Simple Computation-Universal Cellular Spaces

ALVY RAY SMITH III*

New York University, The Bronx, New York

ABSTRACT. The specialization of the theory of cellular spaces (cellular automata) to those spaces which compute partial recursive functions is presented. Neighborhood reduction and state-set reduction are shown to be particularly simple in this special theory, and one dimension is proved to be sufficient for computation universality. Several computation-universal cellular spaces (CUCS's) are exhibited which are simple in the sense that each cell has only a small number q of states and a small number p of neighbors. For example, a 1-dimensional CUCS with $pq = 36$ is presented. Two quite different proofs of the existence of a 1-dimensional CUCS with only two neighbors are given. Finally, one of the theorems derived is used to settle three open decidability questions.

KEY WORDS AND PHRASES: cellular automata, cellular space, tessellation automata, tessellation space, iterative array, modular computer, universal computer, template, stencil, Turing computable, complexity trade-offs, tag system, minimum neighborhood, axiom system, PC-simple, primitive configuration, von Neumann neighborhood

CR CATEGORIES: 5.22

Introduction

This paper is a specialization of the general theory of cellular automata to that particular subset of cellular automata which perform computations of partial recursive functions. The general theory is presented in [11]. As will be seen, such specialization leads to substantial simplification over what was previously obtained in such areas as neighborhood reduction and state-set reduction. We will show the existence of quite simple computation-universal cellular spaces, where by "simple" is meant low state count and small neighborhood. These spaces are to be contrasted with the computation-universal spaces of von Neumann [15], Thatcher [14], Codd [3], and Arbib [1]. The technique to be exploited in this paper is that of simulation of Turing machines by cellular automata. In particular, we use a special simulation technique called "wiring-in," the first example of which is described in the proof of Theorem 1 below. Such techniques will finally be used to answer several questions posed by Lieblein [6].

Definitions

Cellular automata theory is of such a nature that it invites the use of quite colorful —or more descriptively, picturesque—terminology, especially in the 1-dimensional (1-D) and 2-dimensional (2-D) cases with which we will be most concerned here. We exploit this property of visualization in the following intuitive model of a cellu-

* Department of Electrical Engineering. Portions of this paper were presented at the IEEE 1968 Ninth Annual Symposium on Switching and Automata Theory. The work was performed under a National Science Foundation Traineeship at Stanford University.

Journal of the Association for Computing Machinery, Vol. 18, No. 3, July 1971 pp. 339–353

Copyright © 1971, Association for Computing Machinery, Inc.

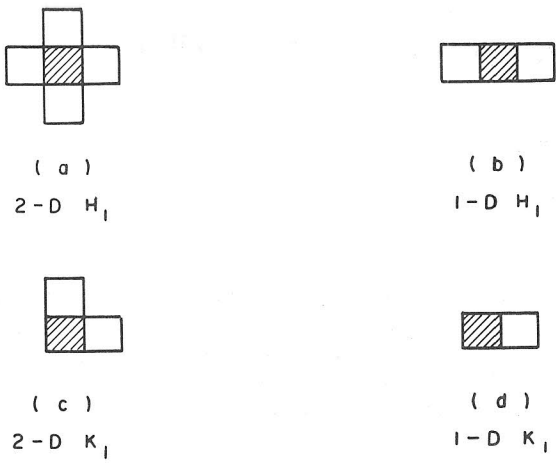


FIG. 1. Some common templates.

lar space. The intuitive model is sufficient for our purposes here; a formal definition can be found in [11].

A *cellular space* can be visualized as an infinite chessboard in two dimensions or an infinite strip of film in one dimension, each square or each frame of which represents a copy of a single finite-state automaton, or *cell*. The space is assumed to operate synchronously in discrete time steps. Each cell has associated with it three things: a *neighborhood*, a *local transition function*, and a *quiescent state*. The state of a cell at time $t + 1$ is given by the local transition function f and depends on both its state at time t and on the states at time t of the cells in its neighborhood, its *neighbors*. The neighborhood of a cell D is a finite set of cells in fixed positions relative to D. If all cells in a given cellular space Z have neighborhoods of the same shape, then the cellular space is said to be *uniform*. In this paper we shall be generally concerned with uniform cellular spaces. It is very probable that uniformity will be relaxed in use of the cellular space model for studying, say, embryology theory where space-varying and time-varying neighborhoods may play an important role. In uniform cellular spaces the one neighborhood type associated with all cells in a given space can be designated by a subset of chessboard squares called a *template*, is as indicated in Figure 1 (a), where we hatch the cell whose neighborhood this is. Thus the neighborhood of cell D is determined by translating the template associated with Z until the hatched *template origin* covers cell D. All cells under the template squares then form the neighborhood of D. A 1-D template in which each square, except the rightmost, shares its right edge with the square to its immediate right is said to be *contiguous*, and a corresponding cellular space is called a *contiguous cellular space*. The quiescent state q_0 is defined such that if a cell and all its neighbors are quiescent at time t, then at time $t + 1$ the cell is still quiescent. Besides these restrictions on each cell, there is often a restriction on the entire space: at time zero, the initial *configuration*—i.e. the initial assignment of states to each cell in the space—must contain only a finite number of nonquiescent cells. Although the restriction is assumed throughout this paper, we shall not make it a requirement on initial configurations. To do so would preclude several interesting problems corresponding to infinitely inscribed initial Turing machine tapes and invalidate at least one paper in the field [5]. Given an ini-

tial configuration c_0, the *global transition function* F (simultaneous invocation of the local transition function in each neighborhood of the cellular space) determines a sequence of configurations, the *propagation* $\langle c_0 \rangle$:

$$c_0, c_1, \cdots, c_t, \cdots,$$

where $c_{t+1} = F(c_t) = F^{t+1}(c_0)$ for all times t. Thus $F(c)(C) = f(N(c, C))$ where C is a cell with neighborhood state $N(c, C)$ in configuration c.

Note that each cell in a cellular space can be assigned a point in an integer lattice. Let $\mathbf{Z}$ be the integers. Then a configuration c in a d-dimensional space Z is a mapping $c: \mathbf{Z}^d \to Q$, where Q is the state set of each cell in Z. Define the *support* of configuration c to be the set sup (c) of nonquiescent cells in v. Usually the term "configuration c" will be used loosely to mean $c \mid \sup(c)$. A configuration c is *passive* if $F(c) = c$. A configuration c' is a *subconfiguration* of c if $c \mid \sup(c') = c' \mid \sup(c')$. By *disjoint* configurations c and d we mean their supports are disjoint. By the notation $c \cup d$ we mean the *union* of c and d, defined by

$$(c \cup d)(C) = \begin{cases} c(C), & \text{if } C \in \sup(c), \\ d(C), & \text{if } C \in \sup(d), \\ q_0, & \text{else,} \end{cases}$$

if c and d are disjoint.

The terminology above is essentially that of Thatcher [14] and Codd [3]. However, the following definition of computation is a slight generalization of their definitions. First, we need several preliminary concepts at hand. From recursive function theory (see [8] for details) we know that there exist effective enumerations of the partial recursive functions. Let the sequence (ϕ_i) be an enumeration of all partial recursive functions of one variable. We will have occasion to use the following well-known theorem.

UNIVERSAL TURING MACHINE THEOREM. *There exists a j and a total recursive function e of two variables such that, for all i and x, $\phi_j(e(i, x)) = \phi_i(x)$ if $\phi_i(x)$ is defined and is undefined if $\phi_i(x)$ is undefined. (We will call ϕ_j in this case a universal Turing machine function and e the encoder function associated with ϕ_j.)*

Consider the configurations with finite support in cellular space Z. Clearly there is an effective enumeration, or indexing of these configurations. Let the sequence (χ_i) be such an enumeration. Then we will speak of partial recursive functions from (χ_i) into (χ_i), instead of from $\mathbf{N}$ into $\mathbf{N}$ where $\mathbf{N}$ is the set of natural numbers. Now we can define computation in a cellular space.

Definition 1. Given a cellular space Z with global transition function F, configuration c, and partial recursive function $g: \mathbf{N} \to \mathbf{N}$, *$c$ computes g* if:

(1) there is a sequence of configurations (d_n), each disjoint from c, which is an effective enumeration of (not necessarily all) configurations;

(2) there is a partial recursive function $h: (\chi_i) \to (\chi_i)$ such that, if $g(n)$ is defined, then there exists a time t_0 such that $h(F^{t_0}(c \cup d_n)) = d_{g(n)}$ (see Remark 1 below); and

(3) there is an $m \geq 1$ and a recursive function $\pi: (\chi_i)^m \to \{0, 1\}$ for determining from a finite sequence of m configurations that t_0 has occurred—i.e. $\pi(c_t, c_{t+1}, \cdots, c_{t+m-1}) = 1$ if and only if $t = t_0$ (see Remark 2 below).

Remark 1. The decoding function h is assumed to be simple in the sense that the cellular space does the computing, not h. h is merely intended to indicate which cells of a given configuration are to be interpreted as the "result" configuration $d_{g(n)}$. For example, we would not want h to be as powerful as a universal Turing machine function, nor would we want $h = g$. The important point is that h is the same for many different functions g.

Remark 2. In this paper, the procedure π for determining completion of a computation is either $\pi(c_t, c_{t+1}) = 1$ if and only if $c_t = c_{t+1}$ (hence $t \geq t_0$) or a closely related variation for which detailed explanation is delayed until presentation of Theorem 1. Thus completion of a computation is signaled by the cellular space (except, perhaps, the extremities of its support) becoming passive.

Definition 2. Z is a *computation-universal cellular space* (CUCS) if there exists a set U of configurations in Z such that for any partial recursive function g, one can effectively find a $c \in U$ such that c computes g.

It is the CUCS which is here our major concern. The principal investigative tool is "simulation," a term defined below. We shall require cellular spaces to simulate other computing devices such as Turing machines. In general, a *monogenic* (cf. deterministic) logical string-manipulation "system" is one for which, given a string S, there is at most one string S' which can be obtained from S in one step. We call the straightforward generalization of the concept of string to d dimensions a *pattern*. Thus a string is a 1-D pattern, and a configuration with finite support in a d-D cellular space is a d-D pattern. A cellular space is then a "pattern-manipulation system," as is a Turing machine or a Post tag system.

Definition 3. A *pattern-manipulation system* T is an ordered pair (P, v), where P is an effectively indexed set of finite patterns so that $v: \mathbf{N} \to \mathbf{N}$ may be considered as mapping patterns; thus T is monogenic.

Definition 4. Consider cellular space Z and pattern-manipulation system T. Let k_1 and k_2 be positive integers; let i index patterns in P; and let $\mathbf{C}$ be the set of all configurations in Z with finite support. Then Z *simulates* T *in* k_2/k_1 *times real-time* if and only if there exist effectively computable and injective mappings $\gamma: \mathbf{N} \to \mathbf{C}$ and δ of recursive functions into recursive functions such that

$$F^{k_2}(\gamma(i)) = \gamma(v^{k_1}(i)),$$

where $F = \delta(v)$. We are particularly interested in the following three cases: (1) $k_1 = 1$, $k_2 = k > 1$ (k *times real-time*); (2) $k_1 = k > 1$, $k_2 = 1$ (k *speed-up*); (3) $k_1 = k_2 = 1$ (*real-time*).

Remark. As with function h in Definition 1, we assume functions γ and δ do not exceed a fixed level of computational complexity in some well-defined hierarchy of computational complexities, although we shall not specify such a hierarchy here. For example, the functions might be restricted to those requiring at most linear time to compute by a single-tape off-line Turing machine.

A Turing machine is then a pattern-manipulation system where P is a set of "instantaneous descriptions" and v is uniquely determined by the next-state function of the control head of the Turing machine. Note that we have implicitly assumed that when a Turing machine or tag system "halts," the associated sequence of patterns continues forever but is passive [i.e. $v(i) = i$].

We will often refer to an (m, n) *Turing machine.* By this is meant a Turing ma-

chine with m symbols and n states which is specified by a state table of the following form (only one typical entry shown):

		states			
		q_0	q_1	$\cdots$	q_{n-1}
symbol	x_0				
	x_1				
	$\vdots$		x_iX/q_j		
	x_{m-1}				

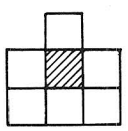
$X \in \{R(\text{right}), L(\text{left})\}$,
$0 \leq i \leq m-1$,
$0 \leq j \leq n-1$.

Cellular Automata Which Compute

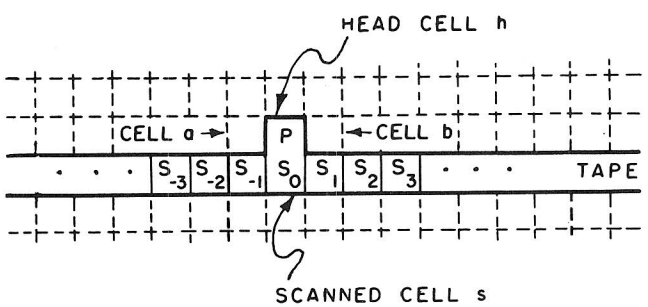
The first theorem below appears essentially unchanged from its presentation by Smith [10]. Its proof is repeated here in some detail, however, because it clearly demonstrates the simulation technique utilized throughout this paper and introduces easily the idea of "pseudosupport" and the "end-of-tape" problem, two concepts frequently encountered in the remainder of the paper.

THEOREM 1. *For an arbitrary* (m, n) *Turing machine* T, *there exists a 2-D, 7-neighbor,* max $(m + 1, n + 1)$*-state cellular space* Z_T *which simulates it in real-time.*

PROOF. Each cell of Z_T is provided with a set Q of $M = \max(m + 1, n + 1)$ states. Without loss of generality, let $Q = \{0, 1, \cdots, M - 1\}$ so that $(i + 1)$ corresponds to symbol x_i of T for $0 \leq i \leq m - 1$ and state $(j + 1)$ corresponds to Turing machine state q_j for $0 \leq j \leq n - 1$. 0 is the quiescent state of Z_T and never corresponds to a Turing machine state or symbol. The geometry of Z_T will be utilized to distinguish a cell whose state $Q_1 \in A = \{1, \cdots, m\}$ corresponds to a Turing machine symbol from a cell whose state $Q_2 \in B = \{1, \cdots, n\}$ corresponds to a Turing machine state. In particular, Z_T has the neighborhood template shown in Figure 2(a).



(a)



(b)

FIG. 2

We cause Z_T to simulate T by embedding a configuration in it which "looks like" T. That is, one row of cells in Z_T is the "tape" of the embedded Turing machine—one cell of Z_T per tape square of T—and one cell in an adjacent row is the "head." Thus the embedded Turing machine configuration will have the form indicated in Figure 2(b) at any one instant. As indicated in Figure 2(b), a and b are always labels for the cells to the left and right, respectively, of the head cell h. All other symbols are state assignments: $S_k \in A$ is always the state of the tape cell at distance $|k|$ from the scanned cell in the direction determined by the sign of k as indicated; and $P \in B$ is the state of head cell h. C_X is used to designate the cell immediately to the $X \in \{R, L\}$ of a finite embedded tape. All cells other than the head and tape cells are assumed to be in the quiescent state 0. Thus C_R and C_L are always in state 0.

Head cell h is made to "move" along the tape subconfiguration simulating the head moves of T by suitable specification of the transition function f for a cell in Z_T. This is simply done. Unless the cell is a, b, h, s, C_R, or C_L, it does not change state. For these six cases, let the Turing machine state-table entry for symbol x_u and state q_v be denoted (x_u, q_v). Then f is given for cells a, b, h, s, C_R, and C_L as indicated in Table I when $(x_u, q_v) = x_p X / q_q$. The last two entries require some explanation.

We must distinguish the blank symbol from the quiescent state in this simulation, else a tape symbol in the tape row could act as a head for the next row of cells interpreted as an entirely blank tape. This creates the *end-of-tape problem*: The simulated Turing machine requires, in general, an infinite tape, but the initial configuration is assumed to have finite support. Hence the last two entries are "tape extenders" which convert the quiescent state to the blank symbol at either end of the necessarily finite embedded tape configuration.

The description of the encoding function γ for the simulation of T by Z_T is completed as follows: There is a state $w \in B$ in each cell of Z_T which corresponds to the starting state of T. The nonblank portion of the (finite) initial tape of T is embedded

TABLE I

Cell C	*Neighborhood state of C*			*Next state of C*	*Conditions*
		P			$S_0 = u + 1$
s	S_{-1}	S_0*	S_1	p	$p \in A$
	0	0	0		$p = v + 1$
		0			
h	0	P*	0	0	in all cases
	S_{-1}	S_0	S_1		
		0			
a	0	0*	P	$\begin{cases} 0 \\ (q+1) \end{cases}$	if $X = R$ if $X = L$, $(q+1) \in B$
	S_{-2}	S_{-1}	S_0		
		0			
b	P	0*	0	$\begin{cases} (q+1) \\ 0 \end{cases}$	if $X = R$ if $X = L$
	S_0	S_1	S_2		
		0			
C_R	S_k	0*	0	1	$1 \in A$ is the "blank" symbol
	0	0	0		
		0			
C_L	0	0*	S_k	1	as for C_R
	0	0	0		

* State of cell C.

in one row of Z_T. The cell above the cell corresponding to the leftmost nonblank square is set to state w and hence represents the initial position of the tape head of T. Then the global transition function F causes the cellular tape subconfiguration to be modified (in a real-time simulation) just as would be the tape of T.

Define the *pseudosupport* of configuration c to be the set psup (c) of all nonquiescent and nonblank cells in c. Then the procedure π for determining completion of a computation is defined by $\pi(c_t, c_{t+1}) = 1$ if and only if $c_t \mid \text{psup } (c_t) = c_{t+1} \mid \cdot$ psup (c_{t+1}) (cf. Remark 2 after Definition 1). Q.E.D.

Remark. It is necessary to distinguish the quiescent state from the state simulating the blank symbol in the proof above in order to get a small state count. Clearly this distinction would not have to be made if we gave $m + n$ states to Z_T, m states to simulate Turing machine states and n states to simulate Turing machine symbols, or if we gave mn states to Z_T, each with two coordinates. In the latter case, Z_T need have only one dimension and this is the simulation given by Balzer [2].

We contrast the construction in the proof above, in which the cell design depends on the Turing machine to be simulated, with the cellular spaces of von Neumann, Thatcher, Codd, and Arbib, in which any Turing machine can be simulated once the cell design is set. The difference between the Turing-machine-dependent cell constructions and the Turing-machine-independent cell constructions is unimportant when the simulated Turing machine of interest is the universal machine, as, for example, in the corollaries below. In this case the machine-dependent cells are clearly superior in the sense that all simulations are real-time (or "almost" real-time, as will be specified in other theorems to follow) as opposed to the very slow simulations in, say, the von Neumann space in which each simulated step requires routing of signals through hundreds and even thousands of cells. Of course, a real-time simulation of a universal Turing machine is not real-time with respect to the original Turing machine simulated by the universal machine.

COROLLARY 1.1. *There exists a* max $(m + 1, n + 1)$*-state computation-universal cellular space for every* (m, n) *universal Turing machine.*

COROLLARY 1.2. *There exists a 2-D, 7-state computation-universal cellular space.*

PROOF. Minsky [7] has cited a (6, 6) universal Turing machine. Q.E.D.

Previous work with cellular spaces has often employed the 5-cell von Neumann, or H_1, neighborhood of Figure 1(a). The 7-neighbor spaces of the type used in proving the theorem above can be replaced easily with spaces having the H_1 neighborhood, as the next theorem indicates.

THEOREM 2. *For an arbitrary* (m, n) *Turing machine* T, *there exists a 2-D,* max $(2m + 1, 2n + 2)$*-state cellular space* Z_T *with the* H_1 *neighborhood which simulates it in 3 times real-time.*

PROOF. The configuration in Z_T which simulates an instantaneous description of T occupies two rows of Z_T as in the proof of Theorem 1, but the information about head moves stored in the 7-cell neighborhood is here encoded into an enlarged state set. In particular, for $b \in \{0, 1\}$, Z_T will have state s_{ib} for each state q_i of T and state s_{jb} for each symbol x_j of T. Suppose Z_T is to simulate a step of T in state q_i and scanning symbol x_j, with corresponding state table entry $x_u X / q_v$. Thus, for some time t, depending on whether the last move was right or left respectively, there will be a configuration in Z_T of the form:

$$\begin{array}{ccc} \cdots\ 0\ \ @s_{i0}0\ \ 0\ \cdots & & \cdots\ 0\ \ 0\ \ s_{i0}@0\ \ \cdots \\ & \text{or} & \\ \cdots\ s_{w0}s_{x1}s_{j0}s_{y0}s_{z0}\ \cdots & & \cdots\ s_{w0}s_{x0}s_{j0}s_{y1}s_{z0}\ \cdots. \end{array}$$

The top row is the "head" row and the bottom row is the "tape" row. The purpose of special state @ will become clear below where the steps in the simulation of one step of T are pictured. Subscripts $b = 0$ and $b = 1$ correspond to $X = R$ and $X = L$ respectively:

$$\begin{array}{lll} t+1: & \cdots\ 0\ 0\ s_{v1}0\ 0\ \cdots & \\ & \cdots\ s_{w0}s_{x0}s_{ub}s_{y0}s_{z0}\ \cdots & \\ t+2: & \cdots\ 0\ @s_{vb}@0\ \cdots & \\ & \cdots\ s_{w0}s_{x0}s_{u1}s_{y0}s_{z0}\ \cdots & \\ t+3: & \cdots\ 0\ 0\ @s_{v0}0\ \cdots & \\ & s_{w0}s_{x0}s_{u1}s_{y0}s_{z0}\ \cdots & \text{if } b = 0, \\ t+3: & \cdots\ 0\ s_{v0}@0\ 0\ \cdots & \\ & \cdots\ s_{w0}s_{x0}s_{u1}s_{y0}s_{z0}\ \cdots & \text{if } b = 1. \end{array}$$

Thus at $t + 3$ the space is ready to simulate another step of T; hence the simulation proceeds in 3 times real-time. It is a simple matter to specify a transition function f to accomplish this simulation, so we leave the details to the interested reader. Note that f must handle the end-of-tape problem as in Theorem 1. Q.E.D.

A consequence of Theorem 2 is a 5×14 2-D computation-universal cellular space obtained by applying the theorem to the (6, 6) universal Turing machine.[1] Here the notation $p \times q$ *cellular space* is short for p-neighbor, q-state cellular space. This might be compared for simplicity to the 5×8 2-D CUCS of Codd [3], although such a comparison is difficult. The 14-state computation-universal configuration occupies two rows of its space and simulates a Turing machine in 3 times real-time, whereas the 8-state configuration covers hundreds of rows of its space and simulates in time on the order of n^2 times real-time, where n is the length of the encoded program and tape of the simulated Turing machine.

Utilization of only two rows of a 2-dimensional space implies immediately the existence of a 3×196 1-D CUCS: Use a state set with two coordinates of 14 values each; they simulate the head and tape rows respectively. But Theorem 3 does better for the 1-D case.

THEOREM 3. *For an arbitrary* (m, n) *Turing machine* T, *there exists a* 1-*D*, 6-*neighbor*, max $(m + 1, n + 1)$-*state cellular space* Z_T *which simulates it in real-time.*

PROOF. Let Z_T have the 6-cell neighborhood template of Figure 3(a). The embedded Turing machine configuration is as illustrated in Figure 3(b); the tape squares occupy every other cell in the space. The transition function f leaves the state of a cell C unchanged except in the six cases a, b, s, h, C_R, and C_L (as in the proof of Theorem 1). The function f is easily specified and is omitted here. It should be noted that tape extenders are required here as in Theorem 1 to convert quiescent cells to cells simulating the blank symbol at the tape "ends." Q.E.D.

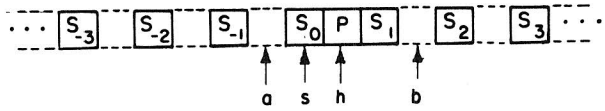
COROLLARY 3.1. *There exists a* 1-*D*, 6-*neighbor*, max $(m + 1, n + 1)$-*state computation-universal cellular space for every* (m, n) *universal Turing machine* T.

COROLLARY 3.2. *There exists a* 1-*D*, 6×7 *computation-universal cellular space.*

PROOF. Let T be Minsky's (6, 6) universal Turing machine. Q.E.D.

[1] A very simple proof exists for Theorem 2 with Z_T having max $(m + 1, 3n + 1)$ states. This yields, however, a 5×19 CUCS. We leave discovery of this proof to the interested reader.

(a)



(b)

FIG. 3

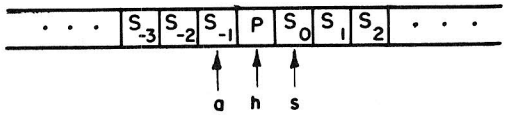


FIG. 4

Theorem 2 is a neighborhood reduction result for the special case of computing cellular spaces in 2-D. Similarly, the next theorem is a specialized neighborhood reduction result in 1-D.

THEOREM 4. *For an arbitrary* (m, n) *Turing machine* T, *there exists a* 1-*D*, $(m + 2n)$-*state*, 3-*neighbor cellular space* Z_T *which simulates it in at most* 2 *times real-time, with the* 1-*D* H_1 *template* [*see Figure* 1 (*b*)].

PROOF. Provide Z_T with the 1-D H_1 template, and embed a Turing machine configuration in Z_T as indicated in Figure 4. The transition function f leaves the state of all cells unchanged except in the cases a, h, and s. It is a simple matter to fill in the details of this function such that tape configurations like $\cdots x_0x_1qx_2x_3 \cdots$, simulating a right move into new state q' after changing symbol x_2 to x_2', appear in time as

$$\cdots x_0x_1qx_2x_3 \cdots$$
$$\cdots x_0x_1x_2'q'x_3 \cdots .$$

Similarly, a left move looks like

$$\cdots x_0x_1qx_2x_3 \cdots$$
$$\cdots x_0x_1q_L'x_2'x_3 \cdots$$
$$\cdots x_0q'x_1x_2'x_3 \cdots .$$

Thus two states, q and q_L, are needed to represent each Turing machine state. In this case, the blank symbol is simulated by the quiescent state; hence there is no end-of-tape problem (cf. Remark after Theorem 1). Q.E.D.

Either the (6, 6) universal Turing machine or the (4, 7) universal Turing machine (also due to Minsky [7]) and Theorem 4 produce the next result.

COROLLARY 4.1. *There is a* 1-*D*, 3×18 *computation-universal cellular space.*

There is nothing sacred about the von Neumann (H_1) neighborhood, of course. In 1-D spaces, the H_1 neighborhood can be reduced to the K_1 neighborhood [see Figures 1(b) and 1(d)] quite readily, as the next theorem attests.

THEOREM 5. *For an arbitrary* (m, n) *Turing machine* T, *there exists a* 1-D, $m(n + 3)$*-state cellular space* Z_T *with the* 2*-cell* K_1 *template which simulates* T *in* 2 *times real-time.*

PROOF. This proof proceeds much like that of Theorem 4. The major difference is that the simulated instantaneous descriptions of T shift in time through the space Z_T. Thus an instantaneous configuration such as $\cdots x_0x_1qx_2x_3x_4 \cdots$ is simulated by state pairs in Z_T as indicated below for left and right moves respectively:

$$
\begin{aligned}
L: \quad & \cdots (x_0, 0)(x_1, 0)(x_2, q)(x_3, 0) \cdots \\
& \cdots (x_1, 1)(x_2', q')(x_3, 2)(x_4, 1) \cdots \\
& \cdots (x_1, q')(x_2', 0)(x_3, 0)(x_4, 0) \cdots
\end{aligned}
$$

$$
\begin{aligned}
R: \quad & \cdots (x_0, 0)(x_1, 0)(x_2, q)(x_3, 0) \cdots \\
& \cdots (x_1, 1)(x_2', 2)(x_3, q')(x_4, 1) \cdots \\
& \cdots (x_1, 0)(x_2', 0)(x_3, q')(x_4, 0) \cdots .
\end{aligned}
$$

Any q must be distinct from 0, 1, and 2. Hence the first element of each state pair has m values and the second has $n + 3$ values. The reader is invited to complete the formal simulation from this sketch. Q.E.D.

Remark. Here the definition of "c computes g" (Definition 1) is not valid as stated, because the computing configuration c is not disjoint from the tape configuration d_n. However, the coordinates of the state space representing c are distinct from those representing d_n. When this is the case for a 1-D space, then the space can always be interpreted as two rows of a 2-D space as in Theorem 1. Under this interpretation, the definition is valid. The computing cellular spaces of the type described in [2] (see Remark after Theorem 1) also require this interpretation of Definition 1. The definition can be formally generalized [11] for those cellular spaces, but we have chosen not to do so for simplicity.

The (4, 7) universal machine yields the following corollary.

COROLLARY 5.1. (*a*) *There exists a* 1*-D,* 2*-neighbor computation-universal cellular space.* (*b*) *In particular, there is a* 1*-D,* 2×40 CUCS.

We now prove Corollary 5.1(a) again but in a much different way. Of course, the purpose of the next theorem is not the re-proof of this corollary but rather the introduction of another tool for the study of computation in cellular space: simulation of Post tag systems. Kilmer [4] has proved several unsolvability results in his study of finite sequential iterative networks by designing networks to simulate Post tag systems. We find similar results for cellular spaces—i.e. infinite sequential iterative networks—also by simulation of tag systems. However, the simulation technique used below differs substantially from that of Kilmer. His cell design must be changed if the initial string (axiom) of the simulated tag system changes. We are able to avoid this in the cell designs below and at the same time use fewer states per cell.

The general tag system (see Minsky [7] for details) to be simulated has alphabet $X = \{x_1, x_2, \cdots, x_m\}$, deletion number P, and associated with the m possible initial letters there are respectively m consequents $C_1, C_2, \cdots, C_m$ of lengths $a_1, a_2, \cdots, a_m$, respectively. The axiom is an arbitrary finite string of letters in X.

THEOREM 6. *For an arbitrary Post tag system T, there exists a 1-D, 2-neighbor cellular space Z_T with the K_1 template which simulates it in $(P + 1)$ times real-time.*

PROOF. Z_L is chosen to have state set $K \times I$, where $K = \{0, 1, 2, \cdots, m, m + 1, m + 2, \cdots, m + P, m + P + 1, \cdots, m + P + a\}$, $a = \max_{i \in I} a_i$, and $I = \{0, 1, \cdots, m\}$. Then states of the form (k, i), $m + 1 \leq k \leq m + P$, $i > 0$, are marker states which, in effect, remove the first P letters of a string in the tag system which is (initially) represented by a string of cells in the cellular space. The state $(0, 0)$ is the quiescent state. States $(k, 0)$, $1 \leq k \leq m$, represent letters in the axiom. States (k, i), $1 \leq k \leq m$, $i > 0$, represent letters in the string after the first production; the index i indicates which consequent should be appended to the present string. States (k, i), $m + P + 1 \leq k \leq m + P + a$, $i > 0$, are consequent-C_i-generating states.

The simulation proceeds as follows: The axiom is coded as a string of cells where letter x_i in the axiom is represented by a cell in state $(i, 0)$. The transition function f of the space is such that this string of states always moves left at one cell per time step. At the left end of the string, however, a marker state is set up which does not propagate left. If the leftmost letter of the string is x_j, then the first marker state is $(m + 1, j)$. As the string moves left into this marker state, the state "gobbles up" the first P letters—i.e. its first element increases by one at each step until marker state $(m + P, j)$ is reached. The string continues to propagate left, but the fact that the first letter of the antecedent just cut off was x_j is recorded in the second element of the state as a j. This information is not propagated left but is held stationary until the right end of the string advances to the cell holding this index. Here the consequent-C_j-generating state $(m + P + 1, j)$ is set up. It remains stationary, only increasing its first element by one at each step until state $(m + P + a_j, 1)$ is reached. This set of a_j states essentially feeds consequent C_j to the left, one element per time step. Meanwhile a new marker state has been set up to remove P more letters from the left end of the left-propagating string.

Every $P + 1$ steps there will exist (if termination has not occurred) a string of states of the form

$$(k_1, i_1)(k_2, i_2) \cdots (k_p, i_p) \cdots (k_n, i_n),$$

where $1 \leq k_q \leq m$ for $1 \leq q \leq n$. If $i_p \neq 0$ but $i_q = 0$ for all $q > p$, then this string corresponds in the simulated tag system to string

$$x_{k_1} x_{k_2} \cdots x_{k_n} C_{i_p} \cdots C_{i_2} C_{i_1}.$$

If $i_n \neq 0$, then $k_n \geq m + P + 1$ and the state string corresponds to tag string

$$x_{k_1} x_{k_2} \cdots x_{k_{n-1}} C_{i(k_n - m - P)} C_{i(k_n - m - P + 1)} \cdots C_{i a_i} C_{i_{n-1}} \cdots C_{i_2} C_{i_1},$$

where C_{ij} is the jth component of consequent C_i. Thus this is the algorithm for recovering the tag system string every $P + 1$ time steps.

The transition function f which performs the simulation outlined above is given in Table II. All other cases are assumed passive. Q.E.D.

Remark 1. No attempt was made to minimize the number of states in the construction above. Hence the $(m + 1)(m + P + a + 1)$ states required are probably extragavant. There are several states never used—e.g. states $(k, 0)$, $m + P + 1 \leq k \leq m + P + a$.

Remark 2. Essentially the same proof works for monogenic normal systems.

TABLE II

Neighborhood state of cell C	*Next state of C*	*Conditions*
$(0, 0)^*(k', i')$	$(m + 1, k')$	$1 \leq k' \leq m$
$(k, 0)^*(k', 0)$	$(k', 0)$	$0 \leq k' \leq m, 1 \leq k \leq m$
$(k, i)^*(k', i')$	$(k + 1, i)$	$m + 1 \leq k \leq m + P, i > 0$
$(m + P, i)^*(0, 0)$	$(0, 0)$	$i > 0$
$(m + P, i)^*(k', i')$	(k', i)	$1 \leq k' \leq m$
	(k'', i)	$m + P + 1 \leq k' \leq m + P + a_{i'}$, $x_{k''} = C_{i'(k' - m - P)}$, $i, i' > 0$
$(k, i)^*(0, 0)$	$(k + 1, i)$	$m + P + 1 \leq k \leq m + P + a_i - 1, i > 0$
	$(0, 0)$	$k = m + P + a_i, i > 0$
$(k, i)^*(k', i')$	(k', i)	$1 \leq k, k' \leq m, i > 0$
$(k, i)^*(0, 0)$	$(m + P + 1, i)$	$i \leq k \leq m, i > 0$
$(k, i)^*(k', i')$	(k'', i)	$1 \leq k \leq m, i, i' > 0, m + P + 1 \leq k' \leq m + P + a_{i'}, x_{k''} = C_{i'(k' - m - P)}$

* State of cell C.

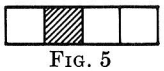


FIG. 5

2ND PROOF OF COROLLARY 5.1(a). Cocke (see Minsky [7, p. 270]) has shown the existence of a Post tag system for simulating a given arbitrary Turing machine. In particular, design Z_T to simulate a tag system T which simulates a universal Turing machine. Q.E.D.

Since a 1-cell neighborhood template offers only finite memory, it is clear that the 2-cell K_1 template used, for example, in Corollary 5.1 is the minimal neighborhood possible for computation universality. Thus, for $p \times q$ CUCS, $p = 2$ is the minimum p. What is the minimum q? What is the minimum product pq? The answer to the first question is $q = 2$, from the general theory of state set reduction. The second question is not so easy. Establishing the minimum product pq is a problem which is probably as difficult as finding the minimum product mn for (m, n) universal Turing machines, but we can establish an upper bound on the minimum pq by applying general cellular automata results. Corollary 7.1 below, which gives this bound, also lists the "sizes" in which CUCS are available, and the different spaces illustrate some of the various tradeoffs possible between state-set size and neighborhood size.

THEOREM 7. *For an arbitrary (m, n) Turing machine T, there exists a* 1-D, 4-*neighbor, $(m + n)$-state cellular space Z_T which simulates it in real-time.*

PROOF. Give Z_T the template of Figure 5. Then the simulation of T proceeds in a manner very similar to that in the proof of Theorem 4, but here the neighborhood is large enough so that the simulation occurs in real time. We leave the details of this proof to the reader. Q.E.D.

COROLLARY 7.1. *There exist* 1-*D $p \times q$ computation-universal cellular spaces with $p \times q = (1)$ 4×11, (2) 6×7, (3) 8×5, (4) 9×4, (5) 13×3, and (6) 21×2.*

PROOF. Let T in Theorem 7 be the (4, 7) universal Turing machine to obtain result (1). Result (2) is just the result stated in Corollary 3.2. The other results are consequences of the following lemma from the general theory [11]. It is stated here without its proof, which requires a great amount of additional material.

STATE REDUCTION LEMMA. *For any 1-D $p' \times q'$ contiguous cellular space Z', there is a 1-D $p \times q$ cellular space Z which simulates Z' in real-time, where $p = N(p' + 1) + k - 1$, N is the smallest integer such that $q' \leq q^N - N$, and k is the smallest integer such that $N \leq q^k - 2^k + 2k - 1$.*

The concept underlying the lemma is the relatively simple idea of a q-ary, N-digit encoding of a configuration in Z' cell-by-cell into Z. Another code, of length k, is required to position each encoded state of Z' in Z. It is this additional code which complicates the proof of the lemma; hence the proof is delayed to a forthcoming paper [12]. See also [13].

Apply the state reduction lemma to the 3×18 CUCS of Corollary 4.1 to get result (3). Apply it to the 2×40 CUCS of Corollary 5.1 to get (4), (5), and (6). Q.E.D.

Thus Corollary 7.1 gives 36 as the upper bound on the minimum product pq. A lower bound is, of course, 4.

The binary CUCS in the corollary above is not the first binary CUCS to be exhibited, but it is the first in one dimension. Codd [3] presents an 85×2 CUCS in 2-D obtained by simulating his 5×8 CUCS (also a 2-D space). We note that a d-D state reduction theorem of Smith [11] would yield a 2-D binary CUCS with at most 38 neighbors when applied to the 5×8 space.

Simple computation-universal cellular spaces are not the only byproducts of the simulation method. We now use it to answer several questions raised by Lieblein [6]. In his terminology, a cellular space as we have defined it is called a *simple axiom system*. The "axiom" is the initial configuration, and "simple" means only one rule of inference—i.e. one local transition function. In particular, Lieblein studies a special case called a *PC-simple axiom system* which is a binary cellular space with initial configuration restricted to be the *primitive configuration* (PC) c_{pc} defined (for $m \in \mathbf{Z}^d$) by

$$c_{pc}(m) = \begin{cases} 1, & \text{if } m \text{ is the origin,} \\ 0, & \text{else,} \end{cases}$$

where 0 is the quiescent state. Thus, in a simple axiom system, each configuration in the propagation of the axiom can be thought of as a theorem. Another view is to treat each of these configurations as a word in a language. The language is finite if the cellular space producing its words becomes passive or cyclic; else it is infinite.

THEOREM 8. *For an arbitrary (m, n) Turing machine T on an initially blank tape, there exists a 1-D contiguous $p \times m$ cellular space Z_T' which simulates T with $c_0 = c_{pc}$.*

PROOF. Assume that the symbol set of T is $\{0, 1, 2, \cdots, m - 1\}$. We require that no subconfiguration can be interpreted by a cell as part of c_{pc}. This is accomplished by the way tape symbols are encoded in the cellular space. In particular, let a symbol i be coded in Z_T as three adjacent cells in states 110. The state of T is simulated by an m-ary N-tuple represented by the states of N cells in Z_T, where N is the integer just greater than or equal to $\log_m n$. Delineate the N "head" cells by three adjacent cells in states 111 on the left and three adjacent cells in states 111 on the right. Thus Z_T might be programmed as in Figure 6(a) for the binary case. Then it is not difficult to check that the template of Figure 6(b) has enough information for simulation of T.

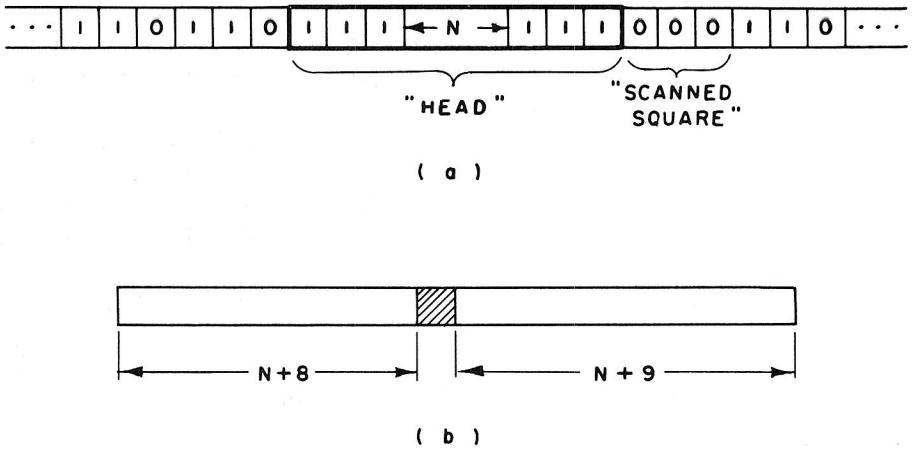


FIG. 6

Thus, during the first time step, transition function f' of the desired cellular space Z_T' sets up the head described above by recognizing c_{pc}. By design, this can only happen at $t = 0$. Note that the blank tape is automatically set up because 0 is coded by cells in states 000 only. Then at the second time step, f' proceeds to function just as would transition function f of the cellular space Z_T described above. Q.E.D.

COROLLARY 8.1. *There does not exist an algorithm that will determine, for a given 1-D PC-simple axiom system, if the system represents a finite or an infinite language.*

PROOF. Let T be a Turing machine which, if started on blank tape, has an unsolvable halting problem (see [7, p. 150]). Let T_2 be the $(2, n)$ equivalent of T. Then design $p \times 2$ cellular space Z'_{T_2} as in the proof of Theorem 8. It is undecidable then whether the language of Z'_{T_2} is finite or infinite. See [9] for T_2. Q.E.D.

COROLLARY 8.2. *It is undecidable, given a configuration c and a 1-D PC-simple axiom system Z, whether c is a theorem of Z.*

PROOF. Let T_2 be the 2-symbol equivalent of T, a Turing machine for which it is undecidable whether it will ever print a given symbol [7, p. 150]. Then Z is Z'_{T_2} which simulates T_2 as in the proof of Theorem 8. Q.E.D.

COROLLARY 8.3. *It is undecidable, given an arbitrary 1-D simple axiom system with global transition function F and any two configurations c and c', whether or not there exists an m such that* $F^m(c) = c'$.

PROOF. Design Z_{T_2} as in the first paragraph of the proof of Theorem 8 for T_2 as in the proof of Corollary 8.2. Q.E.D.

Summary and Conclusions

Several methods are presented above for obtaining substantial computing power from quite simple cellular automata. In particular, one dimension is shown to be sufficient for universality on the set of partial recursive functions. A cellular automaton with this power is called a computation-universal cellular space (CUCS). We have demonstrated several examples of such spaces which are "simple" in the sense that their neighbor-state product is small (e.g. Corollary 7.1 presents a CUCS with this product equal to 36). Hence we conclude that early work with cellular auto-

mata uses spaces of undue complexity. Most notable of these is the 5-neighbor, 29-state CUCS in [15] which requires two dimensions.

Several techniques for simulating Turing machines and tag systems by cellular automata are also described above (in which cases we say the Turing machine, or tag system, is "wired in"). Our work is related to that of Wagner [16], who in effect wires "spider automata" into cellular spaces (with Mealy-type cells), because a degenerate spider is a Turing machine. These techniques, coupled with results from the general theory of cellular automata, lead in two different ways to the proof of the existence of a CUCS with only two neighbors. This is at first a surprising result since computation universality implies a two-way flow of information whereas a 2-neighbor space seems to be capable of only a one-way flow.

Finally, the techniques are used to obtain several unvolvability results which arise when a cellular automaton is employed as a language generator.

ACKNOWLEDGMENT. I wish to extend my warm thanks to Michael Arbib who introduced me to the biological implications of cellular automata. This paper lays some groundwork for investigation of these implications and is a direct result of his inspiration.

REFERENCES

1. ARBIB, M. A. A simple self-reproducing universal automaton. *Inform. Contr. 9* (1966), 188–189.
2. BALZER, R. M. Studies concerning minimal time solutions to the firing squad synchronization problem. Ph.D. dissertation, Carnegie Inst. of Tech., Pittsburg, Pa., 1966.
3. CODD, E. F. *Cellular Automata.* ACM Monograph Series, Academic Press, New York, 1968.
4. KILMER, W. L. On dynamic switching in one-dimensional iterative logic networks. *Inform. Contr. 6* (1963), 399–415.
5. LEE, C. Y. Synthesis of a cellular computer. In *Applied Automata Theory*, J. T. Tau, Ed., Academic Press, New York, 1968, pp. 217–234.
6. LIEBLEIN, E. A theory of patterns in two-dimensonal tessellation space. Ph.D. dissertation, University of Pennsylvania, Philadelphia, Pa., 1968.
7. MINSKY, M. *Computation: Finite and Infinite Machines.* Prentice-Hall, Englewood Cliffs, N. J., 1967.
8. ROGERS, H., JR. *Theory of Recursive Functions and Effective Computability.* McGraw-Hill, New York, 1967.
9. SHANNON, C. E. A universal Turing machine with two internal states. In *Automata Studies*, C. E. Shannon and J. McCarthy, Eds., Princeton U. Press, Princeton, N. J., 1956.
10. SMITH, A. R., III. Simple computation-universal cellular spaces and self-reproduction. Proc. IEEE Ninth Annual Switching and Automata Theory Symp., pp. 269–277.
11. SMITH, A. R., III. Cellular automata theory. Tech. Rep. 2, Digital Systems Lab., Stanford U., Stanford, Calif., 1970.
12. SMITH, A. R., III. Cellular automata complexity trade-offs. *Inform. Contr. 18* (1971) (In press).
13. SMITH, A. R., III. General shift-register sequences of arbitrary cycle length. *IEEE Trans. Comp. C-20* (1971), 456–459.
14. THATCHER, J. W. Universality in the von Neumann cellular model. Tech. Rep. 03105-30-T, ORA, U. of Michigan, Ann Arbor, 1964.
15. VON NEUMANN, J. *The Theory of Self-Reproducing Automata*, A. W. Burks, Ed. U. of Illinois Press, Urbana, Ill. 1966.
16. WAGNER, E. G. An approach to modular computers I: Spider automata and embedded automata. Tech. Rep. IBM RC 1107, IBM Watson Research Lab., Yorktown Heights, N.Y., 1964.

RECEIVED MARCH 1970; REVISED FEBRUARY 1971