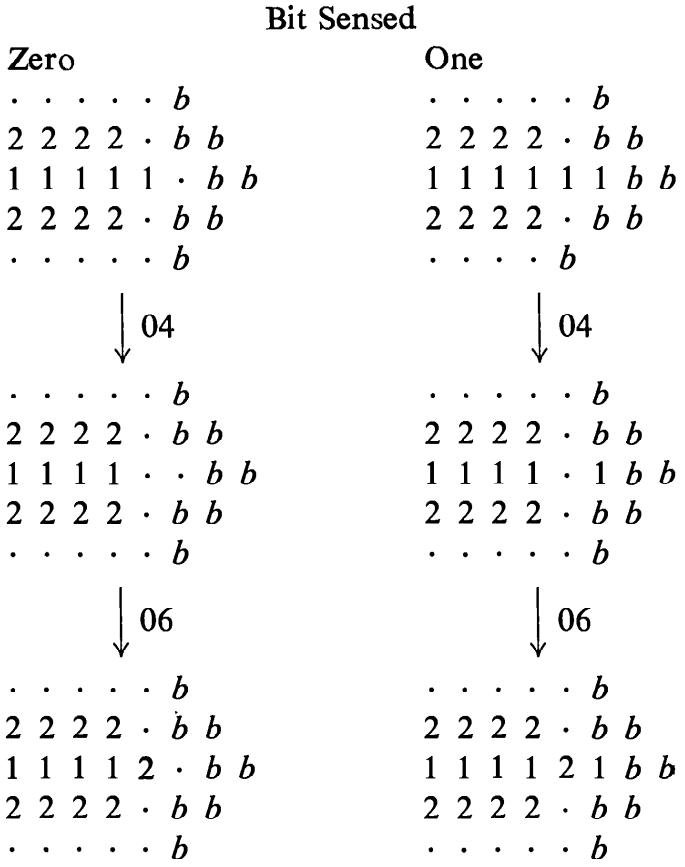


FIG. 4.24. The *cap-after-sense* operation.

4.25 Signal Injection

Suppose that through appropriate use of the various operations described we have constructed a configuration over (0, 1) which is the completely passive form of some machine M, say. We wish to activate M. To do this two signals must be injected into M: first, the sheathing signal 06, then after a suitable delay the activating signal 07.

One might be tempted simply to link the construction path C (which is sheathed) directly to some (unsheathed) path in M and then send a signal 06 out on C. This can be done, but the unfortunate result is that it virtually welds the offspring M to its parent machine, the computer-constructor at the other end of C. At best a very cumbersome operation would appear to be necessary to cut the umbilical cord.

Thus a signal other than 06—namely 05—is used to inject 06 into the offspring. For this purpose signal 05 is given limited mobility over an

unsheathed path. Its conversion to 06 requires a special subconfiguration, the *injection receiver*, in M (see Fig. 4.25).

The signal sequence for injecting signal 06 is 07–05–06. The last signal merely serves to restore the cap on path C. The injection receiver is left in a changed condition—a condition suitable for the subsequent entry of the activating signal 07.

Injection of this signal may be effected by the signal sequence 06–07–07–06–06 (see Fig. 4.26). Note how the third signal (07) gains entry to the offspring by use of a cell in state 3 as a bridge. Path C is

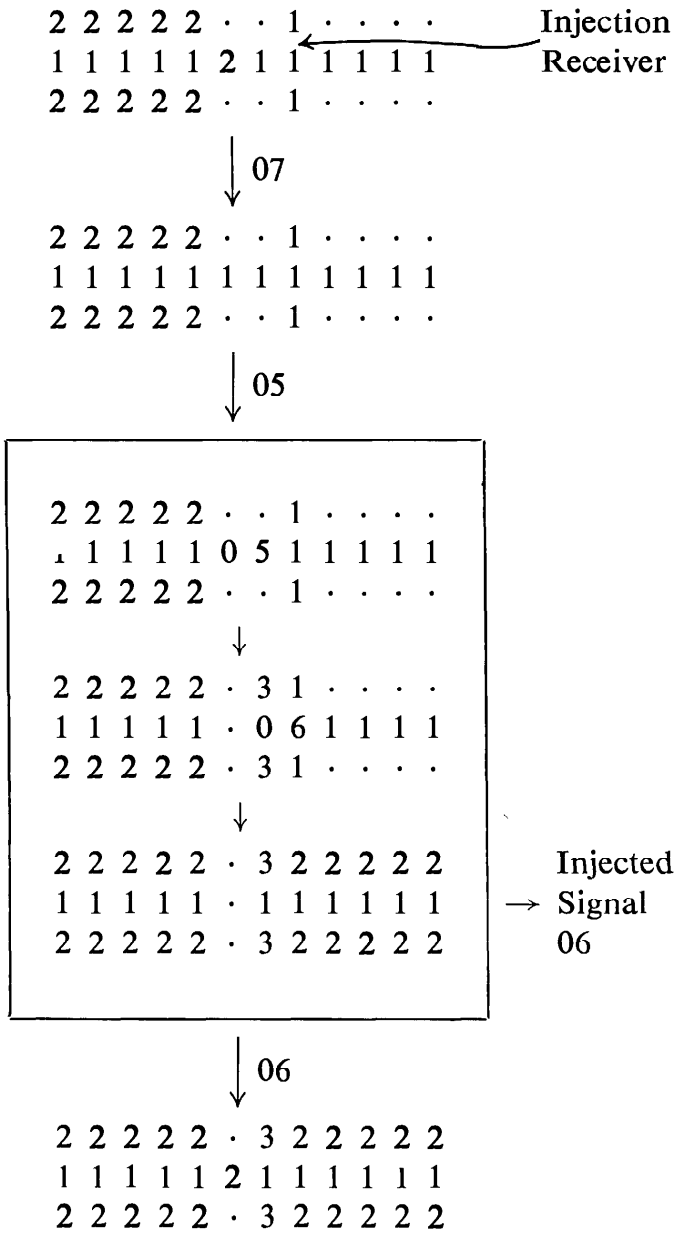


FIG. 4.25. The *inject signal* 06 operation.

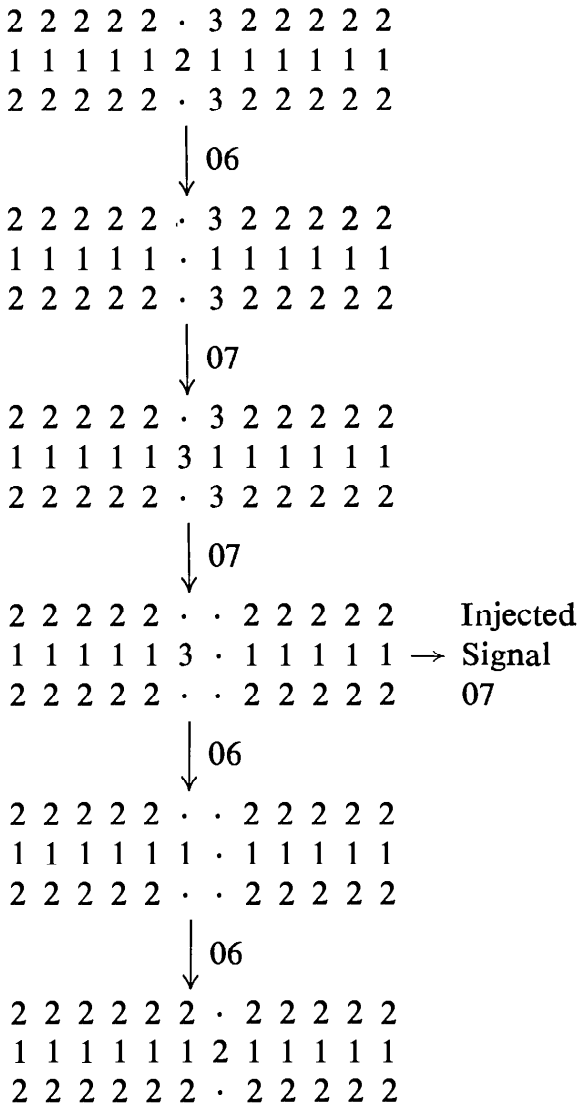


FIG. 4.26. The *inject signal* 07 operation.

left in a condition such that its end may be withdrawn from the offspring by use of appropriate retraction operations.

4.26 Summary of Signal Sequences and Operations

Table 4.7 summarizes the signal sequences and operations discussed above. Abbreviations for the operations are also included, since we shall need to refer to the operations many times in Chapter 6, where the design of a universal computer-constructor is presented.

Although an 8-state, 5-neighbor cellular space has now been described, we still have the task of demonstrating its universality. This will be completed in Chapter 6.

TABLE 4.7
SIGNAL SEQUENCES AND OPERATIONS

Abbreviation	Operation	Signal sequence
x	*extend*	07–06
xl	*extend left*	04–04–05–06
xr	*extend right*	05–05–04–06
r	*retract*	04–05–06–06
rl	*retract left*	05–06–06–06
rr	*retract right*	04–06–06–06
m	*mark if zero*	07–06–04–05–07–06
e	*erase if one*	06–07–04–05–06–06
sw	*sense and wait*	07–07
cs	*cap after sense*	04–06
i	*inject signal* 06	07–05–06
j	*inject signal* 07	06–07–07–06–06

4.27 Transition Function Tables

Transitions are encoded into 6-digit numbers as follows. The leftmost digit is the state of the center cell. The next four digits are the states of the (noncenter) neighbors taken clockwise so as to yield the smallest four-digit integer. The rightmost digit is the image under f of the neighborhood state represented by the leftmost five digits. Both a period and the digit zero are used to denote state zero.

Table 4.8 lists those transitions which are exceptions to the rule that the state of the center cell remains unchanged in any neighborhood whose alphabet is (0, 1, 2, 3). Table 4.9 provides a code for various types of behavior. This code is used in Table 4.10, which lists transitions for neighborhoods whose alphabet is not (0, 1, 2, 3). Accompanying each transition is a coded reference to a sample of behavior in which this transition is invoked.

TABLE 4.8
THE SHORT TABLE

Transition	Comment
012121	Path self-repair
3···22	Path self-repair
3·1·22	Gate control
3·1·30	Gate control
212323	Gate control
312322	Gate control
002131	Path end self-repair
012326	Echo generation
012227	Echo generation

TABLE 4.9
BEHAVIOR CODE

Code	Behavior	Code	Behavior
c	*collision* (not at junction)	*pg*	*propagation at gate* (subord path)
e	*erase*	*pm*	*cell p change-of-state*
fi	*fan-in*	*qm*	*cell q change-of-state*
fo	*fan-out*	*r*	*retract*
g	*gate control*	*s*	*sense*
i	*inject signal* 06	*sh*	*sheathing*
j	*inject signal* 07	*ss*	*stability of sheath*
k	*cell k change-of-state*	*t7*	*transforming to* 07
m	*mark*	*x*	*extend*
ns	*node stability*	*xl*	*extend left*
p	*signal propagation*	*xr*	*extend right*
pe	*path end*		

TABLE 4.10
THE LONG TABLE

0···40 *k*	0···50 *k*	0··140 *e*	0··350 *i*	0··410 *e*	0··520 *i*
0·1·40 *e*	0·1·50 *i*	0·1420 *e*	0·2410 *e*	0·2730 *j*	0·3·70 *j*
0·3720 *j*	0·3730 *j*	0·2721 *j*	0·3631 *i*	011241 *fi*	011251 *fi*
011261 *fi*	011271 *fi*	011421 *fi*	011521 *fi*	011621 *fi*	012141 *fi*
012151 *fi*	012161 *fi*	012171 *fi*	012241 *p*	012251 *p*	012261 *p*
012271 *p*	012351 *p*	012421 *p*	012441 *fo*	012521 *p*	012531 *p*
012551 *fo*	012621 *p*	012661 *fo*	012721 *p*	012731 *p*	012771 *fo*
013241 *p*	013421 *pg*	013521 *pg*	013621 *pg*	013721 *pg*	013731 *t7*
014221 *p*	014241 *fo*	014321 *p*	014421 *fo*	015221 *p*	015231 *p*
015251 *fo*	015321 *p*	015521 *fo*	016221 *p*	016261 *fo*	016621 *fo*
017221 *p*	017271 *fo*	017721 *fo*	0···62 *k*	0··162 *k*	0··252 *xl*
0··262 *sh*	0··422 *xr*	0··612 *k*	0··622 *sh*	0··662 *sh*	0·1·62 *sh*
0·1162 *k*	0·1262 *sh*	0·1612 *k*	0·1622 *sh*	0·1662 *sh*	0·2·62 *g*
0·2262 *sh*	0·2612 *sh*	0·2622 *sh*	0·6112 *k*	0·6212 *sh*	0·6222 *sh*
0·6262 *sh*	0·6612 *sh*	011162 *k*	011662 *sh*	022262 *sh*	022662 *sh*
0···73 *k*	0··153 *i*	0··513 *i*	0·1·73 *e*	0·2·73 *g*	1···40 *e*
1··140 *e*	1··410 *e*	1·1·40 *e*	1·1140 *e*	1·1410 *e*	1·4110 *e*
111140 *e*	113421 *g*	113521 *g*	113621 *g*	113721 *g*	1··362 *i*
1··632 *i*	1·1·72 *s*	1···73 *s*	1··244 *xr*	1·2424 *pe*	1·2434 *pe*
1·3424 *pe*	1·3434 *pe*	111244 *fo*	111424 *fo*	112144 *fo*	112244 *p*
112424 *p*	112434 *pg*	112774 *fi*	114224 *p*	117274 *fi*	117724 *fi*
122244 *pe*	122344 *pe*	122434 *pe*	122444 *c*	123244 *pe*	123344 *pe*
123434 *pe*	124244 *c*	124334 *pe*	133344 *pe*	1··525 *xl*	1·1·55 *i*
1·2525 *pe*	1·2535 *pe*	1·3525 *pe*	1·3535 *pe*	111255 *fo*	111525 *fo*
112155 *fo*	112255 *p*	112445 *fi*	112525 *p*	112535 *pg*	114245 *fi*
114425 *fi*	115225 *p*	122255 *pe*	122355 *pe*	122535 *pe*	122555 *c*
123255 *pe*	123355 *pe*	123535 *pe*	125255 *c*	125335 *pe*	133355 *pe*
1···66 *sh*	1··166 *sh*	1··266 *sh*	1··616 *sh*	1··626 *sh*	1··666 *sh*
1·1·66 *sh*	1·1166 *sh*	1·1266 *sh*	1·1616 *sh*	1·1626 *sh*	1·1666 *sh*
1·2266 *pe*	1·2616 *sh*	1·2626 *pe*	1·2636 *pe*	1·3626 *pe*	1·3636 *pe*
1·6·66 *sh*	1·6116 *sh*	1·6166 *sh*	1·6216 *sh*	1·6226 *pe*	1·6266 *c*
1·6616 *sh*	111156 *i*	111266 *fo*	111626 *fo*	112166 *fo*	112266 *p*
112556 *fi*	112626 *p*	112636 *pg*	112666 *fi*	115256 *fi*	115526 *fi*
116226 *p*	116266 *fi*	116626 *fi*	122266 *pe*	122366 *pe*	122636 *pe*
122666 *c*	123266 *pe*	123366 *pe*	123636 *pe*	126266 *c*	126336 *pe*
133366 *pe*	1·2727 *pe*	1·2737 *pe*	1·3727 *pe*	1·3737 *pe*	111277 *fo*
111727 *fo*	112177 *fo*	112277 *p*	112727 *p*	112737 *pg*	113437 *t7*
113537 *t7*	113637 *t7*	113737 *t7*	117227 *p*	122277 *pe*	122377 *pe*
122737 *pe*	122777 *c*	123277 *pe*	123377 *pe*	123737 *pe*	127277 *c*
127337 *pe*	133377 *pe*	2···60 *k*	2·1·60 *k*	2·2·60 *g*	2···71 *s*
2··171 *x*	2··711 *x*	2·1·71 *s*	2·1171 *x*	2·1711 *x*	2·7111 *x*
211171 *x*	2···42 *ss*	2···52 *ss*	2··142 *ss*	2··152 *ss*	2··162 *ss*

TABLE 4.10 (continued)

2··242 *ss*	2··262 *ss*	2··272 *ss*	2··342 *ss*	2··352 *ss*	2··362 *ss*
2··372 *ss*	2··412 *ss*	2··512 *ss*	2··522 *ss*	2··532 *ss*	2··612 *ss*
2··622 *ss*	2··632 *ss*	2··722 *ss*	2··732 *ss*	2·1·42 *ss*	2·1·52 *ss*
2·1142 *ss*	2·1152 *ss*	2·1162 *ss*	2·1412 *ss*	2·1512 *ss*	2·1522 *ss*
2·1532 *ss*	2·1612 *ss*	2·1622 *ss*	2·1722 *ss*	2·2142 *ss*	2·2152 *ss*
2·2162 *ss*	2·2172 *ss*	2·2242 *ss*	2·2252 *ss*	2·2262 *ss*	2·2272 *ss*
2·2412 *ss*	2·2422 *ss*	2·2432 *ss*	2·2522 *ss*	2·2532 *ss*	2·2612 *ss*
2·2622 *ss*	2·2632 *ss*	2·2712 *ss*	2·2722 *ss*	2·2732 *ss*	2·3422 *ss*
2·3432 *ss*	2·3522 *ss*	2·3532 *ss*	2·3622 *ss*	2·3722 *ss*	2·4112 *ss*
2·4122 *ss*	2·4222 *ss*	2·5112 *ss*	2·5122 *ss*	2·5222 *ss*	2·6112 *ss*
22·6122 *ss*	2·6162 *ss*	2·6222 *ss*	2·6262 *ss*	2·7122 *ss*	2·7222*ss*
2·7722 *ss*	211142 *ss*	211152 *ss*	211162 *ss*	211242 *ss*	211252 *ss*
211262 *ss*	211272 *ss*	211422 *ss*	211522 *ss*	211622 *ss*	211662 *ss*
211722 *ss*	211772 *ss*	212142 *ss*	212152 *ss*	212162 *ss*	212172 *ss*
212242 *ss*	212252 *ss*	212262 *ss*	212272 *ss*	212342 *ss*	212352 *ss*
212362 *ss*	212372 *ss*	212422 *ss*	212522 *ss*	212622 *ss*	212722 *ss*
213262 *ss*	213272 *ss*	214222 *ss*	214422 *ss*	215222 *ss*	215232 *ss*
215322 *ss*	215522 *ss*	216222 *ss*	216232 *ss*	216262 *ss*	216322 *ss*
217222 *ss*	217232 *ss*	217722 *ss*	222242 *ss*	222252 *ss*	222262 *ss*
222272 *ss*	222442 *ss*	222552 *ss*	222662 *ss*	222772 *ss*	2··253 *qm*
2··423 *pm*	2·1423 *pm*	2·2·73 *g*	2·2513 *qm*	2·3·63 *g*	2·3·73 *g*
223243 *g*	223253 *g*	223263 *g*	223273 *g*	3··260 *r*	3··270 *m*
3··620 *r*	3··720 *m*	3·1620 *pm*	3·1720 *m*	3·2610 *qm*	3·2710 *m*
3···61 *s*	3··251 *qm*	3··421 *pm*	323242 *g*	323252 *g*	323262 *g*
323272 *g*	3···43 *k*	3···53 *k*	3··523 *ns*	3·1·43 *e*	3·1·53 *e*
3·1523 *ns*	3·2423 *ns*	3·2523 *ns*	3·2623 *ns*	3·2723 *ns*	322243 *ns*
322253 *ns*	322263 *ns*	322273 *ns*	3·1·64 *e*	3·1·77 *j*	4···10 *e*
4··120 *fi*	4··210 *fi*	4·1·20 *fi*	4·1120 *fo*	4·1210 *fo*	4·1220 *p*
4·2110 *fo*	4·2120 *p*	4·2210 *p*	4·3120 *pg*	4···21 *xr*	4··221 *c*
4·2·21 *pe*	4·2221 *pe*	4·2231 *pe*	4·2321 *pe*	4·2331 *pe*	4·3221 *pe*
4·3231 *pe*	4·3321 *pe*	4·3331 *pe*	5···10 *i*	5··120 *fi*	5··210 *fi*
5·1·20 *fi*	5·1120 *fo*	5·1210 *fo*	5·1220 *p*	5·2110 *fo*	5·2120 *p*
5·2210 *p*	5·3120 *pg*	5···21 *xl*	5··221 *c*	5·2·21 *pe*	5·2·31 *pe*
5·2221 *pe*	5·2231 *pe*	5·2321 *pe*	5·2331 *pe*	5·3221 *pe*	5·3231 *pe*
5·3321 *pe*	5·3331 *pe*	6···10 *sh*	6··110 *sh*	6··120 *fi*	6··210 *fi*
6·1·10 *sh*	6·1·20 *fi*	6·1110 *i*	6·1120 *fo*	6·1210 *fo*	6·1220 *p*
6·2110 *fo*	6·2120 *p*	6·2210 *p*	6·2230 *pe*	6·2320 *pe*	6·2330 *pe*
6·3120 *pg*	6·3220 *pe*	6·3230 *pe*	6·3320 *pe*	6·3330 *pe*	612320 *s*
6····1 *sh*	6···21 *sh*	6··221 *c*	6·2·21 *pe*	6·2221 *sh*	7···10 *j*
7·1120 *fo*	7·1210 *fo*	7·1220 *p*	7·2110 *fo*	7·2120 *p*	7·2210 *p*
7·2230 *pe*	7·2320 *pe*	7·2330 *pe*	7·3120 *pg*	7·3130 *pe*	7·3220 *pe*
7·3230 *pe*	7·3320 *pe*	7·3330 *pe*	712220 *s*	7···21 *pe*	7··221 *c*
7·2·21 *pe*	7·2221 *x*				

Chapter 5

Components

5.1 Introductory Remark

In this chapter we consider various configurations called *components*, which play an important role in the design of the universal computer-constructor (UCC). These components together with the signal sequences and their corresponding operations will enable us to escape from the complexities of the transition function.

To understand the operation of these components little more than the following information is required:

1. The behavioral properties of gates (see Section 4.15).
2. The direct relationship between path length and time—if two paths branch out from a common point A to points B, C respectively, and if path length AB is strictly greater than path length AC, then a signal which duplicates at A will arrive at B strictly later than at C.

5.2 Notation and Convention

To get away from the details of path structure and gate structure we represent a sheathed path by a line, and a gate by a large arrowhead as in Fig. 5.1. An attempt has been made to represent certain essential

inequalities in path length by showing some paths in the diagrams as obviously longer than others. These inequalities are made explicit in the text when it is felt necessary to do so.

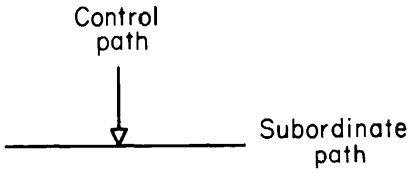


FIG. 5.1. Symbol for gate.

We adopt the convention that all gates are initially off unless explicitly stated otherwise.

5.3 Permanent One-Way Lock

The main purpose of the permanent one-way lock is to obtain one-way paths. Figure 5.2 shows a path *ACEDB*, which is required to carry signals from *A* to *B* but not in the reverse direction.

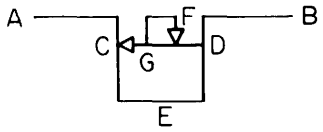


FIG. 5.2. Permanent one-way lock.

The first signal to enter must be the activating signal 07. It can enter either at *A* or *B*. Assume it enters at *A*. When it reaches *D* it duplicates, one copy proceeding on to *B*, the other turns on gates *C*, *F*.

These gates, once turned on, cannot be turned off, since *F* prevents any signal from gaining access to the control paths. With *C* permanently on, any signal proceeding from *B* toward *A* will be annihilated at *C*. Any signal proceeding in the reverse direction is, however, allowed to pass.

Now assume the first signal (07) to enter does so at *B*. Upon reaching *D* it duplicates, one copy proceeding toward *E* and the other turning on

gates C, F. Path DEC is longer than DGC and, as a result, gate C gets turned on before the signal arrives from D via E. Consequently, this signal is annihilated at C and $ACEDB$ is established as a one-way path.

One application of the one-way lock is shown in Fig. 5.3. The gate G is operated by signals from any of the paths $A, B, \ldots, Z$.

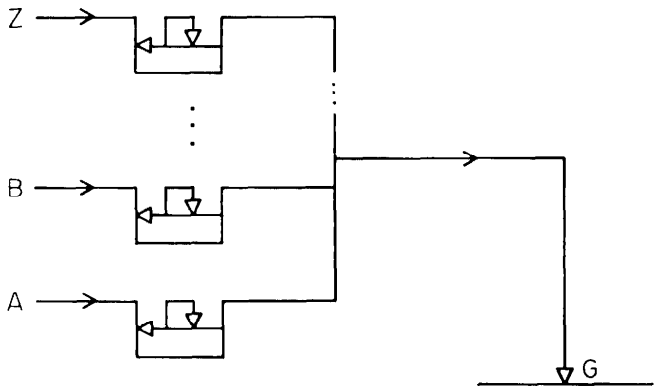


FIG. 5.3. Gate with multiple inputs.

5.4 Permanent Two-Way Lock

This component permits the first signal (which must be 07) to pass through it in one direction or the other (not both). Thereafter, the path is permanently blocked to all types of signals.

Its mode of operation is quite simple. Referring to Fig. 5.4, the first signal passes through, since gates E, F are off, but duplication at C, D

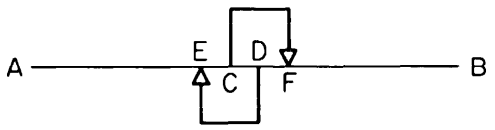


FIG. 5.4. Permanent two-way lock.

yields copies which turn both these gates on. Henceforth, the control paths for E, F are inaccessible.

Figure 5.5 shows a component which could easily be mistaken for a two-way lock. Careful examination reveals that its behavior is quite different. We shall have no further use for it.

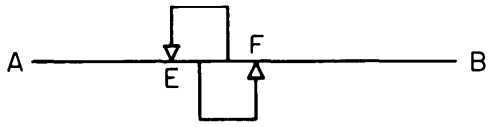


FIG. 5.5. Alternating two-way lock.

5.5 Subordinate-Restored Gate

Suppose a gate F (see Fig. 5.6) is normally on. A signal is sent to F over the control path C to turn it off. Suppose we require the first signal traversing the subordinate path to restore F to its normal state.

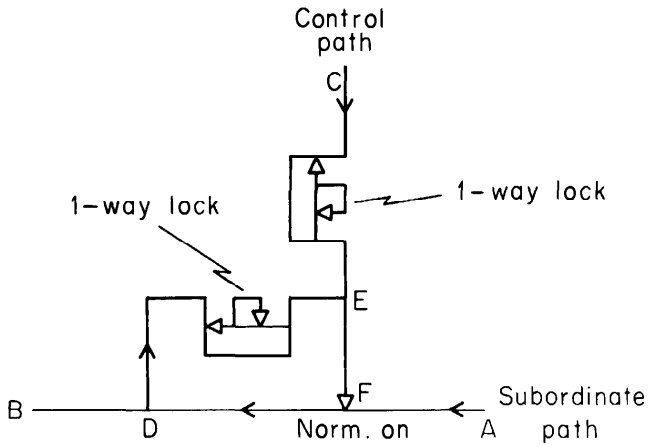


FIG. 5.6. Subordinate-restored gate.

Then, if we restrict the signals so as to be 07, the subordinate-restored gate fulfills this function.

If a sequence of 07 signals is arriving at A and only one of the sequence is to be allowed to pass, then the path DEF must be shorter than the distance between successive signals.

5.6 Periodic Emitter

The periodic emitter converts a single input signal 07 into copies which are emitted indefinitely with a fixed frequency. With the design

shown in Fig. 5.7 the smallest period obtainable is ten time steps, but with a more elaborate design it can be made as small as four time steps (the minimum distance between the signal states of successive signals permitted by the transition function f is four units).

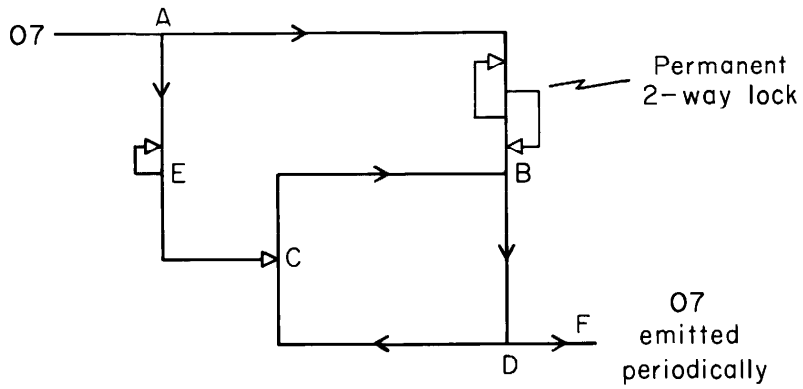


FIG. 5.7. Periodic emitter.

The single input at A duplicates, one copy going to the gate C and the other to the next junction B. Duplication again occurs at B, one copy going to C (on the subordinate path) and the other going to the next junction D. Because path AEC is shorter than path ABC, the signal which proceeds from B to C finds gate C already turned on and is therefore annihilated. As a result the signal proceeding from D clockwise around the loop $DCBD$ does not encounter and collide with a signal proceeding in the opposite direction. Every time the clockwise-circulating signal reaches D, duplication gives rise to a copy which is output at F—hence the periodic output.

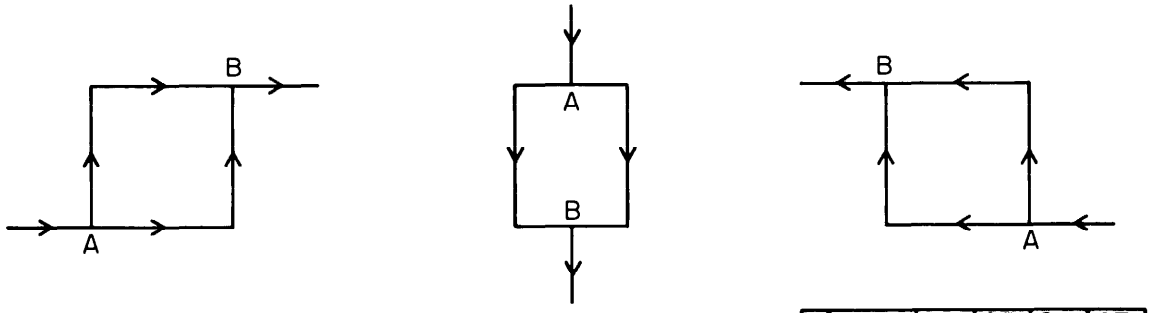
Duplication also occurs every time this circulating signal reaches B and a copy proceeds toward A. However, it is promptly annihilated by the two-way lock between B and A.

5.7 Signal Transformer Type 456

It was observed in Section 4.12 that, if two copies of the same type of signal arrive simultaneously at a T-junction, a product signal is formed (see Table 4.4). A simple way to realize this simultaneous arrival is shown in Fig. 5.8.

The input signal duplicates at *A*. The resulting copies traverse paths of equal length to *B*. Thus their product is output from *B*.

Note that the only outputs possible from this type of transformer are 04, 05, and 06. In contrast, we now consider a transformer whose only output is 07.



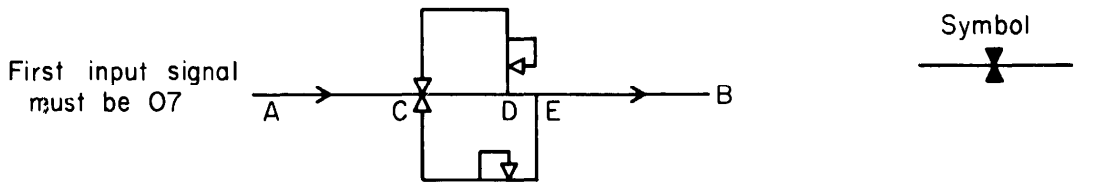
Input	04	05	06	07
Output	05	06	06	04

FIG. 5.8. Signal transformers type 456.

5.8 Signal Transformer Type 7

Suppose we require to have a gate operated when a certain signal arrives, but we do not know in advance whether it will be 04, 05, 06, or 07. The transition function does not provide for 04, 05, 06 turning gates on; nor does it provide for 04, 05 turning gates off. The signal transformer provides a means of converting the signal, whatever its type, to 07, thus ensuring that the gate will be operated.

The first signal input at *A* (see Fig. 5.9) must be 07 in order to get the two gates at *C* turned on. Once turned on these gates stay on permanently due to the locking loops. Thereafter, any signal arriving at *C* from *A* is converted to 07 (see Section 4.15).



Input	04	05	06	07
Output	07	07	07	07

FIG. 5.9. Signal transformer type 7.

5.9 Crossover for Two Unidirectional Paths

Whenever it is required to get one signal from a point a to another A and a second signal from a point b to another B and the straight line aA intersects the straight line bB, the paths for these two signals must intersect (the space is only 2-dimensional). The intersection must be realized by means of two T-junctions, because the transition function does not provide for direct X-crossings. Additional gates are needed to prevent an input signal at one point from becoming an output at all three remaining points.

Figure 5.10 shows a crossover suitable for two unidirectional paths. It does not, however, permit signals to cross one another concurrently. For the purpose of a demonstration of universality we can ignore questions of speed or other types of efficiency in the design of cellular components.

The first input signal at a and the first at b must be 07 in order that the one-way locks and signal transformers type 7 be properly set initially.

Suppose a signal enters at a some time after the initializing. It duplicates at C, again at D, and a third time at H. The copy proceeding

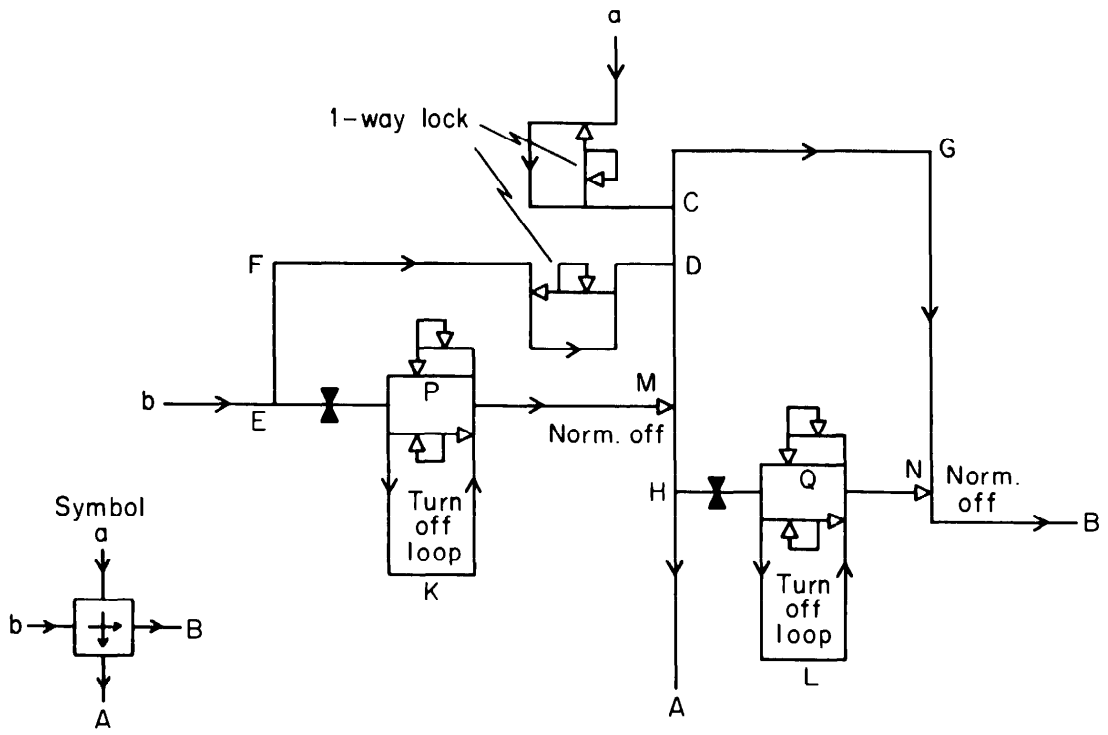


FIG. 5.10. Crossover for two unidirectional paths.

toward F is annihilated by the one-way lock between D and F. As a result M does not get turned on. One copy from H proceeds out on path A as desired. The other turns gate N on before the copy proceeding along CGN reaches N. Accordingly, this copy is annihilated upon reaching N and no output occurs at B. Gate N is turned back off by a copy (transformed into 07 by the signal transformer type 7), which proceeds around the turn-off loop L.

For correct operation the following inequalites must be satisfied:

$$CHLN > CGN > CHQN$$
$$EKM > EFDM > EPM.$$

Clearly, these inequalities are satisfiable.

5.10 Crossover for One Bidirectional and One Unidirectional Path

This component (shown in Fig. 5.11) makes use of two crossovers of the type described in the previous section. The bidirectional path BB is split into two unidirectional paths. Gates are required in addition to those for the one-way locks in order to prevent signals from circulating around the loop formed by the twofold split of path BB.

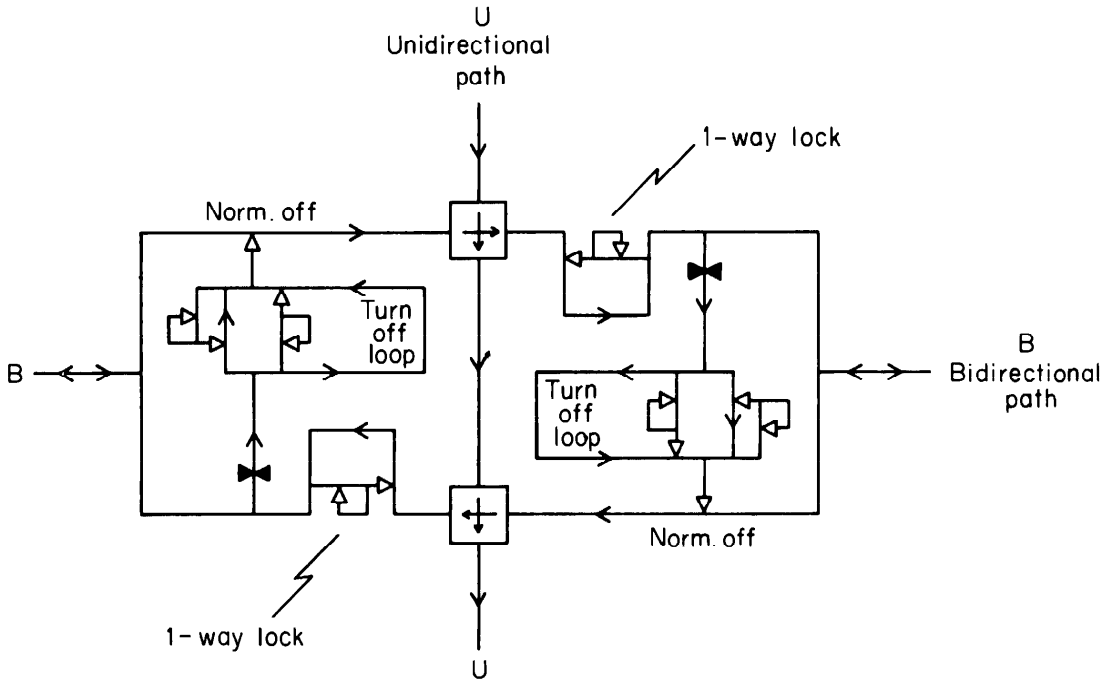


FIG. 5.11. Crossover for one bidirectional and one unidirectional path.

5.11 Echo Switch

The purpose of this component is to provide switching to enable signals of all types to go from the signal source *U* (see Fig. 5.12) out on the bidirectional path *V* (but not on *W*) and to enable the echo signal

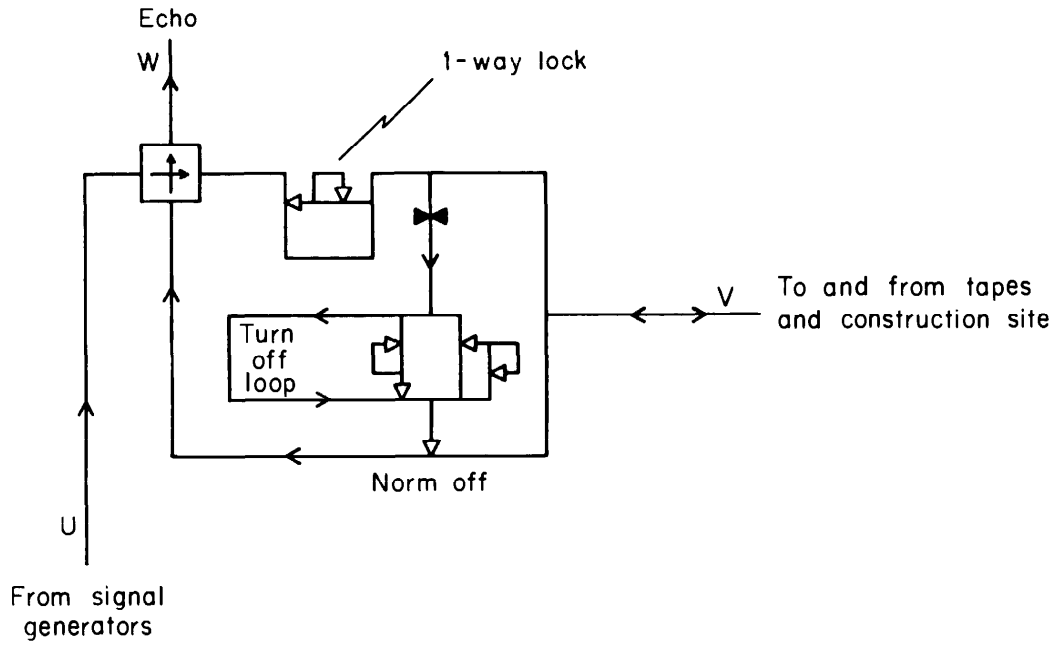


FIG. 5.12. Echo switch.

(06 or 07) to return on path *V* and thence proceed on *W* (but not on *U*). Its design follows the pattern of previous components. For correct operation the first signal out and the first echo back must be 07.

5.12 Echo Discriminator

The echo discriminator converts the echo signal from its original form (06 or 07 on a single path) into a more useful form (07 on path *A* or 07 on path *B* respectively—see Fig. 5.13). It also produces the *echo receipt signal* (07 on path *C*) whenever an echo is received, regardless of its type.

The net effect of the first signal from the periodic emitter (which is required to precede all other inputs to this component) is to turn on gate L, the two one-way locks connected at R and the two-way lock between P and R. When an echo signal arrives, it is directed to gates L, M. The gate M being normally off is turned on if and only if the echo is 07. The gate L, on the other hand, is turned off by the echo whether

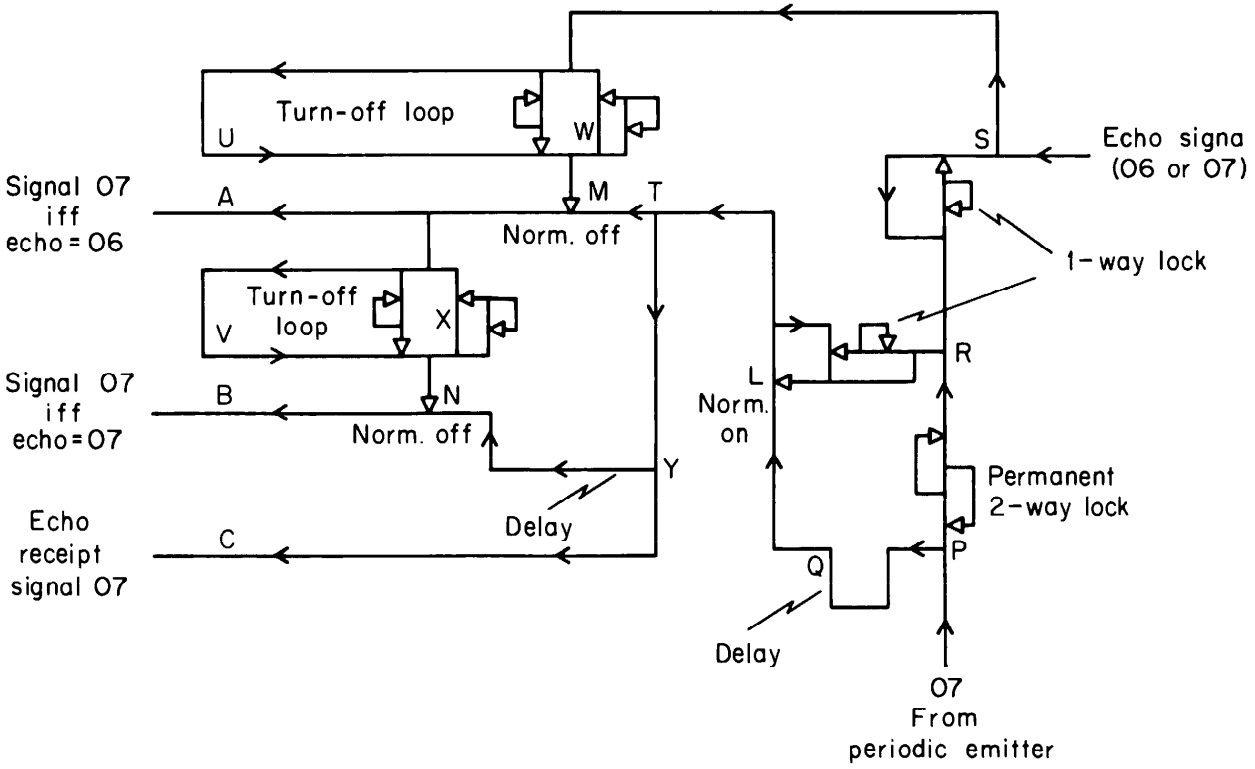


FIG. 5.13. Echo discriminator.

it is 06 or 07. L is subordinate-restored and permits only one signal to pass from the periodic emitter. This single signal is output unconditionally at C and, in addition, on either path A or path B according as gate M is off or on. A second gate N is used to facilitate this process and both M and N are restored as necessary by the turnoff loops U, V respectively.

For correct operation certain inequalities must be satisfied:

$$SUM > SRL + d + LTM > SWM$$
$$TVN > TYN > TXN$$

where d is the period of the periodic emitter.

5.13 Decoder

In the design of the universal computer-constructor given in Chapter 6, a decoder with the following properties is used:

1. It has two echo input paths, one labeled "07 *iff echo* = 06" (we can regard this path as the *zero* input), the other labeled "07 *iff echo* = 07" (we can regard this path as the *one* input).
2. It can store four successive inputs (*zero* or *one*).
3. Upon receipt of a read-out signal 07 over a third input path, it routes this signal out over a path f_i ($i = 0, 1, \ldots, 15$) where the 4-bit input is the standard binary code for i; furthermore, the decoder is returned to its original state and is therefore ready for the next 4-bit input.

Such a decoder is shown in Fig. 5.14. All gates shown must be turned on to get the decoder into its normal state. Associated with each arrowhead

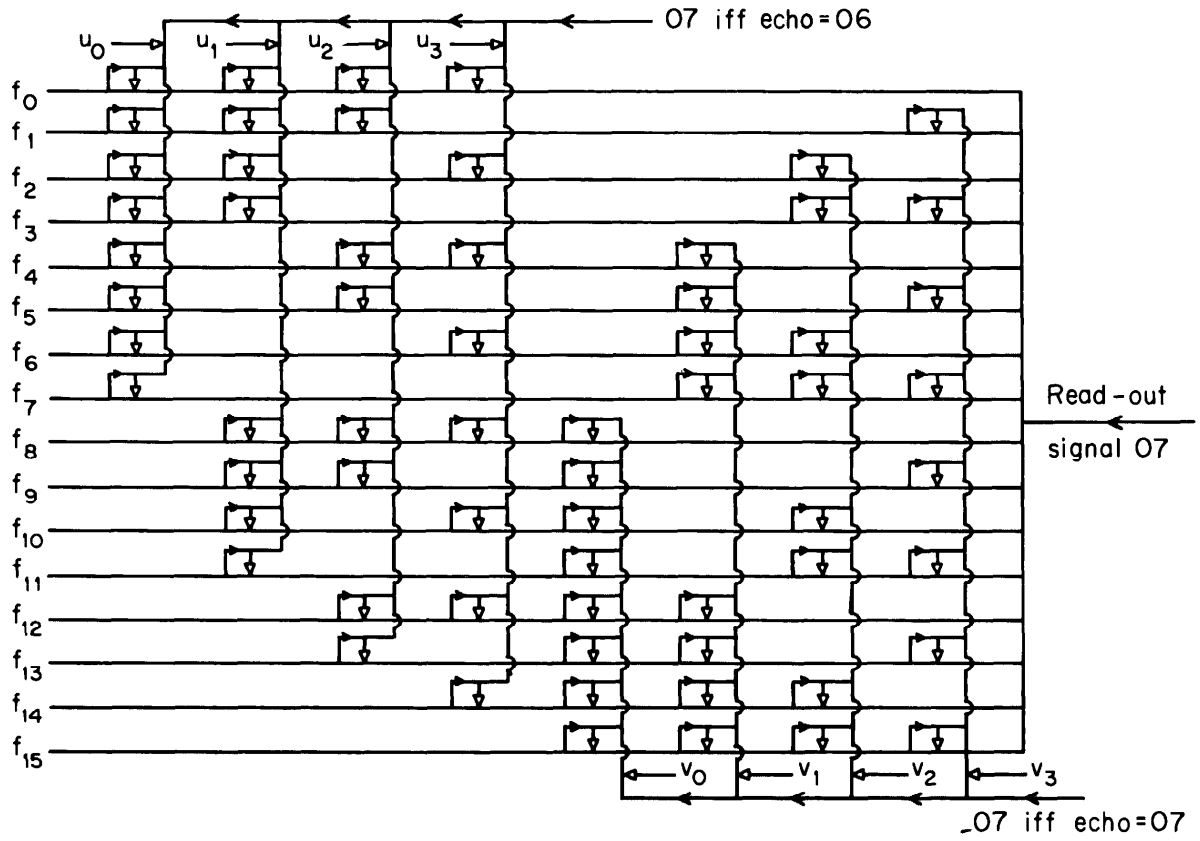


FIG. 5.14. Decoder for 4-bit operation code.

of conventional type (i.e., all except those representing gates) is a one-way lock.

The gates $u_0, u_1, u_2, u_3, v_0, v_1, v_2, v_3$ are operated from a remote source as follows: the pair u_i, v_i are turned off to permit read-in for bit i ($i = 0, 1, 2, 3$) and are then turned back on again.

Note that the read-out signal must restore each column of gates which has been turned on. Accordingly, the vertical paths in the diagram are bidirectional.

Chapter 6

A Self-Reproducing Universal Computer-Constructor

6.1 Objectives

In this chapter we reach the final step of demonstrating the universality of the 8-state, 5-neighbor cellular space defined in Chapter 4. This step consists of exhibiting a universal computer-constructor (UCC) embedded in the cellular space.

To be more specific, let P^* denote the set of all configurations[1] over (0, 1) other than those which intersect the finite area occupied by the machine. We require a single machine which, when supplied in each case with a suitable program tape, has the following capabilities:

1. It can construct, read, and erase any member of P^*.
2. It can compute any Turing-computable function from P^* into P^*.
3. It can construct and activate a copy of itself at any location which is sufficiently clear of the constructor itself.

In designing the particular UCC to be described below the major guiding factor was simplicity of description. Thus, features which would have contributed to speed of operation, programmer convenience, or small size were rejected if, in our judgment, they complicated the task of communicating an understanding of the machine to human beings.

[1] By definition a configuration has finite support.

One result of this philosophy is that few operations in the machine are carried out concurrently, in spite of the many opportunities.

While the underlying cellular space is synchronous in the sense that every cell changes state simultaneously, the UCC is essentially asynchronous. In general, the operations take an amount of time which is dependent on the distance to the end of the relevant path, and some of these distances vary with time. Moreover, the basic component for logical operations—the path gate—has the latch property: i.e., it can store information indefinitely. Thus, internal operations in the UCC can proceed without deadlines.

6.2 Programmer's View of UCC

The UCC commands available to the programmer enable him to control the movements (extension and retraction) of two paths, one associated with a data tape D, the other associated with a construction site C (see Fig. 6.1).[1]

The data tape is a linear string of cells, infinite on the right. Its primary use is computational. Thus, if the machine is to be used as a Turing machine with a 1-dimensional tape, path C can be ignored, initial data would be placed on tape D, and final results would appear there.

On the other hand, if the machine is to be used for constructing only, then it is possible to ignore path D, providing the program is tailored precisely to the structure to be built. For more general construction it would be desirable to use tape D at least for storing the specifications of the structure to be built and possibly for the results of intermediate computations as well.

Another use of C and D in combination occurs in the generation of specifications of a structure from an actual copy located in some part of the construction region. In this case, D can be used for whatever (computable) transformation is required on the information read from the construction site, as well as for storing the final result.

[1] The labels C, D will also be associated with the corresponding paths and with the gates which control transmission along these paths (we shall leave the context to resolve the resulting ambiguity).

We shall see that the only means of intercommunication between paths C, D is through the *conditional transfer* command and that this is perfectly adequate for this universality demonstration.

Path C is, of course, the appropriate path for transmission of signals to sheathe and activate a completed structure. A structure which is to receive such signals in a meaningful way must have an injection receiver located in an accessible position (and may have several such receivers).

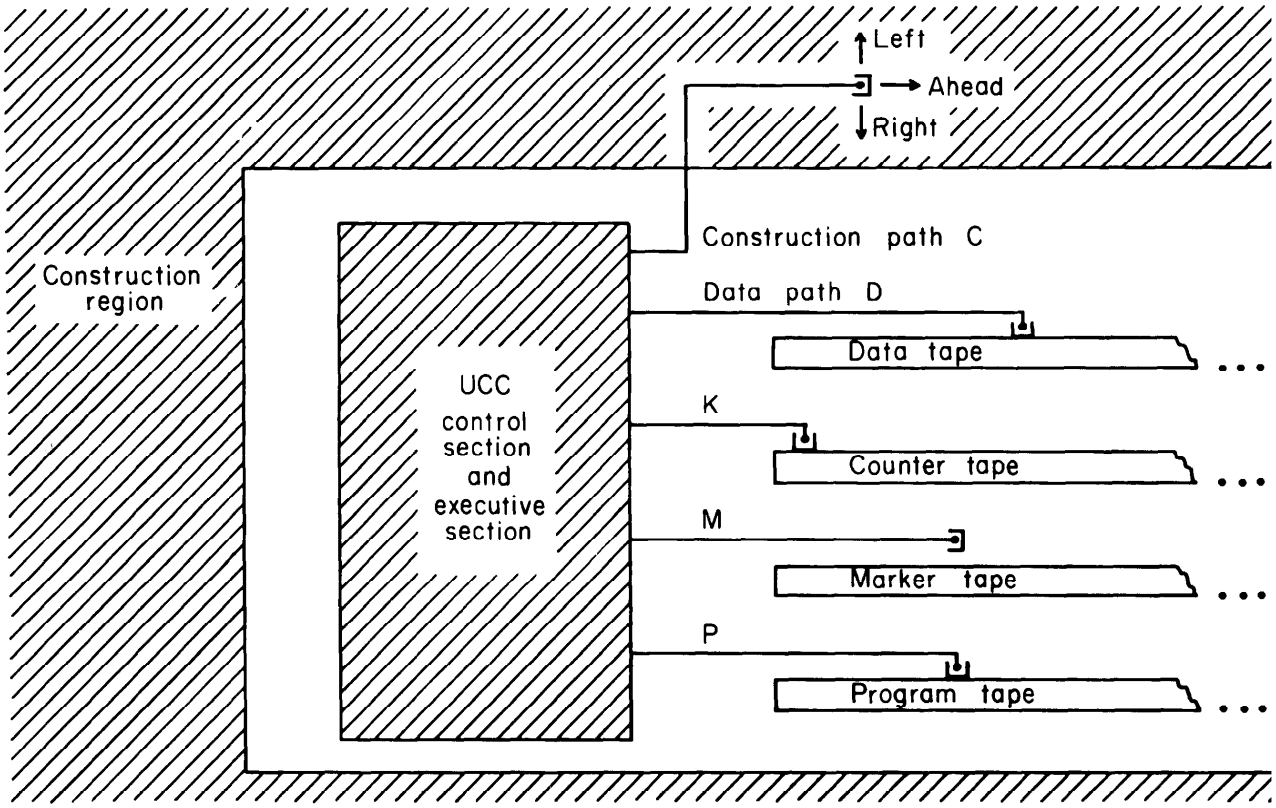


FIG. 6.1. The universal computer-constructor and its construction region.

Now we can turn to the UCC commands in detail. Their absolute and symbolic codes together with abbreviated names are listed in Table 6.1.

The command z needs no further explanation at this point. The path selection commands c, d have an alternating property in that the first execution of c selects path C, while the second execution blocks that path, and so on; a similar remark applies to the command d and the path D. It is possible to have both paths selected concurrently and this is certainly legitimate if any of the commands other than $t(k)$, i, j are to be issued during their simultaneous selection. The issuing of $t(k)$, i, j in

such circumstances may lead to the development of neighborhood states for which f is undefined (note that by definition a linear tape does not possess an injection receiver).

The UCC commands x, xl, xr, r, rl, rr, m, e, i, j produce precisely the same effect as their signal sequence counterparts and do so on whatever paths (C or D or both) are selected. We shall continue, however, to distinguish between UCC commands on the one hand and signal sequences on the other for reasons which will become apparent later.

TABLE 6.1
COMMANDS FOR UNIVERSAL COMPUTER-CONSTRUCTOR

Absolute code	Symbolic code	Operation
0	z	*stop*
1	c	*select or block construction path*
2	d	*select or block data path*
3		
4	x	*extend*
5	xl	*extend left*
6	xr	*extend right*
7	r	*retract*
8	rl	*retract left*
9	rr	*retract right*
10	m	*mark if zero*
11	e	*erase if one*
12	$t(k)$	*transfer on mark*
13		
14	i	*sheathe* (*inject signal* 06)
15	j	*activate* (*inject signal* 07)

The command $t(k)$ has no counterpart in the list of signal sequences listed in Chapter 4. Similarly, the signal sequences cs, sw have no corresponding UCC commands. Actually, cs and sw are used many times over along with many of the other signal sequences to implement the conditional transfer $t(k)$.

The command $t(k)$ is defined in the same way as Wang's [7] conditional transfer, except that it applies to the selected path (C or D, but not both). In other words, if a mark is detected under the cap of the

appropriate path, control is transferred to the command with identifying number k in the program; otherwise, control proceeds to the next instruction in the normal sequence.

Correct operation of the UCC depends on adherence to the following convention (at least, in the absolute or compiled version of the program): the first command in the program is assigned the value zero for its identifying number k; each successive control entry point is assigned a value one larger; no other commands possess identifying numbers. Thus, $t(k)$, when executed and if a mark is detected, gives rise to a transfer of control to the kth control entry point beyond the first command in the program.

We now proceed to show that the UCC commands form a complete set with respect to the objectives outlined above. Although this set is quite rudimentary, it is not minimal: for example, changes in the data tape and at the construction site could be effected by means of only one path—whenever necessary, this path would be retracted from the tape and extended to the site, or vice versa.

6.3 Universality of UCC Command Set

The set of commands,

$$d,\ x,\ xr,\ r,\ rr,\ m,\ e,\ t(k),\ z$$

is complete with respect to computation on a 1-dimensional tape because the Wang set can be synthesized from it. First we note that the conditional transfer $t(k)$ is identical to the Wang conditional transfer. The remaining Wang commands and their behaviorally equivalent sequences of UCC commands are listed below:

Wang command	UCC sequence
mark	$t(k),\ m,\ k$:
erase	$t(k),\ m,\ k$: $e,\ x$
shift left	$r,\ rr,\ r,\ xr,\ x$
shift right	$r,\ rr,\ x,\ xr,\ x$

where k denotes the identifying number for the UCC command that follows it and the parentheses, period, and colon are used as punctuation marks to distinguish such an identifying number from actual commands (symbolic or absolute).

Assume that the construction region is initially entirely quiescent. Clearly, the set of commands,

$$c,\ x,\ xl,\ xr$$

permits the extension of the construction path to an arbitrary location in the construction region, and the set,

$$r,\ rl,\ rr$$

permits its retraction to its original position. By use of the command m, when the construction path has been extended to some location, the state of the cell under the path cap can be changed to 1.

To complete the demonstration that an arbitrary configuration[1] over (0, 1) can be constructed, we need only observe that accessibility by the constructing path to the cells which still need marking can be maintained throughout the construction process if this process is properly ordered. For example, one can treat the construction site as a rectangular array and perform the construction row by row, beginning with the row farthest from the constructor and ending with the row nearest to it.

A known (0, 1) configuration in a known location can similarly be destroyed by use of the command e for precisely those cells initially in state 1.

An unknown (0, 1) configuration in a known area can be read destructively using both $t(k)$ and e, the erasure being necessary in order to gain access by the constructing path to cells not on the perimeter of the configuration.

We have now shown that a configuration, which can interpret and execute the UCC commands, can construct, read, and erase any member of P^*. It can also compute any Turing-computable function defined on the set of all 1-dimensional tapes. The ability to compute any Turing-computable function defined on P^* now follows from the observation that any bit sensed in the construction region by means of

[1] By definition a configuration has finite support.

the command $t(k)$ can be copied on to the 1-dimensional tape D—or vice versa—through selection by $t(k)$ of appropriate subprograms.

Thus, when we have exhibited the UCC and shown that it can, in fact, interpret and execute the UCC commands, there will be but one property left to demonstrate: namely, its ability to reproduce itself (see Section 6.13). In passing, we observe that we could contemplate a variety of higher-level languages which would possess the same computation and construction universality as the UCC commands and which would be more convenient for the programmer. Consideration of such languages would not be relevant to our present goals.

6.4 Structure of UCC

For convenience in exposition we shall divide the UCC (see Fig. 6.2) into three major sections:

1. The *memory section* containing the tapes D, K, M, P.
2. The *executive section* (section E, for short) which handles most of the signal sending, receiving, and storage.
3. The *control section* which provides appropriate sequencing for activities in the executive section.

The memory section communicates with section E via a bidirectional path, which terminates at the echo switch in the E section and at the path ends C, D, K, M, P in the memory section. All of the information passed from the memory section to section E is transmitted in the form of echo signals. All of the information passed in the reverse direction is transmitted in the form of the signal sequences already described.

We shall refer to gates B, C, D, K, M, P, which are associated with the paths to the tapes and construction region as the memory gates. Gates A, E, F, H, which are centrally located in section E, will be called the *central gates*, and gates t_i, u_i, v_i the *echo destination gates*.

The links between section E and the control section are unidirectional paths carrying isolated 07 signals which ultimately operate the central gates, echo destination gates, and the memory gates. (There are

no gates in the control section other than for crossover and one-way locks.) Because these links are quite numerous, they are exhibited in the UCC block diagram by means of a label identification technique. Thus, the outputs s_0, e_i, f_i, p_i, q_i from section E should be identified with the

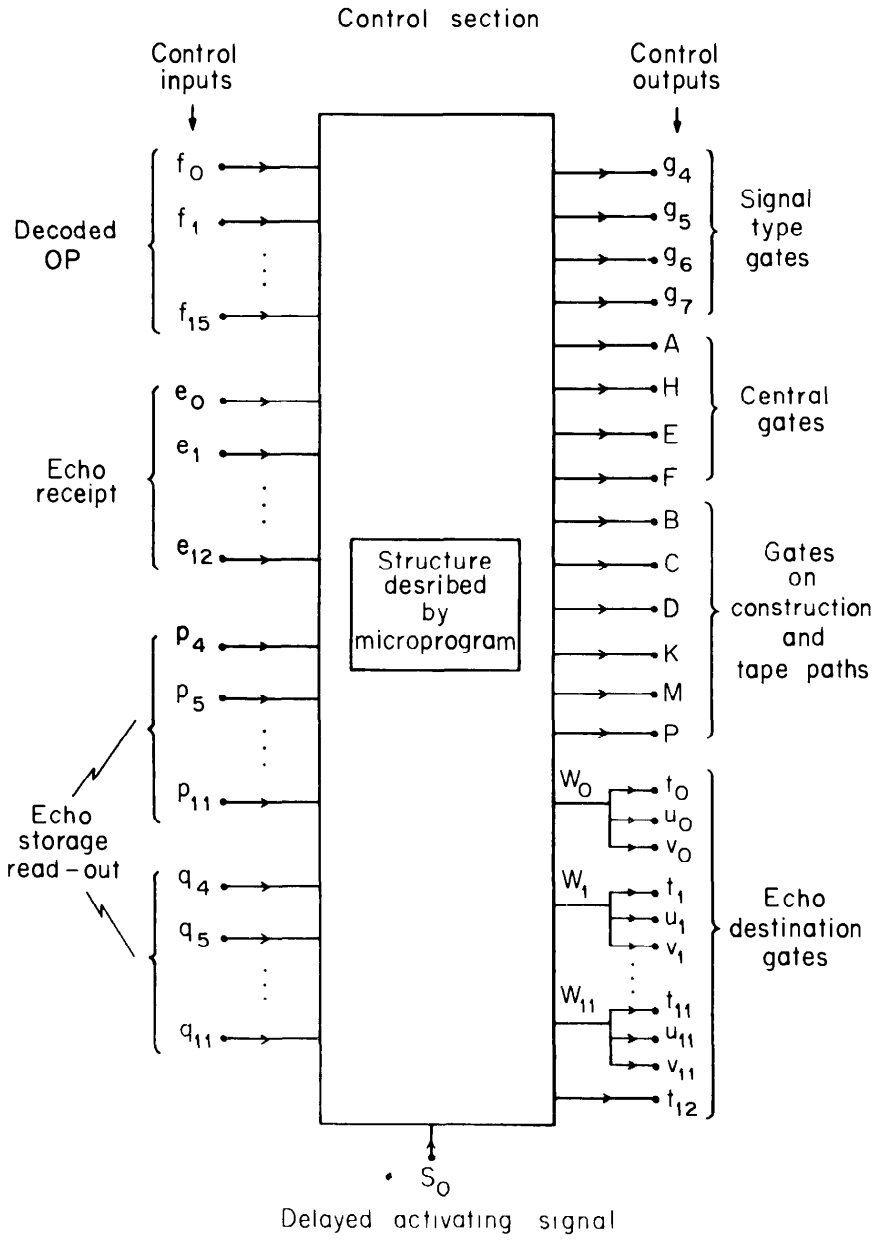


FIG. 6.2. Block diagram of UCC (part 1).

similarly labeled inputs to the control section, and the outputs g_4, g_5, g_6, g_7, A, H, E, F, B, C, D, K, M, P, t_i, u_i, v_i from the control section should be identified with the similarly labeled inputs to section E and the memory section.

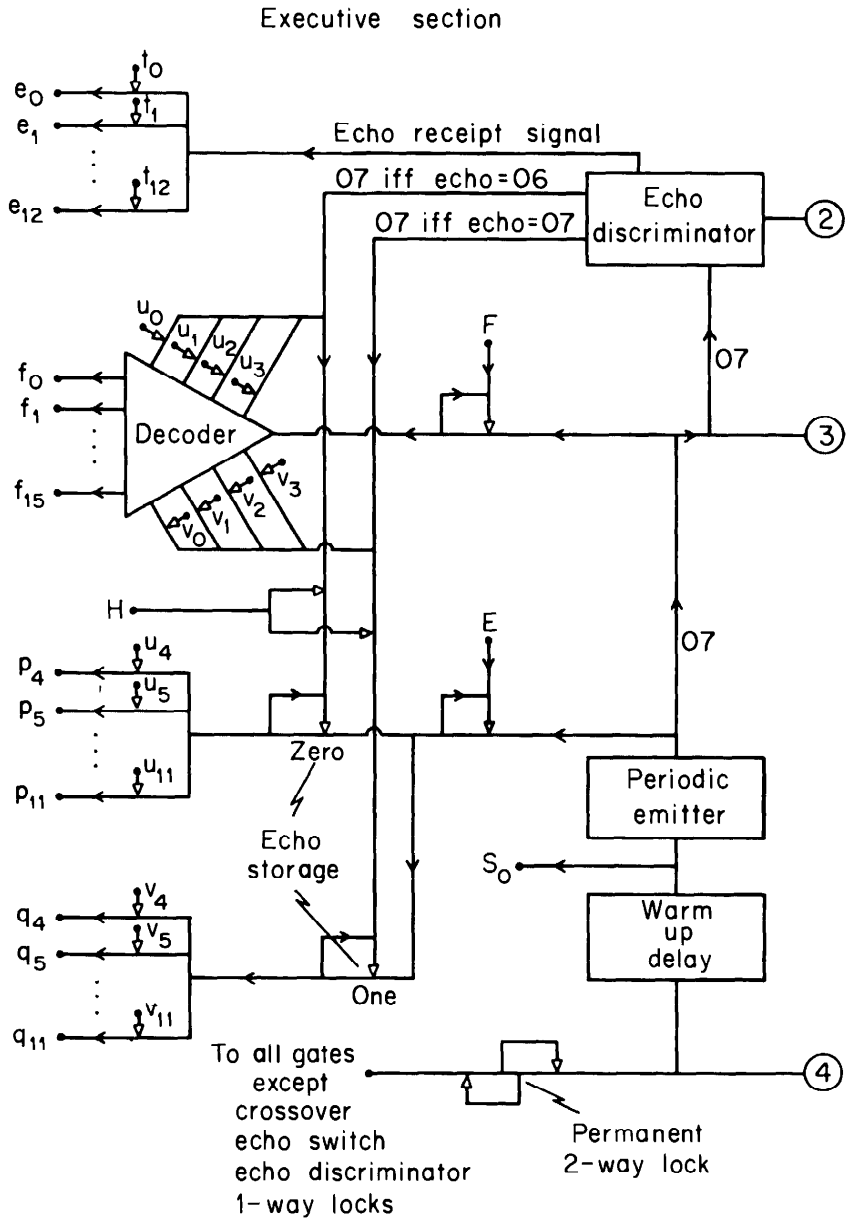


FIG. 6.2. Block diagram of UCC (part 2).

6.5 Memory Section

The program tape P is intended to provide read-only memory for a UCC program of arbitrary length. All the commands of the program are placed on this tape, beginning at the *base cell* (see Fig. 6.2, part 4). Marks to identify control entry points are omitted.

These marks are placed on a separate tape, the marker tape M, because this results in a considerable simplification in the format of P

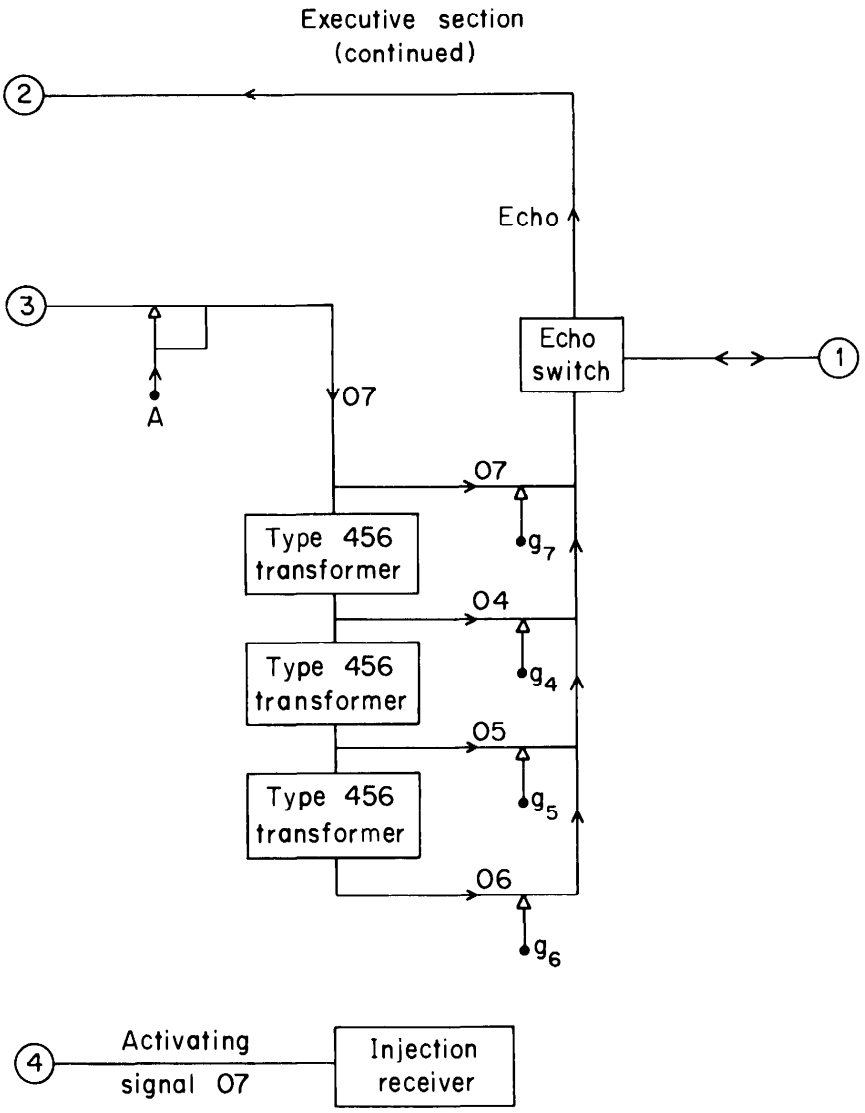


FIG. 6.2. Block diagram of UCC (part 3).

and, hence, in the interpreting organization within the executive section of the machine.

Except for the conditional transfer $t(k)$ a UCC command consists of a 4-bit operation code. The conditional transfer consists of the appropriate 4-bit operation code immediately followed by a string of k ones which is terminated by a single zero for punctuation. The variable k is unbounded and its value for any conditional transfer is the identifying number of the command to which control is to be transferred when the condition is satisfied.

The placement of marks on M can be defined as follows: if the nth

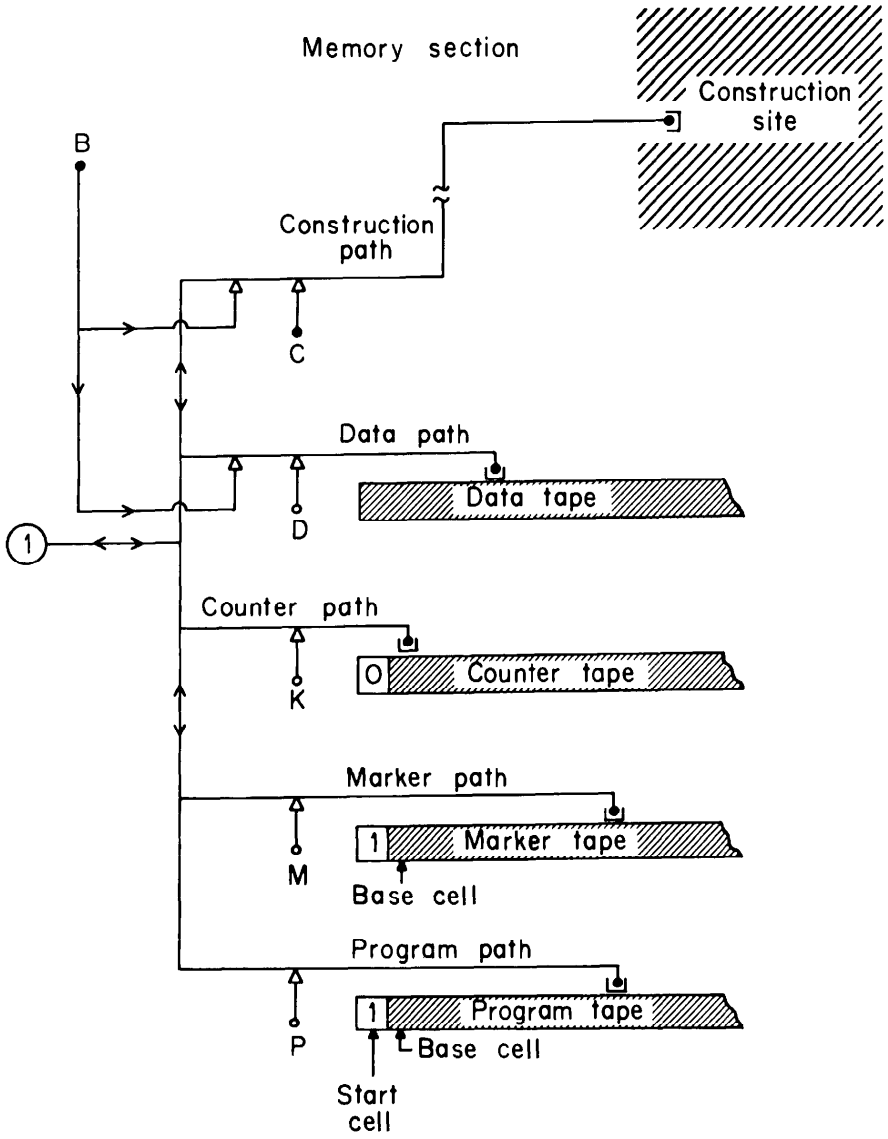


FIG. 6.2. Block diagram of UCC (part 4).

bit[1] on the tape P is the high-order bit of the 4-bit code for a command which happens to be a control entry point for the program, then the nth bit on tape M is a one; otherwise, all bits on M are zero with the exception of the bit in the cell preceding the base cell, which is set to 1 in order to identify the start of tape M.

The counter tape K is used solely for the execution of conditional transfers for which a mark is detected. In such cases the identifying number k is copied on to tape K where it is used to control the search for the kth identifying mark on M and the corresponding command on P. The details of this procedure are given in Section 6.11.

[1] By convention, $n = 0$ for the base cell and the state of the base cell in M is 1.

6.6 Executive Section

The functions of this section are to generate appropriate sequences of signals (04, 05, 06, 07) to route these signals out on to appropriate paths (C, D, K, M, P), to receive and store echo signals, and to read out the stored signals to the control section upon request by that section. The sequence in which all these activities takes place is determined by signals from the control section.

We shall postpone consideration of the start-up procedure for the machine and assume for the present that all the gates explicitly mentioned in the UCC diagram (Fig. 6.2) have been turned on. In addition, the periodic emitter is emitting the signal 07 every n time steps, where n is fixed.

Suppose we wish to send signal 04 out on path P. This may be accomplished by turning gates P, g_4, A off, providing some care is taken about the timing of these events relative to one another. Thus, we must ensure that once a signal 07 has been allowed to pass gate A,

1. The immediately following signals from the periodic emitter are not allowed to pass A.
2. By the time the released signal has been transformed to 04 and has reached gate g_4 this gate has been turned off and will accordingly allow the signal to pass.
3. By the time signal 04 reaches the gate P this too has been turned off and will allow the signal to pass.

To meet requirement (1) gate A is provided with subordinate signal restoration (note the feedback loop from the subordinate path to the control path). To meet requirements (2) and (3) we adopt a very conservative policy: namely, that the memory gate (in this example P) must be turned off first, the signal type gate (in this example g_4) next, and gate A last. As might be expected this policy leads to a simpler control section than would otherwise be the case.

Further details of the routing of signals out onto paths C, D, K, M, P are given in subsequent sections.

The receipt of echo signals is handled by the echo switch and the echo discriminator. The former component merely routes the echo

signal on to the echo discriminator and, in so doing, protects the signal generating components. The discriminator generates the echo receipt signal for transmission to the control section and converts the echo into a form suitable for storage.

The storage of echo signals is handled by the decoder—in case a 4-bit operation code is being read from the program tape—and by the gates labeled echo storage—in case any other bit is being sensed on any of the paths *C*, *D*, *K*, *M*, *P*. Both the decoder and the echo storage gates hold their information until read out, at which time they are restored to their normally on state by the read-out signal.

The decoder is read out whenever gate *F* is turned off. The output signal proceeds to the control section over path f_i, where i is determined by the four bits just read in. Echo storage is read out whenever gate *E* is turned off, and in this case the output signal proceeds to the control section over path p_i or q_i according as a zero or one was stored, where i is determined by previous gating (u_i, v_i) put into effect by the control section.

6.7 Control Section

The control section consists essentially of nothing but a collection of paths connecting its inputs to its outputs. Why then is it treated as a separate section? There are two important reasons. First, the connections are too numerous to represent them all clearly in a block diagram. Second, they must satisfy certain delay constraints in combination with paths in the memory and *E* sections.

By separating the control section it becomes possible to express both the connections and the constraints which they must satisfy in a very simple form. A language even more primitive than the UCC command language is used for this purpose. We shall refer to this language as the *microlanguage*, to its primitive terms as *microcommands*, and to the expression in that language which defines the control section as the *microprogram*.

It will be found that the microprogram together with the UCC block diagram gives both a concise and clear description of the internal operation of the UCC as a whole. One reason for this is that it enables us to dispense with many details of geometric layout.

6.8 The Microlanguage

The primitive terms in this language are the labels of all inputs to and outputs from the control section. A statement in the language has the form

$$i: \quad j_1, j_2, \ldots, j_n$$

where i is the label of an input to the control section or of an internal connection within this section, and j_r $(r = 1, 2, \ldots, n)$ are labels of outputs from the control section or of internal connections within this section.

The *structural interpretation* of this statement is as follows: path i (a control input) is successively connected to paths j_r $(r = 1, 2, \ldots, n)$ in such a way that each successive connection entails an additional delay of y time steps as indicated in Fig. 6.3 (see the enlarged view of step i).

The value of y is fixed and is determined as follows. The layout of the memory section and section E is planned in detail (i.e., specific cells are assigned to the various paths and components). Next, the layout of the control section is determined to the same level, except that the loops

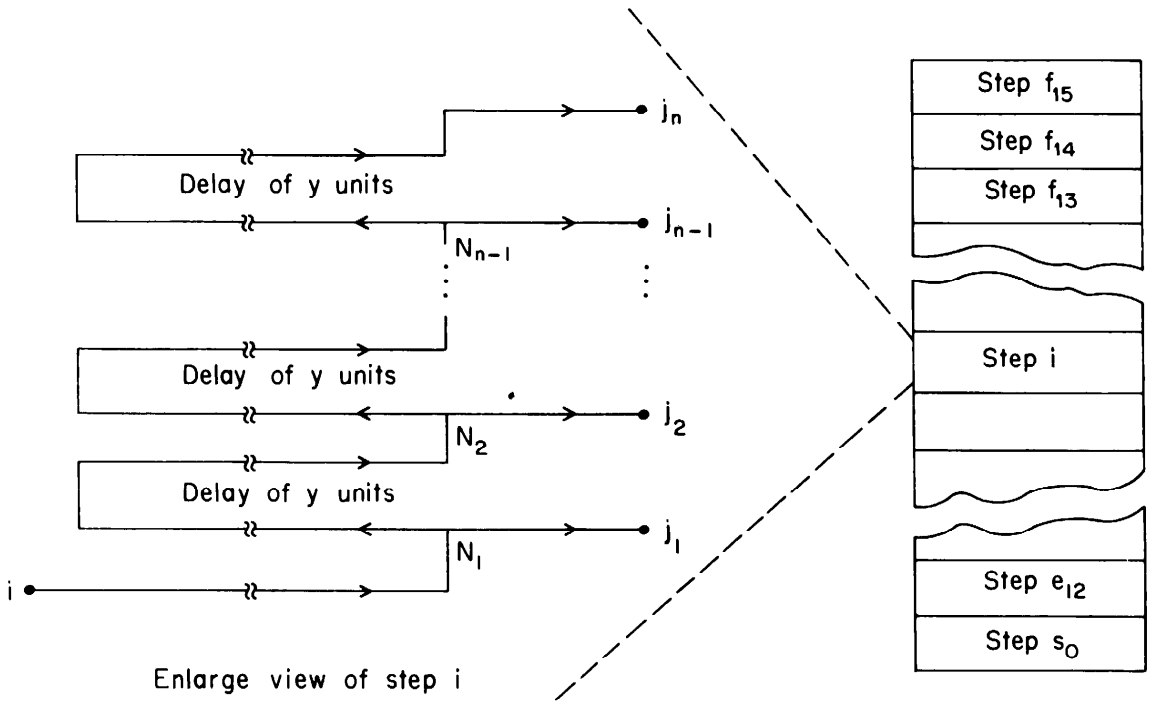


FIG. 6.3. The microprogram steps.

which depend on y (and which are located at the extreme left of the control section) are left unfinished until y has been calculated. Now we observe the path distances from each fan-out point N_r (Fig. 6.3) to the corresponding terminal gate nodes. We may take as value for y any integer larger than the maximum of all these path distances.

Choosing such a value for y guarantees that, for any signal arriving on control input path i, operation of the gates corresponding to output path j_r will be completed by this signal before it sets out on output path j_{r+1}.

Thus, we arrive at the following *behavioral interpretation* for the typical microlanguage statement listed above. Microprogram step i calls for the operation in sequence of gates j_r ($r = 1, 2, \ldots, n$). This sequence of operations, consequent upon the arrival of a signal 07 over path i, will be referred to as an execution of step i.

As an illustration of the two interpretations of a statement in the microlanguage consider the following sample drawn from the microprogram itself. The statement is

$$f_1\colon \quad C, s_3$$

Structurally, this statement means that the path f_1 (a control input path) is connected via output path C to gate C and, with a delay of y units, to the path s_3 which is the (internal) input path for another microprogram step.

Behaviorally, this statement means that step f_1 calls for the operation of gate C and, after this has been completed, calls for the step s_3 to be executed.

6.9 Microsequences

In order to make the microprogram more concise and clear we shall use certain abbreviations for frequently occurring sequences of primitive terms. Thus, the sequence

$$g_s, A, g_s \qquad (s = 4, 5, 6, 7)$$

is denoted by G_s. Assume gate g_s is on (which it normally is). Then the execution of G_s results in the transmission of signal $0s$ from section E to

the memory section over whatever tape or construction paths happen to be selected.

The next level of abbreviation consists of employing the symbolic codes for signal sequences discussed in Chapter 4 and listed in Table 4.7. These codes are now to be used as abbreviations for the sequence of gating operations which generates the corresponding signal sequence.

Take for example *xr*. In Table 4.7 this code represents the signal sequence 05–05–04–06. When used in the microprogram, *xr* stands for the microsequence,

$$G_5, G_5, G_4, G_6$$

or, written out in full,

$$g_5, A, g_5, g_5, A, g_5, g_4, A, g_4, g_6, A, g_6.$$

In this way we arrive at the list of abbreviations in Table 6.2.

TABLE 6.2
MICROSEQUENCES

Abbreviation	Operation	Microsequence
G_s	———	g_s, A, g_s
x	*extend*	G_7, G_6
xl	*extend left*	G_4, G_4, G_5, G_6
xr	*extend right*	G_5, G_5, G_4, G_6
r	*retract*	G_4, G_5, G_6, G_6
rl	*retract left*	G_5, G_6, G_6, G_6
rr	*retract right*	G_4, G_6, G_6, G_6
m	*mark if zero*	$G_7, G_6, G_4, G_5, G_7, G_6$
e	*erase if one*	$G_6, G_7, G_4, G_5, G_6, G_6$
sw	*sense and wait*	G_7, G_7
cs	*cap after sense*	G_4, G_6
i	*inject signal* 06	G_7, G_5, G_6
j	*inject signal* 07	G_6, G_7, G_7, G_6, G_6

The microsequence abbreviations should not be confused with UCC commands. The UCC commands are obtained from the program tape in coded form. They have to be decoded, interpreted, and executed. The microsequences, on the other hand, are "built-in" and are used to fetch UCC commands from tape and to implement their interpretation and execution.

6.10 Selection of Microprogram Steps

The following statements taken from the microprogram are intended to help the reader understand the way echo signals are employed to select alternative steps in the microprogram:

s_4: *sw*
e_4: *cs*, *E*
p_4: s_4
q_4: *H*, w_4, *rr*, *M*, *x*, *M*, *xr*, *P*, s_3.

Assume that prior steps have left the following gates turned off:

$$P, H, t_4, u_4, v_4$$

and all other echo destination, central, and memory gates are on. Assume also that the leftmost cell of tape *P* (labeled start cell in the UCC block diagram) is under the cap of path *P*. These two assumptions are in fact valid when the first step above is about to be executed.

Execution of this step results in the transmission of the signal sequence for *sense and wait* from section *E* out on path *P*. The leftmost bit of tape *P* is therefore sensed and an echo signal (06 if the bit is zero; 07 if the bit is one) is generated.

Shortly after this signal has been transformed by the echo discriminator, we would observe that one of the two gates in echo storage has been turned off (the *zero* gate if the echo was 06 prior to transformation, the *one* gate if it was 07), and, in addition, the echo receipt signal is on its way over path e_4 into the control section. (Remember that we assumed t_4 was off and all other t_i on.)

The next step of the microprogram to be executed is accordingly the one above which has e_4 as its label. The two outcomes of this execution are

1. The end of path *P* is restored to its normal (i.e., capped form) by the *cap-after-sense* signal sequence.
2. The echo storage read-out gate *E* is turned off, permitting a signal 07 from the periodic emitter to pass out of section *E* over

path p_4 or q_4 according as to whether the echo was a zero echo or a one echo (remember that we assumed u_4 and v_4 were turned off while all other u_i and v_i were left on).

Thus, the next step in the microprogram to be executed is either step p_4 or step q_4, according as the leftmost bit on tape P when last sensed was a zero or a one respectively. Step p_4 merely calls for step s_4 once again. Therefore, the net effect of the steps listed above is to cause the leftmost cell of tape P to be repeatedly sensed until it assumes state 1.

This cell—the start cell—can therefore be used to delay the start of action by the UCC on its program. Immediate starting can be achieved by arranging for the start cell to be in state 1 at activation time, delayed starting by arranging for its state to be 0 until some other machine changes it to 1.

Step q_4 prepares for the fetching and decoding of the next UCC command on tape P. Thus, gates H, t_4, u_4, v_4 are turned back on; the cap of path P is retracted away from tape P; both paths M, P are extended ahead by one cell; path P only is extended right so that its cap is now directly above the base cell of P (see diagram); path P is deselected and step s_3 is called for.

6.11 The Microprogram

We have now provided a detailed account of the behavioral meaning of several typical steps in the microprogram. Before listing the microprogram itself we define the state of the machine at the time the first step is executed.

All gates shown explicitly in the UCC diagram (Fig. 6.2) and all gates in the decoder must be on. They are put into this state by the activating signal, as described in Section 6.13 on self-reproduction by the UCC. The remaining gates belong to the echo switch, the echo discriminator, the many crossovers, and the one-way locks. All gates belonging to the echo switch and echo discriminator are off (except gate L, which is on). The same applies to all crossovers and one-way locks which have not been traversed by the activating signal in turning on the first-mentioned

set of gates. Those which have been so traversed are in their proper setting for routine operation.

The microprogram assumes certain initial positions for the tape paths. Any convenient initial position may be selected for construction path C.

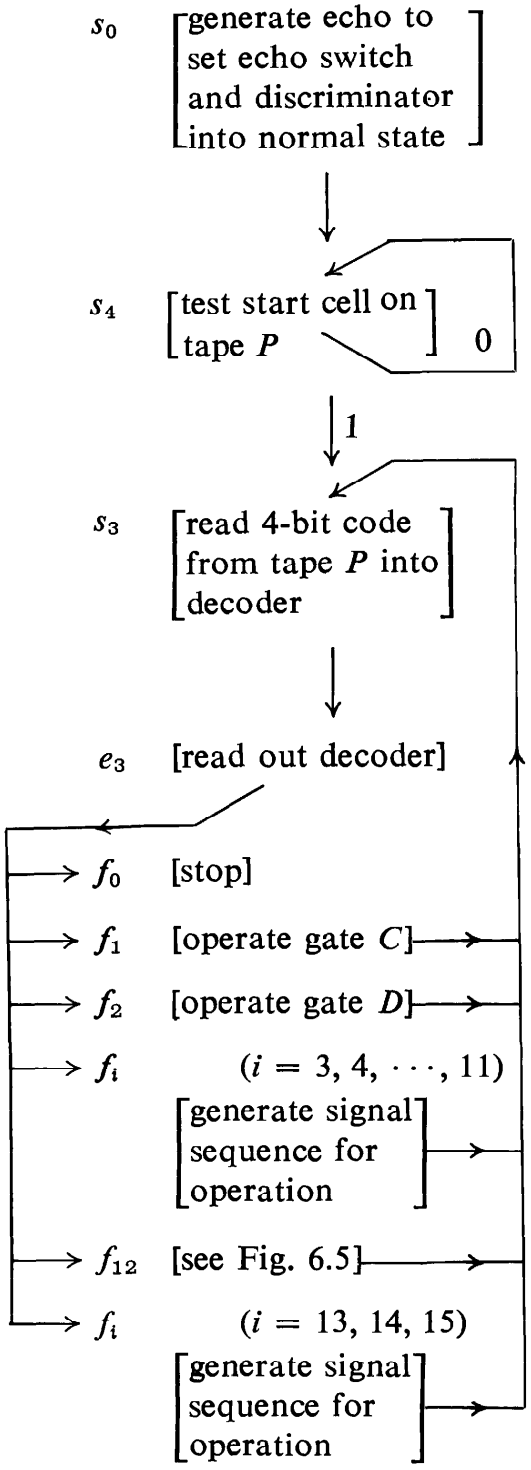


FIG. 6.4. Flow chart for microprogram.

These initial positions are defined by specifying the sequence of operations which will bring the appropriate path cap into contact with the leftmost cell of the corresponding tape. Thus, tapes D, K require the sequence xr–x; tape M requires x–xr; tape P requires simply xr.

The initial position of M permits the choice of x as the first micro-sequence to be executed. As a result of this choice, signal 07 is the first signal to be sent out from section E over the bidirectional path leading to the tapes. Crossover components are thus correctly initialized.

To provide for correct initial gate settings in the echo switch and echo discriminator, the first step of the microprogram to be executed (step s_0) calls for path M to sense the leftmost cell of tape M. This cell is

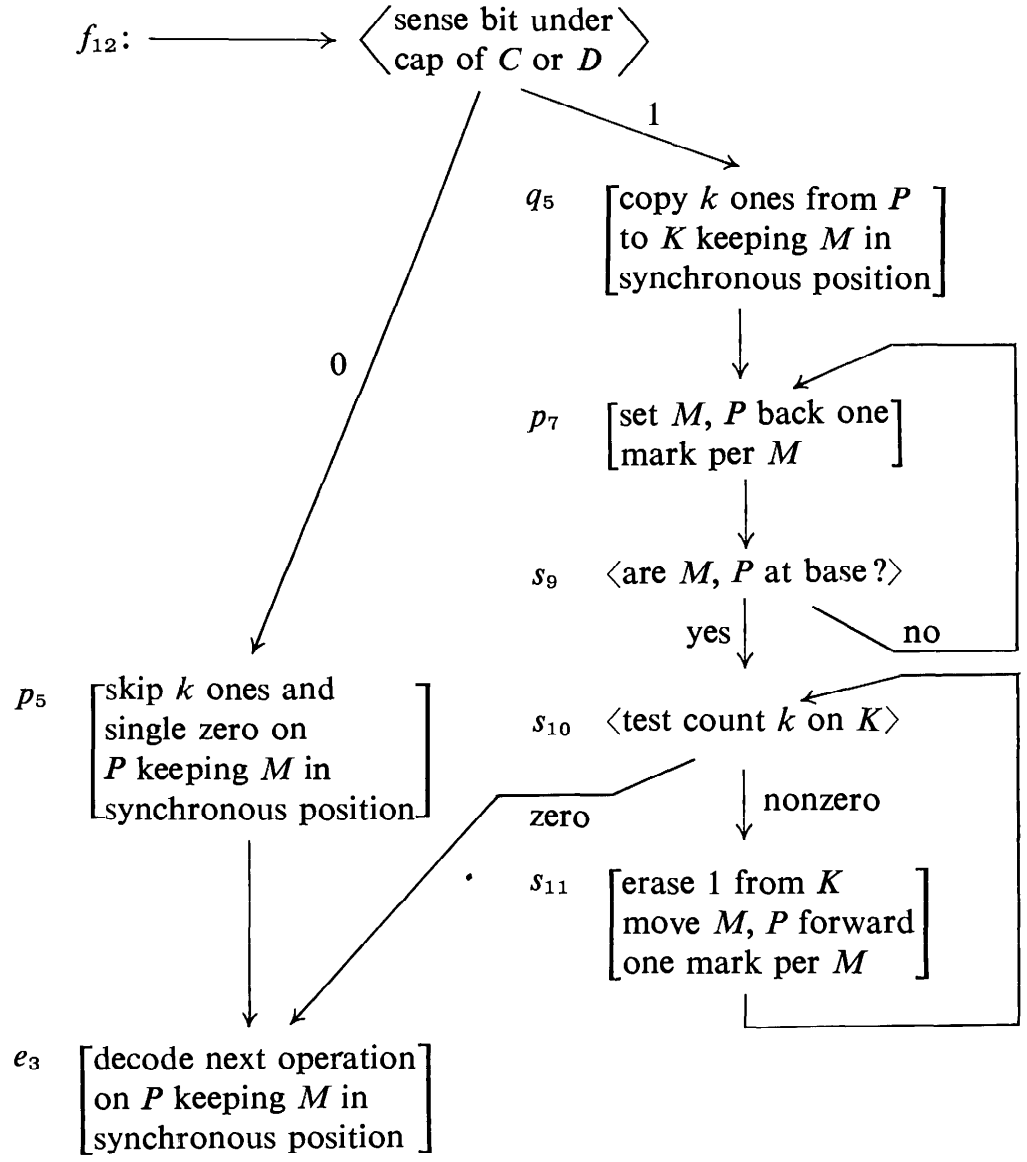


FIG. 6.5. Flow chart for conditional transfer.

known to be in state 1 (by the format rules for tape M). Hence, the first echo signal to pass back from the memory section to section E is 07, which gives the required initial settings.

The flow charts given in Fig. 6.4 and Fig. 6.5 are intended as aids to understanding the microprogram listing given below. The microprogram begins at step s_0.

s_0: M, x, xr, t_{12}, sw
e_{12}: $cs, rr, M, D, B, K, P, xr, P, x, K, B, D, P, H, w_4, s_4$
s_4: sw
e_4: cs, E
p_4: s_4
q_4: $H, w_4, rr, M, x, M, xr, P, s_3$
s_3: P, w_0, sw
e_0: $cs, rr, M, x, M, xr, w_0, w_1, sw$
e_1: $cs, rr, M, x, M, xr, w_1, w_2, sw$
e_2: $cs, rr, M, x, M, xr, w_2, w_3, sw$
e_3: $cs, rr, M, x, M, xr, w_3, P, F$
f_0: (see note)
f_1: C, s_3
f_2: D, s_3
f_3: s_3
f_4: B, x, B, s_3
f_5: B, xl, B, s_3
f_6: B, xr, B, s_3
f_7: B, r, B, s_3
f_8: B, rl, B, s_3
f_9: B, rr, B, s_3
f_{10}: B, m, B, s_3
f_{11}: B, e, B, s_3
f_{12}: H, w_5, B, sw
e_5: cs, B, E
p_5: w_5, w_6, P, s_6
s_6: sw
e_6: cs, E
p_6: $w_6, rr, M, x, M, xr, P, H, s_3$
q_6: rr, M, x, M, xr, s_6
q_5: w_5, w_7, P, s_7
s_7: sw
e_7: cs, E
p_7: $w_7, rr, M, r, P, xr, w_8, s_8$
q_7: $P, K, r, rr, x, xr, x, m, K, P, rr, M, x, M, xr, s_7$
s_8: sw
e_8: cs, E
p_8: rr, P, r, P, xr, s_8

q_8: w_8, *rr*, *r*, *xr*, w_9, *sw*
e_9: *cs*, *rr*, *x*, *xr*, *E*
p_9: w_9, w_8, p_8
q_9: w_9, *M*, s_{10}
s_{10}: w_{10}, *K*, *sw*
e_{10}: *cs*, *E*
p_{10}: *K*, w_{10}, *H*, *M*, *rr*, *M*, *P*, *xr*, *P*, s_3
q_{10}: *e*, *rr*, *r*, *xr*, *x*, *K*, w_{10}, w_{11}, *M*, s_{11}
s_{11}: *rr*, *P*, *x*, *P*, *xr*, *sw*
e_{11}: *cs*, *E*
p_{11}: s_{11}
q_{11}: w_{11}, *M*, s_{10}
f_{13}: s_3
f_{14}: *B*, *i*, *B*, s_3
f_{15}: *B*, *j*, *B*, s_3
(End of microprogram)

Note that, when the *stop* command *z* is decoded, a signal 07 goes to control over path f_0. The microprogram step f_0 is not connected to a control output path. This means that the path terminates in the control section. The effect of 07 arriving at the capped end of this path is to change the cap cell from state 2 to state 1. This cell therefore provides an effective means of determining when the UCC has stopped. (We could go further and have step f_0 restore the paths for the program tape and marker tape to their initial states, turn off the start cell, and call for repeated testing of this cell until it is turned on once again. For simplicity, we omit this.)

6.12 Some Conventions

The following rules for machine layout and construction planning are needed principally to avoid the generation of configurations for which the transition function is not defined. The rules as stated are somewhat more conservative than is absolutely necessary. This is done for the sake of simplicity.

The requirements for machine layout are

1. Paths running side-by-side should be so separated that, when sheathed, they do not share a common sheath.

2. Each gate node should be positioned so that the core bit under it is at least two cells removed from the core bit at a corner or junction.

The requirements for construction planning are

1. Any path which at some later time may have to be retracted should not have its sheath in contact with that of any other path.
2. Jogs in the construction path (produced by the sequence *xr–xl* or *xl–xr*) should be avoided.
3. When the construction path C is in position for the injection of the sheathing signal, the nearest corner in path C should be sufficiently far back that the sequence of commands *r–r* would be legitimate.
4. After the injection of the sheathing signal, sufficient time should be allowed before the injection of the activating signal so that the situation in which 07 reaches a gate on the control path before the subordinate path is sheathed is avoided.

Most of these requirements are automatically met if components are not crowded together and if ample space is allocated for the various maneuvers to be executed by the construction path.

As an illustration of the basis for the requirements listed above consider the situation shown in Fig. 6.6. Two paths share a common

```
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
1 1 1 1 1 1 1 1 0 4 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
1 1 1 1 1 1 1 1 1 5 0 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

FIG. 6.6. Sheath shared by two paths.

sheath in violation of the first requirement for machine layout. A signal 04 is proceeding along one path and 05 along the other. The neighborhood state 22425 arises, and the transition function is not defined on this state.

In order that the UCC be self-reproducing, we impose two further requirements:

1. The injection receiver should be located in a peripheral location where it is accessible to a construction path extended from some other machine.
2. The warm-up delay (see Fig. 6.2), which can be implemented by a single (unbranched) path, should be of sufficient magnitude to guarantee that all gates required to be on prior to execution of the first microprogram step are in fact on before the activating signal emerges from the delay.

6.13 Self-Reproduction by UCC

The block diagram of the UCC (Fig. 6.2), the component diagrams (Chapter 5), the microprogram and all associated conventions collectively define a set of possible implementations of the UCC. By implementation we mean a network of unsheathed paths which, if sheathed and activated, would become an active UCC.

Let us focus our attention on one of these implementations and denote it by J. Being a network of unsheathed paths, J is a (0,1) configuration.

Let the initial configuration in the cellular space be an active UCC ready to execute the first step of its microprogram and equipped with program tape P and marker tape M. Then, if J is sufficiently clear of UCC, there exist tapes P, M such that UCC constructs J. Note also that we can always choose a UCC which is located so as to provide sufficient clearance.

Since J contains an accessible injection receiver, tapes P, M can also provide for injection of first the sheathing signal and then the activating signal. Adequate delay between these two injections can be realized by having P, M call for "vacuous" sequences of operations on path C, such as x–r, xr–rr, and so on.

After completing the injection into J of the activating signal, UCC can withdraw its construction path C back to its initial position. Meanwhile, the activating signal in J is routed to all gates which must be turned on prior to the execution of the first microprogram step. A copy of the activating signal emerges from the warm-up delay after all these

gates are turned on. It proceeds both to the periodic emitter (thus setting it into action) and to the control section over path s_0 (thus initiating the first step in the microprogram).

At this point activation of the offspring J is complete. J is operating independently of its parent and its state is identical to the initial state of its parent. Thus UCC has reproduced itself.

Since J belongs to the class P^* for which UCC is universal in computation and construction, we have now established the combined computation-construction-universality of the 8-state, 5-neighbor cellular space defined in Chapter 4.

Chapter 7

Methodology

7.1 Experimental Stage

The cellular spaces under investigation at this stage were those with from two to eight states per cell and 90°-rotation-symmetric neighborhoods with five or nine elements. Having selected a particular number of states per cell and a particular neighborhood, experimentation began with the transition function completely undefined. The long-range goal was to find among the immense number of possible transition functions one (or more) which, in conjunction with appropriate initial configurations, would yield behavior which could clearly be identified as computation and construction. The identification was required to be in accordance with the definitions of Chapter 2.

Exhaustive search was out of the question, even for the largest and fastest digital computers—not only because of the immensity of the population of transition functions (though that alone would have sufficed), but also because the set of all possible configurations from which a choice had to be made was infinite. Since the goal of exhibiting a universal computer-constructor was such a distant one, tentative subgoals were formulated and attempts made to realize them. Many examples of these subgoals were cited in Chapter 4. Many others not cited there were tried, but in most cases these proved incompatible with earlier or later goals.

Typically, one or more of the transitions needed for the target behavior turned out to be incompatible with the transitions already in

the transition table (note that each neighborhood state can have no more than one image). In some cases, especially when the table had grown large, it appeared to be impossible to overcome the incompatibility without going back to some subgoal realized at a much earlier stage and striking out its transitions from the table. Then, a new attack was made on the implementation of this earlier subgoal so as to make the most recent subgoal realizable. The great advantage of using a digital computer for this experimenting lay in its speedy execution of the humdrum calculations and bookkeeping and the resulting freedom for the experimenter to devote his energies to the more exciting matters of frequent changes in tactics and occasional changes in strategy.

7.2 An On-Line Program

We refer to the computer program as P and discuss it in two parts: first, its basic processing abilities, and second, its provisions for man-machine interaction.

The basic function of P is to apply a partially defined transition function f to a finite rectangular array A of cells in which a configuration of states has been initially planted. The application of f entails

1. Invoking the neighborhood definition and symmetries for the particular cellular space in question.
2. Reporting externally any neighborhood state which is encountered in A and for which f is not defined.

At these points of nondefinition of f, human judgment is needed. Specifically, the experimenter must decide which of the cellular states of the given space shall be the image under f of the neighborhood state just printed out. Having made his decision, he communicates it to P via the typewriter, using one of the on-line commands discussed below. Processing of the array then continues until another neighborhood state is encountered for which f is not defined or until the array has been completely updated for one time step.

To get a series of experiments started, P is loaded together with the parameters for the class of cellular spaces to be investigated. These

parameters consist of the number of states per cell and a precise definition of the neighborhood expressed in the form of a list of template coordinates. The ordering in this list is later used by P when generating the coded form for neighborhood states.

An initial transition table is then read in. Depending on the type of experiment to be carried out, the table may be large or small. Following the table the rectangular array A of cellular states (in initial form) is loaded. The experimenter may now call for the first time step in the cellular space to be computed.

Decisions are required of the experimenter not only at the points of nondefinition of f, but also at the end of each time step in the cellular space. Here, the experimenter must decide whether P is to proceed to the next time step, whether it is to be set back to some previous time step, or whether the experiment should be restarted with different initial conditions. In setting back or restarting it may be necessary to erase some or all of the most recent additions to the transition table and to restore the array A to its initial form. Small changes are typed in directly. Large changes are handled by loading table and array tapes dumped at some previous time. The dumping activity itself is yet another item requiring judgment on the part of the experimenter.

In order that the experimenter may readily put into effect all the various kinds of changes he may consider necessary, P has been designed to interpret a set of on-line commands. These commands are listed below in seven categories. The following items of information concerning the digital computer used for the experiments may help the reader understand these commands better.

The computer in question was small in scale, having 8 K words of core memory with 18 bits per word and a core access time of 5 microseconds. Although it was equipped with several different types of input-output equipment, only four devices were used: the paper tape reader, paper tape punch, on-line typewriter, and off-line Flexowriter. Thus, on-line commands of the first category—those for tape input—are concerned with control of the paper tape reader, interpretation of appropriate formats, and storage of information in appropriate parts of core memory. Similarly, the commands in category six, which refer to punching, are concerned with control of the paper tape punch, generation of appropriate output formats, and acquisition of information from appropriate parts of core memory. The commands them-

selves along with their parameters are, of course, entered into the computer at the on-line typewriter.

To facilitate man-machine interaction almost all of the subroutines of *P* end by interrogating the typewriter. In certain cases, *P* waits until the experimenter reaches a decision and enters an on-line command. In other cases, switches are sensed to determine if the experimenter wishes to intervene at this time. We now list the on-line commands.

On-Line Commands

1. Processing

z	*clear arrays A, B*
a	*apply f to A and produce B*
m	*move B to A and clear B*
ma	*move B to A, clear B, apply f to A*

2. Table Modifications

sxK	*search table f for neighborhood state K; if found, type out the image; otherwise, type out "?"*
de	*delete element specified in most recent sxK command; type out number of elements remaining in table f*
eJ	*erase last J elements of table f*
xK	*append to table f* *either (1) element K (six characters)* *or (2) implied element with K (one character) as image*
sfJ	*sort all transitions in table f on column J*

3. Array Modifications

aI, J	*set array focus to row I, column J*
cxK	*change focal element to K and advance focus 1 right*
uJ	*move focus up J positions*
dJ	*move focus down J positions*
lJ	*move focus left J positions*
rJ	*move focus right J positions*

4. Tape Input

ip	*input parameters of cellular space*
if	*input transition table f*
ia	*input array A*
ic	*input octal corrections*

5. Typewriter Input

wI, pJ	*type plus octal J into location I*
wI, mJ	*type minus octal J into location I*

6. Output

tpI	*type out contents of location I*
tmI	*invert sign and type out from location I*
ta	*type array A*
tb	*type array B*
tf	*type transition table f*
pa	*punch array A*
pb	*punch array B*
pf	*punch transition table f*

7. Command Modification and Execution

/	*ignore what has just been typed in*
tab	*execute command just typed in*

7.3 Why On-Line?

In an investigation of this kind the case for direct man-machine interaction is overwhelmingly strong. Consider the following aspects.

After a number of attempts to obtain a certain kind of behavior, the experimenter may chance upon a neat approach. It is a great deal easier for him to recognize a neat approach when it arises than it is to characterize in advance the class of all neat approaches.

In several of our experiments phenomena occurred which were totally unexpected. An example of this was the echo signal, which was accidentally generated in an experiment with a class of 3-state spaces. Again, it is far easier for the experimenter to recognize interesting phenomena when they occur than it is to characterize in advance the class of all interesting phenomena.

Rapid interaction between man and machine tends to develop the intuition and perceptivity of the experimenter. The immediacy of the machine's response has the effect of maintaining a high level of cerebral activity in the experimenter.

We have already observed that exhaustive search is out of the question. What then of the alternative of carrying out experiments off-line? The effect of this would be to slow the machine's response from a second (or even less) to four hours (or even more). Inevitably under such a regime progress would be agonizingly slow and the experimenter would very likely decide to turn his attention to other problems.

7.4 Testing Stage

The end result of the experimental stage was the 8-state, 5-neighbor cellular space defined in Chapter 4. It was felt desirable to subject the transition function for this space to larger scale tests than those it had passed in the course of its development.

Two major tests were carried out. Both were completed successfully in all respects. The first test involved the construction—in completely passive form and at an initially quiescent site—of part of the universal computer-constructor described in Chapter 6. The test continued with the injection of a sheathing signal and an activating signal to determine if the various components worked properly in conjunction with one another. The components tested in this way included the injection receiver, periodic emitter, signal transformer type 456, and one-way lock.

A large volume of printed output was obtained to show in detail the progress of construction, activation, and operation of the activated structure. Only a very small sample of this output can be included here.

In Fig. 7.1 at location ① a small part of the (0, 1) configuration has already been constructed. Note how the construction path ② is

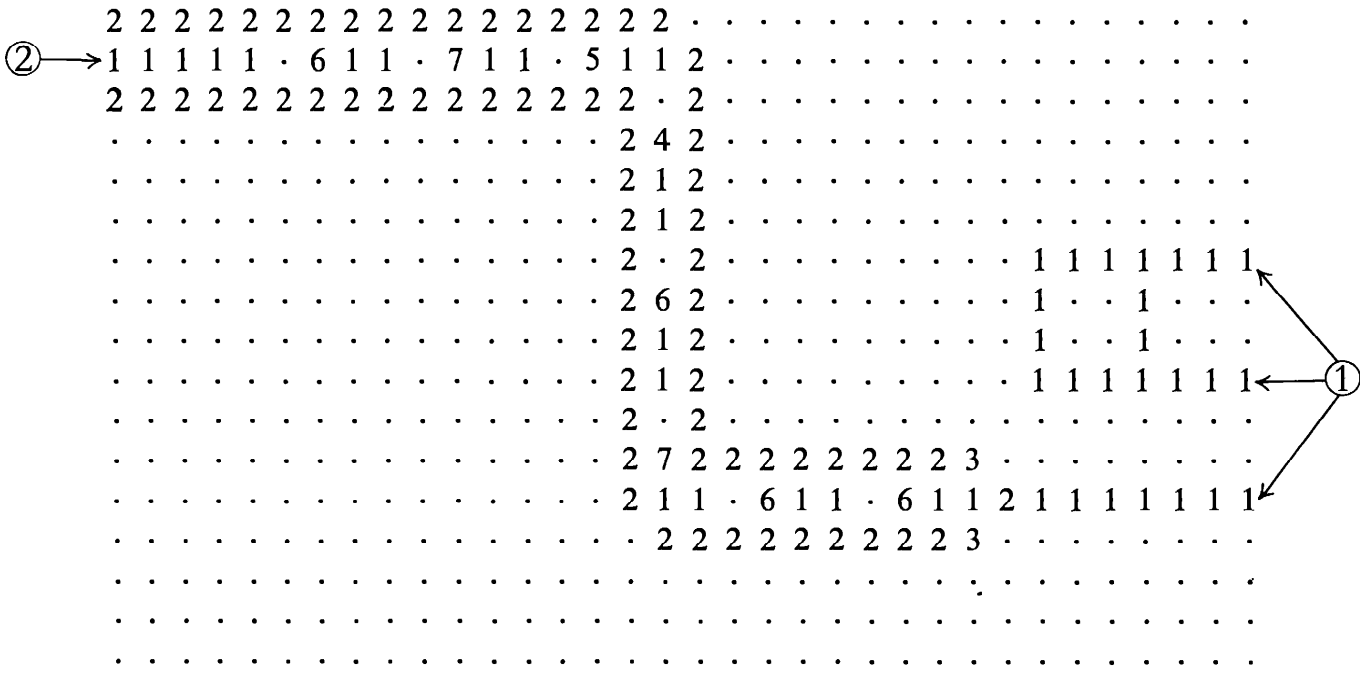


FIG. 7.1. Construction in progress.

extended from the top left corner of the rectangular array via two right-angled bends to the perimeter of the partially completed structure. The two 06 signals near the path end are about to complete the operation *retract*. Following them is the signal sequence for the operation *mark*, which will add one more bit to the structure. Additional signals are periodically inserted at ② by the digital computer in accordance with a construction program expressed in terms of UCC commands.

Figure 7.2 shows the (0, 1) configuration in completed form ready for injection of the sheathing signal. The construction path is in position for this injection to be made and, in fact, the signal 05 is about to enter the injection receiver at ③. The receiver will transform it to 06 and sheathing will begin.

Figure 7.3 shows activating signal 07 at ④ just after its injection. Sheathing is still in progress at ⑤, but has proceeded sufficiently far that the activation signal will not encounter any gates whose subordinate paths are still unsheathed.

Figure 7.4 shows the fully sheathed and activated structure. Careful examination of the sheathing discloses that the three gates in the structure have all been turned on (see ⑥, ⑦). A 07 signal is circulating clockwise at ⑧ in the periodic emitter; another is proceeding to the right at ⑨. Signals 04, 05, 06, 07 periodically leave the structure on the

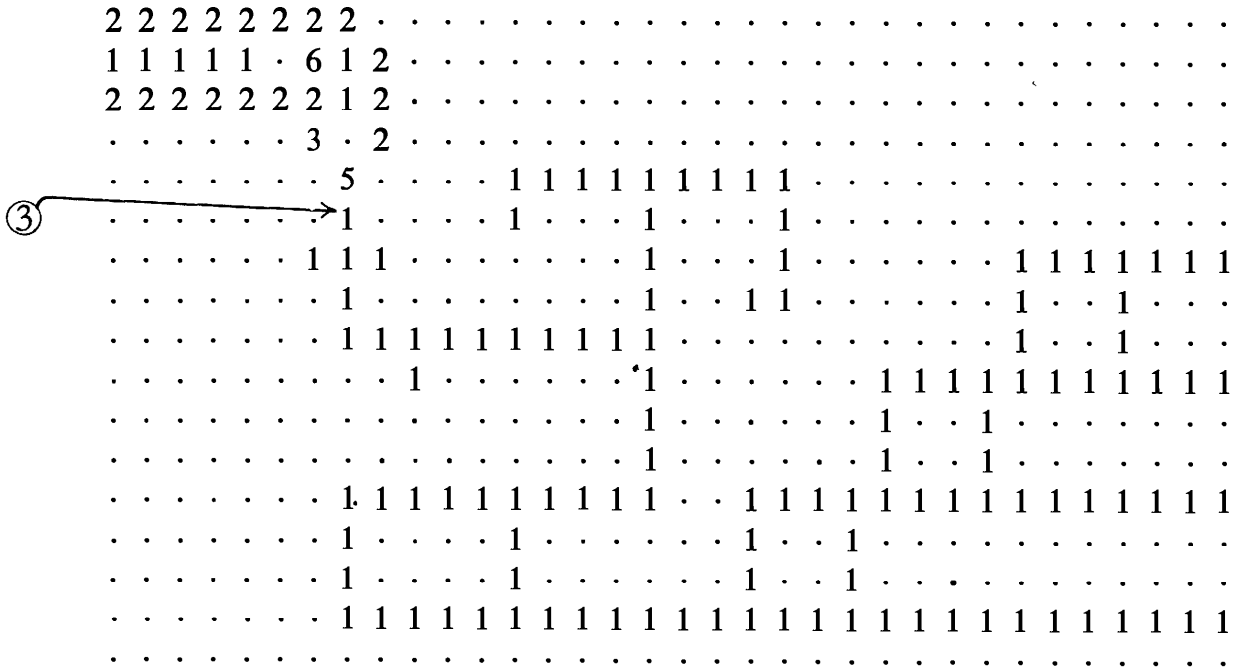


FIG. 7.2. Injection of sheathing signal.

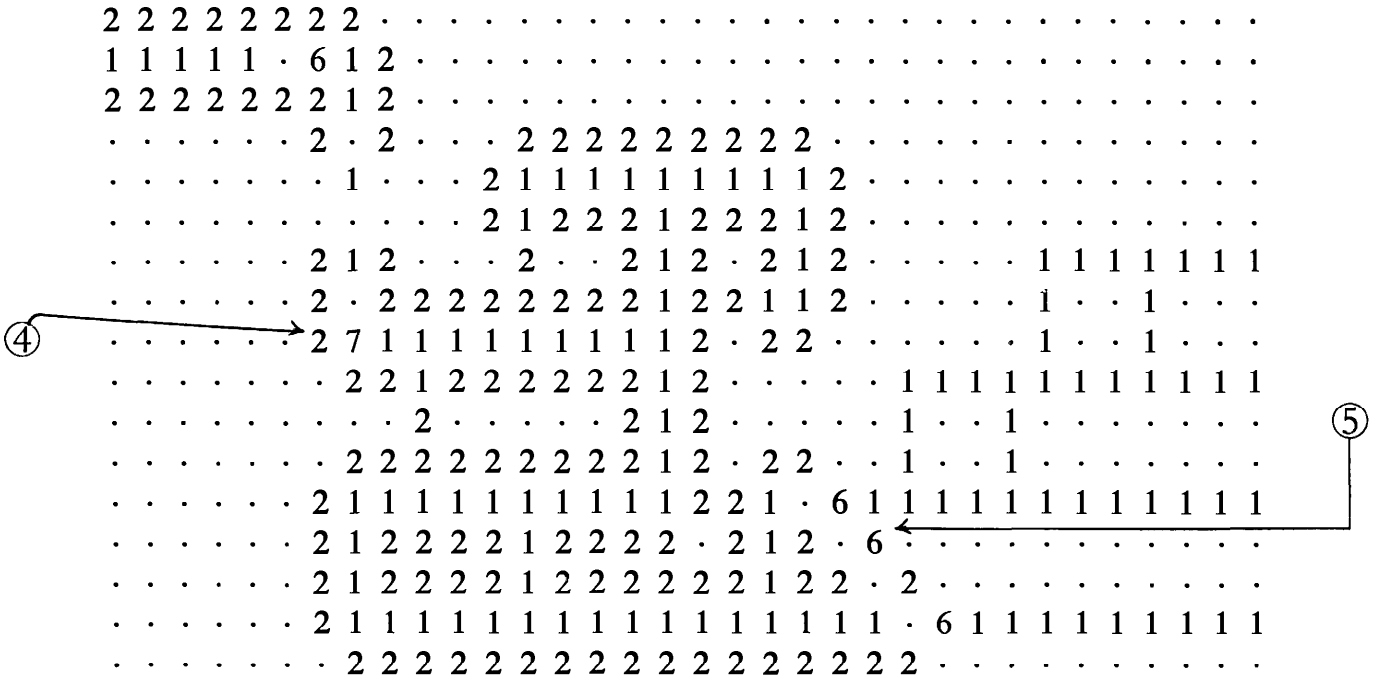


FIG. 7.3. Activating signal injected.

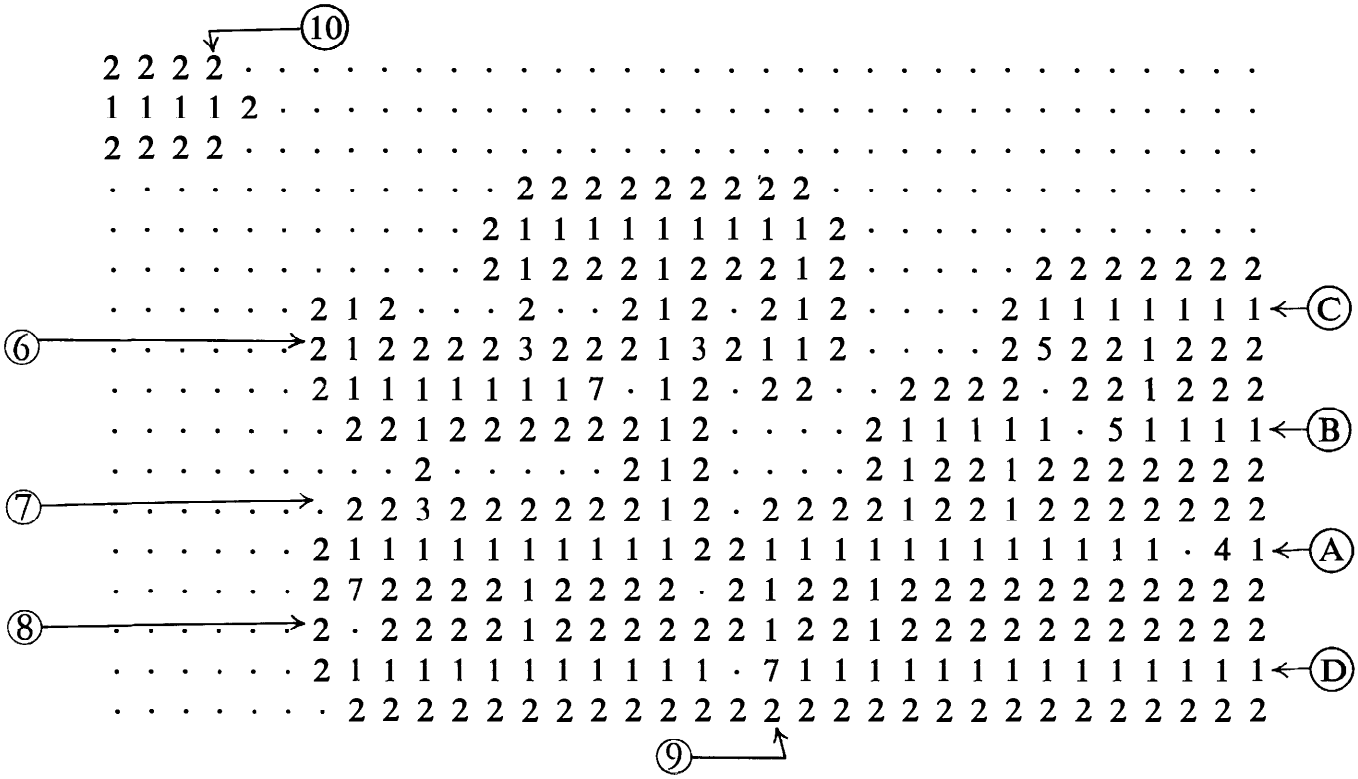


FIG. 7.4. Fully sheathed and activated structure.

paths labeled Ⓐ, Ⓑ, Ⓒ, Ⓓ. In the figure, signals 04, 05 are about to leave. In four time steps the collision of two copies of 05 will cause 06 to be generated. Note how the construction path has been withdrawn to ⑩.

The second major test was concerned with the operations *sense and wait*, *cap after sense*, and *erase*. An initial (0, 1) configuration was set up with the property that in its perimeter both 0 and 1 occurred with all the distinct (0, 1) neighborhood states possible in a perimeter. The first part of the test consisted of the execution of a program which advanced the construction path successively to each cell in this perimeter and sensed its state. In each case the echo signal was observed to be the correct one. The construction path was advanced in the second part of the test to just those perimeter cells in state 1 and the *erase* operation executed to convert them to 0. Again, each of these operations was executed in the correct manner.

Chapter 8

Conclusions

8.1 Summary

The principal results developed in this book are as follows:

1. There exists a strongly rotation-symmetric 8-state, 5-neighbor cellular space which is computation-construction-universal (see Chapters 4, 5, 6).
2. In this space all two-dimensional completely passive configurations can be read and erased (as well as written) by other configurations. As a consequence, this space may well be strictly more powerful than von Neumann's (weakly-) rotation-symmetric 29-state, 5-neighbor cellular space.
3. A necessary condition for computation universality is that, by choice of initial configuration, it is possible to obtain boundable propagation arbitrarily far from its source. This result is stated more precisely in Section 3.3, Proposition 5.
4. There does *not* exist a computation-universal 2-state, 5-neighbor cellular space with a neighborhood which is 90°-rotation-symmetric (see Proposition 6).
5. However, there does exist a 2-state cellular space which is computation-construction-universal (see Proposition 8).

In addition to the development of these results, several new techniques are described:

1. The three-phase development of organization, when one machine is constructing another, starting with the development of a (0, 1) configuration, followed by sheathing and activation.
2. A new set of commands, which is universal with respect to computation and construction, and at the same time does not depend on the existence in the space of a distinguished direction.
3. The echo signal technique for reading completely passive configurations (1- and 2-dimensional)—a technique which eliminates the need for a specially designed tape unit (compare von Neumann and Thatcher).
4. The use of bidirectional paths and their exploitation by means of cellular gates.
5. The sheathing of paths to provide insulation and a means of establishing the relative directions left and right.
6. A technique for simulating one cellular space by means of another.
7. Experimental techniques for exploring propagation, computation, and construction by means of on-line interaction with a general-purpose, digital computer.

8.2 Open Questions

The work reported in previous chapters has generated a number of interesting questions for which, it is believed, answers are not immediately available. To conclude this work we list some of these questions.

1. Is the existence of unbounded but boundable propagation necessary to computation universality?
2. By removing the restriction that tapes must be completely passive configurations, can an interpretation of computation in cellular spaces be developed for which both the computation and construction defined in Chapter 2 are special cases?
3. More generally, what can be said about interpretations of computation in cellular spaces in which the distinction between tape and control unit is discarded?

4. In another direction, can we find sufficient conditions for the computation and construction universality of a cellular space—preferably expressed in terms of a set of elementary behaviors? If such conditions were available, proofs of universality such as that given in Chapters 4, 5, 6 might be cut down to a tolerable length. Experimental investigations might also be simplified.

5. Consider the maximum number p of bits of completely passive information which can be stored per cell in readable, writable, and erasable form. Let q be the total bit capacity of each cell, i.e.:

$$q = \log_2 n$$

where n is the number of states per cell. How close can the data efficiency p/q approach 1? Note that Moore's Garden-of-Eden theorem [4] implies that the data efficiency must be strictly less than one. More generally, what can be said about the density of Garden-of-Eden configurations in cellular spaces? (Note that many people have mistakenly interpreted Moore's theorem as implying that all cellular spaces are nonuniversal with respect to construction.)

6. Can the partial transition function specified in Chapter 4 be extended (without adding new states to the space) so as to make possible the destruction of a machine which is active but not actively defending itself? It seems quite likely that, if a ninth state were added, it would be possible for one machine to inject a signal into another which would strip the sheathing from all patns of the receiving machine, thus converting it to passive form—and this could finally be erased.

The reader who is interested in obtaining information on a variety of related topics is referred to the book "Essays on Cellular Automata" [8].

References

1. Hedetniemi, S., Variants of Thatcher's algorithm for constructing pulsers. Tech. Rept. 03105-29-T, ORA, University of Michigan, 1964.

2. Holland, J. H., Iterative circuit computers. *Proc. Western Joint Computer Conference, San Francisco, California, May 3–5, 1960,* pp. 259–265.

3. Lee, C. Y., Synthesis of a cellular universal machine using the 29-state model of von Neumann. *Automata Theory Notes,* University of Michigan Engineering Summer Conferences, 1964.

4. Moore, E. F., Machine models of self-reproduction. *Proc. Symposia Applied Mathematics,* Vol. XIV, Amer. Math. Soc., 1962, pp. 17–34.

5. Thatcher, J. W., Universality in the von Neumann cellular model. Tech. Rept. 03105-30-T, ORA, University of Michigan, 1964.

6. von Neumann, J., Theory of automata: construction, reproduction, homogeneity, Part II of "The Theory of Self-Reproducing Automata," ed. A. W. Burks. University of Illinois Press, Urbana, Illinois, 1966.

7. Wang, H., A variant to Turing's theory of computing machines. *J. Assoc. Comp. Mach.,* **4**, 63–92 (1957).

8. "Essays on Cellular Automata," ed. A. W. Burks. University of Illinois Press, Urbana, Illinois, 1968.

Index

Author entries do not include complete references; see p. 118.