Math. Systems Theory 20, 43-52 (1987)

Mathematical Systems Theory



# On the Computational Power of Totalistic Cellular Automata

Dan Gordon*

Department of Mathematics and Computer Science, University of Haifa, Haifa 31999, Israel

**Abstract.** Totalistic cellular automata, introduced by S. Wolfram, are cellular automata in which the state transition function depends only on the sum of the states in a cell's neighborhood. Each state is considered as a nonnegative integer and the sum includes the cell's own state. It is shown that one-dimensional totalistic cellular automata can simulate an arbitrary Turing machine in linear time, even when the neighborhood is restricted to one cell on each side. This result settles Wolfram's conjecture that totalistic cellular automata are computation-universal.

## 1. Introduction

Since their introduction by von Neumann [16], cellular automata (or cellular spaces) have been studied very extensively. For the classical theory of cellular automata, see [1]-[3], [13], and [16]. This model of computation is well suited to simulation of biological systems [7], has given rise to the theory of L-systems [14], and has been used to model physical systems [18 and other papers in same book]. Needless to say, computational problems such as decidability, computability, language acceptance, and computational complexity have all been extensively studied and are the subject of ongoing research—see, for example, [9], [10], and [15].

Recently, Wolfram has introduced the concept of a totalistic cellular atomaton [17], [18]. In this model, an automaton state is considered as a nonnegative integer and the state transition function depends only on the sum of the states in a cell's neighborhood. The sum includes the cell's own state. Wolfram studied

* Research performed while visiting the Department of Computer Science, University of Cincinnati, 1984/85.

totalistic automata in one dimension and, based on an extensive statistical evaluation, he conjectured that totalistic automata are computation-universal. For a short, informal introduction to totalistic automata see [4].

In this paper we prove Wolfram's conjecture by showing that totalistic automata can simulate any Turing machine in linear time. It is well known that cellular automata can simulate an arbitrary Turing machine [1]-[3], [13], [16], and Smith [15] has given simple constructions for the one- and two-dimensional cases. However, in these simulations, the cellular automata are nontotalistic.

One well-known example of a simple cellular space which can simulate any Turing machine is the game of "Life" [5], [6]. This two-dimensional space has only two states, but it is not totalistic, since the state transition function depends both on the cell's state and the sum of its neighbors' states (we shall call such spaces semitotalistic). The computation-universality of "Life" thus still leaves open the questions of semitotalistic spaces in one dimension and totalistic spaces in one and two dimensions.

With the advent of VLSI technology [11], [12] it has become feasible to implement very large one- and two-dimensional arrays of interconnected processing elements. In one dimension, each processing element is connected to its two immediate neighbors, and in two dimensions we can have either a square grid pattern of interconnections or a hexagonal pattern, where each element is connected to six neighbors. The implications of this technology are that cellular spaces can be implemented in VLSI, but that this can be done more easily when an automaton's neighborhood contains only its immediate neighbors. This restriction is observed by our computation model.

The rest of the paper is organized as follows: Section 2 gives the basic definitions; Section 3 contains the main simulation result; Section 4 discusses the semitotalistic case; and Section 5 concludes with some open questions.

## 2. Definitions

Our definitions will follow closely those of Codd [3] and Smith [15]. Let $Z$ denote the set of integers and $|A|$ the cardinality of a set $A$.

**Definition 1.** A one-dimensional cellular space is a 4-tuple $(N, V, v_0, f)$ where $N$ is a finite set of integers called the *neighborhood template*; $V$ is a finite set of *states*; $v_0 \in V$ is called the *quiescent state*; and $f$: $V^{|N|} \to V$, called the *state transition function*, satisfies $f(v_0, \ldots, v_0) = v_0$.

We can view a cellular space as an infinite set of cells, one cell for each $i \in Z$, which we denote by CELL($i$). The neighborhood of CELL($i$) is the set of cells $\{\text{CELL}(i+n) | n \in N\}$. Usually, the neighborhood will include the cell itself, i.e., $0 \in N$. A *configuration* $c$ is some particular assignment of states to cells: $c$: $Z \to V$; $c(i)$ can be viewed as the state of CELL($i$). In the following, we assume that $n_1, n_2, \ldots, n_{|N|}$ are the elements of $N$ in increasing order.

**Definition 2.** Let $(N, V, v_0, f)$ be a cellular space. The *global transition function* $G$ is defined as a mapping from the set of all configurations to itself as follows: let $c$ be a configuration, then $G(c)$ is the configuration obtained from $c$ by changing the state of each cell according to the transition function $f$ applied to each cell's neighborhood: $G(c)(i) = f(c(i+n_1), c(i+n_2), \ldots, c(i+n_{|N|}))$.

**Definition 3.** Let $(N, V, v_0, f)$ be a cellular space such that $0 \in N$, $V$ is a set of nonnegative integers and the quiescent state is $v_0 = 0$.

(a) The cellular space is called *totalistic* if there exists a function $g$ of one variable such that for all $v_1, \ldots, v_{|N|} \in V$, $f(v_1, \ldots, v_{|N|}) = g(\sum_{i \in N} v_i)$.
(b) The cellular space is called *semitotalistic* if there exists a function $g$ of two variables such that $f(v_1, \ldots, v_{|N|}) = g(v_k, \sum_{i \in N-\{k\}} v_i)$, where $k$ is 0's position among $N$'s elements (in increasing order).

We shall be mainly interested in totalistic cellular spaces in which a cell's neighborhood consists of itself and its two immediate neighbors, i.e. $N = \{-1, 0, 1\}$. We call such a space a *three-neighbor cellular space.*

We assume that the reader is familiar with the concept of a two-way tape Turing machine [8]. Recall that an instantaneous description (ID) is a coding of the nonblank tape symbols, the machine's internal state, and the position of its read/write head.

We also need the concept of a Turing machine simulation by cellular automata in $k$ times real time ($k$ an integer). Briefly, it means that successive IDs of the Turing machine are coded by every $k$th configuration of the cellular space. The formal definition appears in [15], and it will also be obvious from our construction how to formalize this notion.

## 3. Turing Machine Simulation by Totalistic Automata

### *3.1. Outline*

Let M be an $(m, n)$ Turing machine [15], i.e., M has $m$ tape symbols and $n$ states. We shall simulate IDs of M by configurations of a cellular space as follows: squares of M's tape (containing tape symbols) will be coded onto alternate cells of the cellular space. The in-between cells will have a special intermediate state designed so that the tape symbols will remain unchanged in successive configurations (unless M's read/write head moves over them). For convenience, we shall refer to cells which code M's tape squares as *primary cells* and to the others as *secondary cells.*

M's read/write head will be simulated by three adjacent cells, the middle one being the primary cell over which the head is positioned. The adjacent secondary cells will code both M's state and the scanned tape symbol, so that their other primary neighbors can determine M's next move. However, the coding will be different for the left and right secondary cells, enabling the adjoining primary cells to tell if the head is to their left or right.

The initial configuration will code M's initial ID by a finite number of nonzero cells; all other cells will be in the quiescent state 0. This conforms to the usual assumption that in an initial configuration, all but a finite number of cells are in the quiescent state.

Since tape cells are separated by intermediate cells, consecutive IDs cannot be simulated by consecutive configurations. After every "primary" configuration simulating an ID, there is an "intermediate" configuration designed to pass information to the left or right (according to the head's move) so that the next configuration will again be a "primary" one simulating M's next ID.

### *3.2. Coding of Tape Symbols*

The $m$ tape symbols (when not scanned by M's head) will be coded by "alphabet states" of the cellular space. These will simply be the integers $1, 2, \ldots, m$. On either side of a primary cell coding a tape symbol, there will be a secondary cell in an "intermediate state" of value $I = 2m$. Figure 1 shows how the left portion of a string is coded into cells and how the transition function operates on these cells. Our scheme follows the following lines:

(1) If a (primary) cell is in state $a$, $1 \leq a \leq m$, then the neighborhood total is $2I + a$, and $4m + 1 \leq 2I + a \leq 5m$. Whenever the total is in this range, the new state is obtained by subtracting $2I$ from the sum, leaving the cell in the same state $a$.

(2) A (secondary) cell in state $I$ will have states $a$, $b$, on either side, with $0 \leq a \leq m$ and $1 \leq b \leq m$ ($a = 0$ means that one of the cell's neighbors is in the quiescent state). The neighborhood total will be $a + I + b$ and $2m + 1 \leq a + I + b \leq 4m$. In this case, the new state is defined to be $I$, leaving the cell unchanged.

(3) A quiescent cell (state 0) bordering a cell in state $I$ will always have its other neighbor in the quiescent state, so its neighborhood total will be $I = 2m$. In this case, the transition function is defined as 0, leaving the cell in the quiescent state.

| Cell states | 0 | 0 | 0 | $I$ | $a$ | $I$ | $b$ |
|---|---|---|---|---|---|---|---|
| Neighborhood totals $T$ | | 0 | $I$ | $I + a$ | $2I + a$ | $a + I + b$ | |
| Range of values of $T$ | | $[0, 0]$ | $[2m, 2m]$ | $[2m + 1, 4m]$ | $[4m + 1, 5m]$ | $[2m + 1, 4m]$ | |
| Operation of transition function | | $T \to 0$ | $T \to 0$ | $T \to I$ | $T \to T - 2I$ | $T \to I$ | |
| New cell states | | 0 | 0 | $I$ | $a$ | $I$ | |

**Fig. 1.** Representation of a Turing machine's tape containing the symbols $ab\ldots$, showing operation of the totalistic automatas' transition function.

### *3.3. Representing the Read/Write Head*

As mentioned, the head and the scanned tape symbol are coded onto three adjacent cells in an asymmetrical manner. This is illustrated in Figure 2, where we see seven cells, with M's state $q$ and scanned symbol $s$ coded into the three center cells. The cells labeled 1, 3, and 5 are primary cells—the others are secondary—and the two extreme cells are in state 0 or $I$. Throughout, we shall use $(I)$ to indicate an integer which is either 0 or $I$. Cells 1 and 5 are respectively in states $a$ and $b$, where $0 \le a,\ b \le m$. If $a \ge 1$, it means that the cell codes the tape symbol $a$; similarly for $b$. The integer $s$ is the scanned tape symbol and the integer $q$ is the machine's internal state, so $1 \le s \le m$ and $1 \le q \le n$.

$A$, $B$, and $C$ are three integer constants, dependent on $m$ and $n$, whose values will be determined later. The left secondary cell (bordering the scanned cell) codes $s$ and $q$ as $sA+qB+C$; the right secondary cell codes $s$ and $q$ as $sA+qB+2C$; and the center cell is coded simply as 3C. The bottom part of Figure 2 shows the neighborhood totals for cells 1-5. Our purpose is to be able to extract from these totals all the relevant information for each of these cells. The coding is designed so that in each of the five totals containing multiples of $C$, the coefficient of $C$ is unique. By extracting that coefficient from the sum, the cell can determine its exact position relative to M's head.

If $C$ is made larger than the maximum possible value of the totals without the $C$ factors, then $C$'s coefficient would be the quotient after dividing (integer division) the total by $C$. So we choose

$$C = \max\{3m+mA+nB,\ m+mA+nB,\ 2mA+2nB\}+1 = 2mA+2nB+1.$$

Note that cells 1 and 5 and cells 2 and 4 have remainders of the same type, so we consider them jointly in the following discussion. The remainders after division by $C$ are: $(I)+a+sA+qB$, for cells 1 and 5; $a+sA+qB$, for cells 2 and 4; $sA+qB$, for cell 3 (after dividing the remainder by 2).

We continue to extract information from the remainders in a similar manner: by choosing $B = 3m+mA+1$, we can obtain M's state $q$ as the quotient after division by $B$. The remainders after divison by $B$ are: $(I)+a+sA$, for cells 1 and


| | | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|---|
| Cell states | $(I)$ | $0 \le a \le m$ | $sA+qB+C$ | $3C$ | $sA+qB+2C$ | $0 \le b \le m$ | $(I)$ |
| Totals | | $(I)+a+sA$ $+qB+C$ | $a+sA$ $+qB+4C$ | $2sA+2qB$ $+6C$ | $b+sA$ $+qB+5C$ | $(I)+b+sA$ $+qB+2C$ | |

**Fig. 2.** The asymmetric representation of a Turing machine in state $q$ scanning tape symbol $s$ (originally in cell 3).

5; $a+sA$, for cells 2 and 4; $sA$ for cell 3. To obtain $s$ (the scanned tape symbol), we choose $A=3m+1$.

Substituting back, we get $B=3m^2+4m+1$ and $C=6m^2(n+1)+8mn+2(m+n)+1$. Clearly, this value of $C$ is large enough to detect the presence of the read/write head in a cell's neighborhood.

We now divide the last remainders by $A$. The quotient is $s$ and the remainders are: $(I)+a$, for cells 1 and 5; $a$, for cell 3. For cells 1 and 5, we can distinguish between $I+a$ and $a$ because $0\leq a\leq m$; if the remainder is $I+a$, we obtain $a$ by subtracting $I$.

### 3.4. *The Intermediate Configuration*

Let us assume the configuration shown in Figure 2, and that M's transition function changes M's internal state from $q$ to $p$, the scanned symbol from $s$ to $t$, and that the head moves one square to the left. Figure 3 shows the first configuration in the top row (cells 1-5 are the same as in Figure 2) and the configuration representing M's next ID in the bottom row. The intermediate configuration is shown in the second row, with the neighborhood totals in the third row.

As shown before, each of the cells 1-5 has the following information: its position relative to the scanned tape symbol, the value of the scanned symbol ($s$), M's state $q$, and its own state, $a$ or $I$, as applicable. From this information and from M's transition function, each of these cells can determine its state in the next (intermediate) configuration. The intermediate configuration is designed to code the following information: value and position of the next scanned symbol

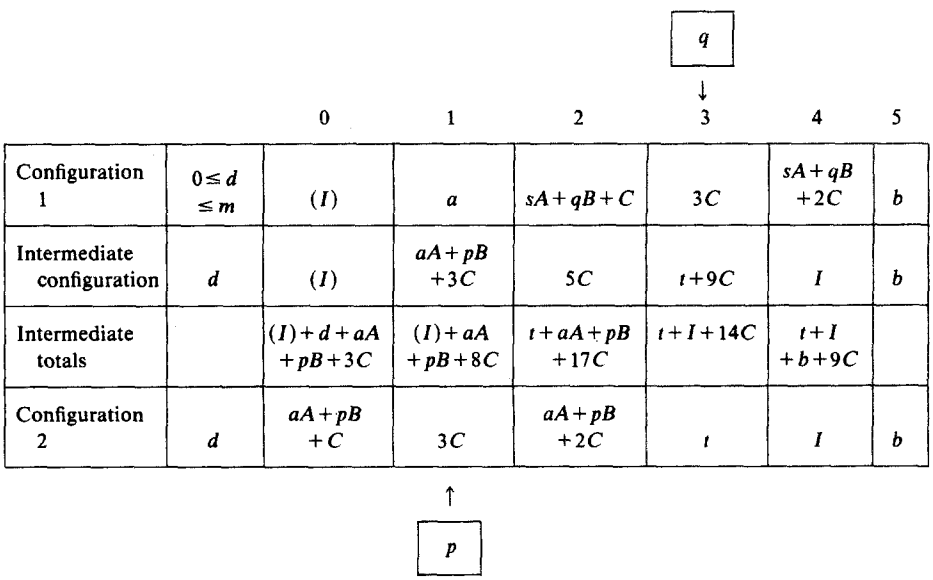


**Fig. 3.** The intermediate configuration representing a move of the Turing machine's head to the left.

($a$ in cell 1); the next state of M ($p$, coded together with $a$ in cell 1); new value and position of the previously scanned symbol ($t$ in cell 3).

The coding method for the intermediate configuration is similar to that of the "primary" configuration. Three multiples of $C$ are used as shown in Figure 3, chosen so that all the multiples of $C$ in the sums will be distinct and different from the previous ones. It is easily seen that all the relevant information can be extracted from the sums as before by dividing by $C$, then by $B$, and then by $A$. After decoding the information from the sums, each of the cells 0–5 enters its new state as shown in Figure 3.

There is one detail to note concerning cell 1: if $a=0$, it means that cell 1 was quiescent and that the head is moving over a tape square that was not scanned before and was not part of M's original input string. In a Turing machine, such tape squares are assumed to contain the blank symbol $\not b$, so in order for the simulation to be correct, cell 1 enters state $\not bA+pB+3C$ (for convenience, we assume that $\not b$ is the integer between 1 and $m$ corresponding to a blank).

If cells 1–5 have determined that M's head moves to the right, then the intermediate configuration will be as shown in Figure 4. We omit further details, but note only that the neighborhood total for cell 2 includes $9C$, which has already appeared before. This is not a problem, because in both cases the cell enters state $I$ in the next configuration.

### 3.5. *Main Results*

From the above simulation we get:

**Theorem 1.** *A Turing machine with m tape symbols and n states can be simulated in twice real time by a one-dimensional, three-neighbor totalistic cellular space with* $54m^2(n+1)+72mn+19m+18n+9$ *states.*

Intermediate head position

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Intermediate configuration | $a$ | $I$ | $t+9C$ | $3C$ | $bA+pB$ $+7C$ | $(I)$ | $0\le e$ $\le m$ |
| Intermediate totals | | $a+I$ $+t$ $+9C$ | $I+t+12C$ | $t+bA+pB$ $+19C$ | $(I)+bA$ $+pB+10C$ | $(I)+e+bA$ $+pB+7C$ | |
| Configuration 2 | $a$ | $I$ | $t$ | $bA+pB$ $+C$ | $3C$ | $bA+pb$ $+2C$ | $e$ |

↑

$p$

**Fig. 4.** The intermediate configuration representing a move of the Turing machine's head to the right.

*Proof.* In the simulation described above, every second configuration corresponds to an ID of the Turing machine, and this is twice real time by the definitions of [15]. It is easy to see that the maximal value of a state is $9C + m$, which yields the above expression. □

**Corollary 1.** *There exists a computation-universal one-dimensional, three-neighbor totalistic cellular space with* 9139 *states.*

*Proof.* Minsky [13] gives a universal Turing machine with four tape symbols and seven states. Substituting $m = 4$ and $n = 7$ into the expression in Theorem 1 yields 9139. □

## 4. Semitotalistic Cellular Automata

In the semitotalistic case, every cell can represent one tape square, but we still need an asymmetrical representation for the read/write head. This is done by having the cell to the *left* of the scanned cell also code the machine's state and the scanned symbol.

Let M be a Turing machine with $m$ tape symbols and $n$ states. Our coding technique is illustrated by an example in Figure 5, where we see 10 tape symbols with the string *abcdef* in squares 3–8, with blanks on either side and M in state $q$ scanning square 6. The assumed blanks on either side of the string are represented as before by the quiescent state (though a blank written by M will be represented by a positive integer). The symbols in squares 3, 4, 7, and 8 are represented simply by integers in the range $1, \ldots, m$. The cell to the left of the scanned cell codes its own symbol $c$, the scanned symbol $d$, and state $q$ as $cA + dB + qC$. The scanned cell codes symbol $d$ and state $q$ as $dB + qC$.

The neighborhood totals shown in the bottom row of Figure 5 do not include the cell's own state. The purpose of the coding is to enable the cells to extract all the relevant information from their own state and from the sum of their neighbors' states, and the technique is similar to the totalistic case. If the neighbor-

| | | | | | | $q$ ↓ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Tape symbols | $\not b$ | $\not b$ | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $\not b$ | $\not b$ |
| Cell states | 0 | 0 | $a$ | $b$ | $cA+dB+qC$ | $dB+qC$ | $e$ | $f$ | 0 | 0 |
| Neighborhood totals | 0 | $a$ | $b$ | $a+cA$ $+dB+qC$ | $b+dB+qC$ | $e+cA$ $+dB+qC$ | $f+dB+qC$ | $e$ | $f$ | 0 |

**Fig. 5.** Representation of a Turing machine's head position, state and tape by semitotalistic automata.

hood total is $\leq 2m$, then the cell remains in its present state in the next configuration. Otherwise the cell is in proximity to the read/write head (cells 4–7 in Figure 5) and it extracts its needed information by dividing the total by $C$, then the remainder by $B$, and then by $A$.

Note that the multiple of $A$ in the scanned cell is 0 and in its left neighbor it is $c$; since this fact is used to distinguish between the two cells, we must have $c \geq 1$. Therefore, even if the scanned symbol is the leftmost, the $\not b$ to its left is represented by the corresponding positive integer (and not by 0). It is straightforward to verify that every cell near the head can obtain its required information from its own state and neighborhood total. By arguments similar to the totalistic case, we get $A = m+1$, $B = (m+1)^2$, and $C = m^3+3m^2+2m+1$. Our simulation result can be stated as:

**Theorem 2.** *A Turing machine with m tape symbols and n states can be simulated in real time by a one-dimensional, three-neighbor semitotalistic cellular space with* $(n+1)(m^3+3m^2+2m+1)-1$ *states*

*Proof.* All that remains to do is to note that the maximal value of a state is $mA+mB+nC$, which yields the above expression. □

**Corollary 2.** *There exists a computation-universal one-dimensional, three-neighbor semitotalistic space with* 967 *states.*

*Proof.* Take Minsky's [13] universal Turing machine with $m=4$ tape symbols and $n=7$ states. □

## 5. Conclusions and Further Research Possibilities

We have seen how an asymmetrical representation of the read/write head enables the cells to distinguish between the left and right sides of the head. The number of automata states required by our simulations is a polynomial in the number of tape symbols ($m$) and number of states ($n$) of the simulated Turing machine. In this respect our results are weaker than those of Smith [15], where the number of states is linear in $m$ and $n$. However, our simulation times (twice real time and real time) are no worse than those of [15].

One question raised by these results is whether one can reduce the number of automata states—say even down to linear in $m$ and $n$—at the possible expense of an increase in the simulation time. If so, what are the precise tradeoffs?

*Lower Bounds on Number of States.* Consider the fact that cells adjacent to the read/write head need to extract the following information from their neighborhood totals: their own state, the scanned symbol, the machine state, and their position relative to the head. The number of possibilities is $\Theta(m^2n)$ and our simulation achieves that bound in the number of states. In the semitotalistic case, note that cell 4 (in Figure 5) needs to extract from the total the machine state $q$, the symbols of cells 5 and 6 as well as to distinguish them from the symbol of

cell 3. Altogether, there are three symbols and one machine state involved, yielding $m^3n$ different combinations, and our simulation does indeed use $\Theta(m^3n)$ states.

Based on these observations, we conjecture that $\Omega(m^2n)$ and $\Omega(m^3n)$ are lower bounds on the number of states required to simulate a Turing machine in the two cases, assuming twice real time and real time for, respectively, totalistic and semitotalistic automata.

*Computation-Universality.* If one is interested only in computation-universality and not in the simulation time, then we are left with the question of what is the minimal number of states required for computation-universality in totalistic and semitotalistic automata. Here, we could have another possible tradeoff relation: between the neighborhood size and the number of states. In the two-dimensional semitotalistic case, "Life" has the obviously minimal number of states (two), assuming that a cell's neighborhood includes all eight closest neighbors. One could also consider the case where a cell's neighborhood includes only the four cells sharing a common boundary, and the hexagonal case where a cell has six neighbors.

## References

[1] M. A. Arbib, *Theories of Abstract Automata*, Prentice-Hall, Englewood Cliffs, NJ, 1969.
[2] A. W. Burks, ed., *Essays on Cellular Automata*, University of Illinois Press, Urbana, II, 1970.
[3] E. F. Codd, *Cellular Automata*, Academic Press, New York, 1968.
[4] A. K. Dewdney, Computer recreations, *Sci. Amer.*, **252** (1985), 18-30.
[5] M. Gardner, Mathematical games, *Sci. Amer.*, **224** (1971) and 226 (1972).
[6] M. Gardner, *Wheels, Life and Other Mathematical Amusements*, Freeman, San Francisco, 1983.
[7] G. T. Herman and G. Rozenberg, *Developmental Systems and Languages*, North-Holland, Amsterdam, 1975.
[8] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, MA, 1979.
[9] O. H. Ibarra and S. M. Kim, Characterizations and computational complexity of systolic trellis automata, *Theoret. Comput. Sci.*, **29** (1984), 123-153.
[10] S. R. Kosaraju, On some open problems in the theory of cellular automata, *IEEE Trans. Comput.*, **33** (1974), 561-565.
[11] H. T. Kung, Let's design algorithms for VLSI systems, *Proceedings of the Caltech Conference on VLSI*, Pasadena, CA, 1979, pp. 65-69.
[12] C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, MA, 1980.
[13] M. Minsky, *Computation: Finite and Infinite Machines*, Prentice-Hall, Englewood Cliffs, NJ, 1967.
[14] G. Rozenberg and A. Salomaa, *The Mathematical Theory of L Systems*, Academic Press, New York, 1980.
[15] A. R. Smith, III, Simple computation-universal cellular spaces, *J. Assoc. Comput. Mach.*, **18** (1971), 339-353.
[16] J. von Neumann, *The Theory of Self-Reproducing Automata* (A. W. Burks, ed.), University of Illinois Press, Urbana, IL, 1966.
[17] S. Wolfram, Statistical mechanics of cellular automata, *Rev. Modern Phys.*, **55** (1983), 601-644.
[18] S. Wolfram, Universality and complexity in cellular automata, in *Cellular Automata* (Proc. Interdisciplinary Workshop, Los Alamos, 1983) (D. Farmer, T. Toffoli, and S. Wolfram, eds.), North-Holland, Amsterdam, 1984.

*Received August 16, 1985, and in revised form February 26, 1987, and in final form April 20, 1987.*