JOURNAL OF COMPUTER AND SYSTEM SCIENCES 50, 87–97 (1995)

On the Computational Complexity of Finite Cellular Automata

KLAUS SUTNER

Stevens Institute of Technology, Hoboken, New Jersey 07030

Received September 28, 1988; revised April 18, 1994

We study the computational complexity of several problems with the evolution of configurations on finite cellular automata. In many cases, the problems turn out to be complete in their respective classes. For example, the problem of deciding whether a configuration has a predecessor is shown to be NLOG-complete for one-dimensional cellular automata. The problem is NP-complete for all dimensions higher than one. Similarly, the question whether a target configuration occurs in the orbit of a source configuration may be P-complete, NP-complete or PSPACE-complete, depending on the type of cellular automaton.
© 1995 Academic Press, Inc.

1. INTRODUCTION

The evolution of configurations on classical infinite, as well as finite, cellular automata has been studied extensively; see, for example, [24, 14, 7, 25]. In this paper we will focus on variations of the following two problems:

Configuration reachability problem. Given a cellular automaton, a source configuration X and a target configuration Y, does Y occur in the orbit of X?

Predecessor existence problem. Given a cellular automaton and a target configuration Y, does Y have a predecessor?

The first problem is well known to be undecidable for infinite one-dimensional cellular automata. The proof hinges on the fact that one-dimensional cellular automata are capable of simulating Turing machines: instantaneous descriptions of a Turing machine are readily expressed as configurations of a one-dimensional cellular automaton and the transition rule of the cellular automaton can simulate the transition function of the Turing machine. Since the halting problem for Turing machines is undecidable, it is also undecidable whether a certain target configuration occurs in the orbit of a source configuration. More precisely, this problem is Σ_1^0-complete if we require both configurations to be 0-finite, i.e., to have finite support. Indeed, one can construct a totalistic, one-way, computationally universal automaton with an alphabet of size 14 (von Neumann's original two-dimensional cellular automaton had 29 states; see [1, 3]). A notorious example for a two-dimensional cellular automaton, using a two-state alphabet and a deceptively simple transition rule, that nonetheless allows one to emulate Turing machines, is Conway's game of Life.

With regard to the existence of predecessor configurations, a remarkable distinction between one- and higher-dimensional cellular automata occurs. In the one-dimensional case, it is decidable whether a given 0-finite configuration has a predecessor. This follows from the fact that the class of all configurations that have a predecessor on a one-dimensional cellular automaton can be constructed as a regular language of biinfinite words; see [15, 22, 20]. On the other hand, Yaku shows in [25] that the problem is in general undecidable for two-dimensional cellular automata. In fact, Yaku establishes the following two results:

THEOREM A. *For any Turing machine M there is a two-dimensional rule ρ_M and a finite 0-finite configuration Y such that Y has a predecessor under ρ_M iff Turing machine M does not halt on the empty tape.*

THEOREM B. *For any Turing machine M there is a two-dimensional rule ρ_M and a 0-finite configuration Y such that Y has a 0-finite predecessor under ρ_M iff Turing machine M halts on the empty tape.*

The rule ρ_M as well as the target configuration Y can be constructed effectively from M. Predecessor existence restricted to 0-finite configurations is clearly Σ_1^0 and thus, by theorem B, Σ_1^0-complete. Since the space of all configurations with the usual product topology is compact, the general problem where the predecessor configuration is allowed to have infinite support is Π_1^0 and, by Theorem A, Π_1^0-complete for infinite two-dimensional cellular automata.

The orbits of configurations have also been used to classify cellular automata. In a recent paper by Culik and Yu (see [11]) the authors propose a formalization of Wolfram's heuristic classification of cellular automata given in [24]. The classification uses four types based on the kind of orbits that occur in the cellular automaton. Cullik and Yu

0022-0000/95 $6.00
Copyright © 1995 by Academic Press, Inc.
All rights of reproduction in any form reserved.

prove that it is undecidable to which class a given cellular automaton belongs. For example, they show that one cannot algorithmically determine whether all configurations on a one-dimensional cellular automaton ultimately evolve to a fixed point. Their results are extended in [18] to show that the Culik–Yu classes are Π_2^0-complete (Classes One and Two) and Σ_3^0-complete (Class Three).

In this paper we will mostly be concerned with finite cellular automata. Needless to say, any question about the evolution of configurations trivially becomes decidable for any finite cellular automaton of fixed size. However, we are interested in the family of automata obtained by using a fixed local map and finite grids of cells of varying size. For finite cellular automata in this sense, the classification leads essentially to the same computational problems as for the infinite machines, see [19]. In particular, it is undecidable if all configurations on a finite cellular automaton evolve to a fixed point. See also [4] for a similar undecidability phenomenon in a slightly different setting.

We will show that the complexity of the predecessor problem depends on the dimension of the cullular automaton. The problem lends itself naturally to a nondeterministic solution: guess the predecessor configuration and verify that it indeed leads to the desired target configuration in one application of the global map. We will show that the problem is complete in nondeterministic logarithmic space for finite one-dimensional cellular automata and NP-complete for all dimensions higher than one. Note that the related problem of testing whether the global map of a cellular automaton is surjective is decidable for infinite one-dimensional automata but undecidable for all higher dimensions; see [2, 20, 13].

The reachability problem as stated above is easily to seen to be PSPACE-complete even in dimension one. It is considerably more difficult to describe the complexity of certain variations of reachability problem. For example, one may require the rule to be predictable in the sense that one can compute $\rho^t(X)$ in polynomial time. Since the time t contributes only $\log t$ to the size of the instance, this rules out any brute force enumeration approach. As we will see, the existence of a cellular automaton that fails to be predictable is equivalent to P $\neq$ PSPACE. Predictable cellular automata lead to reachability problems in NP and we will show that for one-dimensional predictable rules reachability of partially specified configurations is already NP-complete. If we consider only a part of the orbit of polynomial length the problem turns out to be P-complete.

Background material on complexity issues can be found in [5, 16]. The necessary definitions for cellular automata are presented briefly in the next section. Finite cellular automata are also discussed in [14, 17]. Section 3 is devoted to the study of one-dimensional cellular automata. In Section 4 we briefly comment on higher-dimensional automata.

2. DEFINITIONS

For our purposes, a *cellular automaton*, or CA for short, can be represented as a quintuple $\mathbf{A} = \langle d, k, w, \rho \rangle$. Here d denotes the dimension of the automaton, i.e., the dimension of the underlying grid of cells. Every cell can assume a finite number of states; the collection Σ of possible states is called the *alphabet* of the automaton. As is customary, we use alphabets of the form $\Sigma = \Sigma_k = \{0, 1, \ldots, k-1\}$, which accounts for the second component of a cellular automaton. In order to exclude degenerate cases, we will always assume that $k \geq 2$. The positive integer w is the *width* of $\mathbf{A}$ and $\rho: \Sigma^N \to \Sigma$ is the *local map* or *local rule* of the automaton. For the sake of simplicity we assume that N, the *basic neighborhood* of the rule, is of the form $N := [-r, r]^d \subseteq \mathbf{Z}^d$ and $w = 2r + 1$, where $r \geq 1$ is the *radius* of the rule.

A map $X: C \to \Sigma$ from the set C of all cells to the alphabet is a *configuration* of the cellular automaton and $\mathscr{C}$ denotes the space of all configurations. We will distinguish between two types of cellular automata:

- infinite CAs, where $C = \mathbf{Z}^d$
- and infinite CAs, where the grid of cells is of the form $C = [n_1] \times [n_2] \times \cdots \times [n_d]$.

Here $[m]$ denotes the set $\{1, 2, \ldots, m\}$. In the finite case, we will refer to $n := n_1 \cdot \cdots \cdot n_d$ as the *size* of the CA. Similarly, we will refer to configurations as being finite or infinite. Note that in the literature infinite configurations with finite support are occasionally referred to as "finite." To avoid any possible confusion, we refer to these configurations as 0-finite: there are only finitely many cells c such that $X(c) \neq 0$.

We need to extend the local map ρ to a *global map* (also denoted by ρ) $\rho: \mathscr{C} \to \mathscr{C}$. To this end, given a configuration X and a cell $c \in C$, define the *local configuration* at c, $X_c: N \to \Sigma$, by $X_c(z) := X(c + z)$. We can then define the global map by $\rho(X)(c) := \rho(X_c)$.

Note that there is a minor technical difficulty with finite cellular automata: the local rule cannot be applied to cells close to the boundary of the automaton. One possible solution to this problem is to use *cyclic boundary conditions* and glue together opposite in the grid. A two-dimensional automaton is thus defined on a torus rather than a plain grid. Another standard solution to this problem is to adopt *fixed boundary conditions* and assume that the missing cells are in a special, fixed state. Unless noted otherwise, this state will always be $0 \in \Sigma_k$. Less informally, modify the definition of a local configuration at cell c as follows:

$$X_c(z) := \begin{cases} X(c+z), & \text{if } c + z \in C \\ 0, & \text{otherwise.} \end{cases}$$

In this paper, we have chosen to use fixed boundary condi-

tions throughout. It is quite straightforward, albeit tedious, to modify the arguments given here so that they apply to the cyclic boundary situation. Finite cellular automata with cyclic boundary conditions are closely related to infinite automata restricted to periodic configurations. On the other hand, fixed boundary conditions do not simply correspond to 0-finite configurations on infinite machines since there the size of the configuration can grow indefinitely.

A *pattern* is a partial configuration, i.e., a map $X_0: C_0 \to \Sigma$, where $C_0 \subseteq C$. We say that a configuration X *matches* pattern X_0 if $X(c) = X_0(c)$ for all $c \in C_0$, in symbols $X_0 \sqsubseteq X$. The *orbit* of a configuration X is the set $\{\rho^t(X) \mid t \geq 0\}$. We will study the following decision problems:

Problem. Predecessor existence problem (PEP).
Instance: A cellular automaton **A** and a target configuration Y.
Question: Is there a predecessor configuration X such that $\rho(X) = Y$?

Problem. Pattern reachability problem (PREP).
Instance: A cellular automaton **A**, a source configuration X, and a target pattern Y.
Question: Is there some configuration in the orbit of X that matches Y?

Since we are dealing with low complexity classes like NLOG and P, let us briefly comment on issues concerning the coding of instances. A cellular automaton $\mathbf{A} = \langle d, k, w, \rho \rangle$ can be coded as follows. The integers d, k, and w are given in binary. The local map is represented by a word $r_1 r_2 \cdots r_p$ over Σ_k of length $p = k^{w^d}$ so that $\rho(X_i) = r_i$. Here $(X_i)_i$ is the canonical enumeration of all local configurations $[-r, r]^d \to \Sigma_k$. Since the symbols of Σ_k are coded as binary words of length $\lceil \log k \rceil$, we can code the whole automaton in $\Theta(k^{w^d} \log k)$ bits. Similarly, a finite configuration X can be expressed as a word of length $n = n_1 \cdot \cdots \cdot n_d$, together with a list of the grid sizes $n_1, \ldots, n_d$. Thus, X can be coded in $\Theta(n \log k + \log n)$ bits.

Both problems are stated in their uniform version, where both the cellular automaton and the configurations are part of the input. We can also consider a parametrized version where the cellular automaton is fixed and only the configurations vary. In this case, the size of the input is $\Theta(n)$, where n is the number of cells in the automaton. All the upper bounds that we are going to establish hold for the uniform version, the lower bounds—in the form of hardness results—are usually for the parametrized version.

The reachability problem admits a number of modifications. First, one can restrict the target patterns to be configurations, i.e., total maps from C to Σ. In this case we have to determine whether a certain target configuration occurs in the orbit of a source configuration.

Second, we can impose a bound on the time t, say, some polynomial function of the number of cells of the automaton; see Section 3 below. Variations of these problems were also considered in [7, 8, 25].

Cellular automata are a model of parallel computation; in order to generate the next configuration, the local rule is applied to all the cells simultaneously. However, the computational cost of simulating a finite cellular automaton on a Turing machine is small; one has to memorize one local configuration and a position in the grid C. This requires only a logarithmic amount of space, as we note in the next proposition.

PROPOSITION 2.1. *Given a finite cellular automaton* **A** *and a configuration X one can compute the successor configuration $\rho(X)$ in deterministic logarithmic space.*

Proof. Suppose that the input is given as d, r, k, ρ, X on the input tape of a Turing machine as described above. To compute $\rho(X)$, one has to successively determine the local configurations X_c, c in $C = [n_1] \times [n_2] \times \cdots \times [n_d]$, and then find $\rho(X_c)$ by a table look-up. A local configuration has size $O(w^d \log k)$ and the arithmetic needed to keep track of positions in C, $r_1, \ldots, r_p$ and X can be handled with $O(\log n)$, $O(r^d \log k)$, and $O(d \log n + \log k)$ bits, respectively. Hence the natural algorithm to compute $\rho(X)$ is in deterministic logarithmic space. Note that this is the analysis for the uniform problem. If ρ is fixed, the algorithm is in deterministic space $O(\log n)$. ∎

The trick employed in [12] to demonstrate that log-space reductions are transitive can also be used to show that $\rho^t(X)$ can be computed in deterministic logarithmic space for any *fixed* t.

PROPOSITION 2.2. *The predecessor existence problem is in nondeterministic polynomial time for all finite cellular automata.*

Proof. To see this, note that one can nondeterministically guess the predecessor configuration X of Y in $O(n \log k)$ steps, where n is the size of the automaton and k's the size of the alphabet, and then verify that $\rho(X)$ indeed equals the target configuration X. The verification procedure is in deterministic logarithmic space as we have just seen; hence the whole procedure is in nondeterministic polynomial time. ∎

PROPOSITION 2.3. *The pattern reachability problem in in deterministic linear space for all finite cellular automata.*

Proof. The obvious algorithm to test whether a configuration Y occurs in the orbit of X is to successively generate all configurations $\rho^0(X)$, $\rho^1(X)$, ... Let n be the number of cells in the cellular automaton. If the target

configuration occurs during the first k^n steps we return "yes," and "no" otherwise. Clearly, this can be done in deterministic linear space. ∎

Local rules for which configurations in the orbit of a given source configuration can be computed in polynomial time will be called predictable. More precisely, a local map ρ is *predictable* iff $\rho^t(X)$ can be computed in time polynomial in the size of X and t. Examples for predictable rules are additive rules. Suppose $\langle \Sigma, \oplus, \otimes, 0, 1 \rangle$ is a semiring and the local map ρ has the property that $\rho(X \oplus Y) = \rho(X) \oplus \rho(Y)$, where the operation on configurations is defined pointwise. Then $\rho^t(X)$ can be computed essentially by generating the tth power of an n by n matrix over Σ, where n is the size of X. The latter problem can trivially be solved in $O(n^3 \log t)$ steps. Predictable rules formalize the notion of computationally reducible rules in [24, 23].

PROPOSITION 2.4. *The pattern reachability problem is in nondeterministic polynomial time all-predictable finite cellular automata.*

Proof. Suppose there is a time t such that $Y \sqsubseteq \rho^t(X)$. Then the least such t is less than k^n, where k is the number of symbols in the alphabet and n is the number of cells. Hence one can guess t and verify that the guess was correct in polynomial time. ∎

Thus, the complexity of PREP for predictable rules is a priori lower than for arbitrary rules. Of course, it might be the case that indeed NP = PSPACE and that the distinction becomes immaterial. Likewise, it is possible that P = PSPACE and that PREP is indeed solvable in polynomial time for each finite cellular automaton. We will show in Section 4 that for some predictable two-dimensional rule, the parametrized version of PREP is already NP-complete.

3. ONE-DIMENSIONAL CELLULAR AUTOMATA

Configuration of a one-dimensional cellular automaton (finite or infinite) can be construed as a word over the alphabet Σ. Therefore, one may use language-theoretic methods to answer questions about the evolution of configurations. Concepts from automata theory are also useful to show decidability of the reversibility problem and the surjectivity problem; see [10, 2]. Reference [9] contains an analysis of the languages of biinfinite words obtained from infinite one-dimensional cellular automata. There is no obvious way to generalize the language-theoretic approach to dimensions higher than one. Indeed, as we will see, finite one-dimension cellular automata significantly from all higher dimensions with respect to PEP.

So assume ρ is a one-dimensional local map over some alphabet Σ. Let $\mathscr{L}_\rho \subseteq \Sigma^*$ be the collection or all finite words that occur as subwords of any configuration $\rho(X)$, constructed as a word. It is not hard to see that $\mathscr{L}_\rho$ is a regular language. As was pointed out in [15, 22], a semiautomaton that accepts $\mathscr{L}_\rho$ can be described in terms of directed, Σ-labeled de Bruijn graphs. To be more explicit, define the de Bruin graph $B(s, \Sigma)$ as follows: $B(s, \Sigma)$ has vertex set Σ^s and edges (ax, xb) for all $a, b \in \Sigma$, $x \in \Sigma^{s-1}$. Now suppose that ρ is a rule of radius $r \geqslant 1$. Set $s := 2r$ and label edge (ax, xb) by $\rho(axb) \in \Sigma$. It is easy to see that the resulting semiautomaton B_ρ accepts $\mathscr{L}(\rho)$ if we think of ρ either as an infinite cellular automaton or as a finite cellular automaton with cyclic boundary conditions. In [20] these de Bruijn automata are used to construct quadratic time algorithms that test whether an infinite one-dimensional cellular automaton is injective, open, surjective, or neither. For a study of the size of the corresponding minimal automata see [21].

To obtain the language associated with periodic boundary conditions we have to declare all vertices of the form $0^r x$ to be initial and all vertices of the form $x0^r$ to be final, $x \in \Sigma^r$ (recall our convention that state 0 is the default value for "missing" cells). In either case, the resulting finite state machine B_ρ accepts $Y \in \Sigma^*$ iff $Y \in \mathscr{L}_\rho$ iff Y as a configuration has a predecessor under rule ρ.

We note in passing that the de Bruijn graph can also be used to determine whether a given 0-finite configuration ${}^\omega 0 Y 0^\omega \in \mathscr{C}$ has a 0-finite predecessor ${}^\omega 0 X 0^\omega \in \mathscr{C}$. In fact, one only has to change the initial and final states in B_ρ as follows: x is initial iff there is an coinfinite path ending at x labeled only by 0. Similarly x is final iff there is an infinite path starting at x labeled only by 0. These points are readily identified by computing the strongly connected components of the subgraph of $B(s, \Sigma)$ whose edges are labeled 0. Hence, given ρ and Y, one can test in polynomial time whether ${}^\omega 0 Y 0^\omega$ has a 0-finite predecessor.

Returning to finite cellular automata, we have the following proposition.

PROPOSITION 3.1. *For any fixed rule ρ in dimension one, the existence of a prececessor configuration can be determined in deterministic linear time and constant space.*

As we will see shortly, the uniform version of PEP is much harder. Note that the size of B_ρ is exponential in k and w. However, one single state in B_ρ has size only $Q(w \log k)$. This is used in the following lemma.

LEMMA 3.1. *The predecessor existence problem for finite one-dimensional cellular automata is in nondeterministic logarithmic space.*

Proof. First note that the size of an instance $\mathbf{A} = \langle d, k, w, \rho \rangle$ and Y is $\Theta((k^w + n) \log k)$, where n is the length of Y, i.e., the number of cells of the automaton.

To test whether the de Bruijn automaton B_ρ from above accepts X, one nondeterministically guesses a path in B_ρ from an initial to a final state and verifies that the edge lables correspond to X. This can be done by keeping pointers to positions on the input tape and two registers that contain the current and the next node in B_ρ. The space requirement for all these objects together is $O(w \log k + \log n)$ and thus logarithmic in the size of the input. ▮

For our hardness results below we will have construct certain local maps. For the sake of legibility, we will use suitable alphabets rather than Σ_k. Also, it is usually inconvenient to specify the local rule as the explicit list described in Section 2. Suppose that we have fixed an alphabet Σ and wish to specify a d-dimensional rule of radius r. Rather than listing the values of ρ on all local configurations $W \in \Sigma^{[-r,r]^d}$, we will usually describe a subcollection of local configurations, called *admissible* local configurations. For all admissible configurations, ρ is defined explicitly. On inadmissible configurations, ρ has some fixed default value. Admissible local configurations can frequently be described as the set of all subconfigurations of a given configuration.

THEOREM 3.1. *The uniform predecessor existence problem for finite one-dimensional cellular automata is* NLOG*-complete with respect to log-space reductions.*

Proof. Membership in NLOG was shown in the last lemma. To see hardness, we will embed the graph accessibility problem (GAP). An instance of GAP is a directed finite graph $G = \langle V, E \rangle$ and two vertices a (the source) and b (the target). The problem is to determine whether there exists a path in G from a to b. We may safely assume that $V = [m]$, $a = 1$, and $b = m$. We will define a rule ρ over the alphabet $\Sigma = [m] \cup \{0, \#\}$ that allows one to translate GAP into a predecessor existence problem. The width of the rule is 3. Define the target configuration X by

$$\#0\#0\# \cdots \#0\# \in \sum^{2m+1}.$$

The admissible local configurations for rule ρ are

$$\#u\#, \quad u\#v, \quad 0\#1, \quad m\#0,$$

where $1 \leqslant u, v \leqslant m$. Thus an admissible local configuration contains symbols in $[m]$ alternating with #'s. All inadmissible local configurations are mapped to 0. On admissible local configurations ρ is defined by

$$\rho(u\#v) := \begin{cases} \#, & \text{if } u = v \text{ or } (u, v) \text{ is an edge in } G, \\ 0, & \text{otherwise,} \end{cases}$$

and

$$\rho(\#u\#) := u.$$

For the remaining inadmissible local configurations W let $\rho(W) := 0$. It is not hard to see that any predecessor Y of X must be of the form

$$\#1\#v_2\# \cdots \#v_{m-1}\#m\#,$$

where $1 \leqslant v_i \leqslant m$. Also, for $i = 1, \ldots, m-1$, we must have $v_i = v_{i+1}$ or (v_i, v_{i+1}) is an edge in G. Hence Y codes a path from 1 to m in G.

The symbols in Σ can be specified by $\lceil \log(m+1) \rceil$ bits. Hence, rule ρ, as well as target pattern X, can be constructed from G in logarithmic space. This shows that GAP $\leqslant_{\log}$ PEP; hence PEP is NLOG-complete with respect to log-space reductions. ▮

One should note that the last construction can easily be modified to produce a one-way rule ρ. Thus PEP is no easier for one-way rules on finite one-dimensional cellular automata than for ordinary rules. By coding vertices in binary, the construction can also be modified to yield rules over the fixed alphabet $\{0, 1, \#\}$ but with radius $\lceil \log m \rceil$. These results should be compared to Section 4, where it will be shown that the parametrized version of PEP is NP-complete for a fixed two-dimensional rule.

We now turn to problems concerning the orbit of a given source configuration. As we have seen in Section 2, testing for the occurrence of a certain configuration in the orbit can be done in linear space. On the other hand, one-dimensional cellular automata can simulate Turing machines. In particular, a linear bounded automaton—i.e., Turing machine whose worktape has the same size as the input—can be simulated by a finite one-dimensional cellular automaton using a rule of width 3. The acceptance problem for arbitrary linear bounded automata (LBA) is PSPACE-complete; moreover, the problem remains PSPACE-complete for a certain fixed linear bounded automaton. Hence we have the following proposition.

PROPOSITION 3.2. *There is a one-dimensional rule ρ such that the pattern reachability problem is* PSPACE*-complete for finite cellular automata using ρ, even when the target pattern is required to be total.*

It is quite straightforward to code the computation of an LBA by a cellular automaton in such a way that acceptance translates into the occurrence of a fixed point configuration $\top^n$. Then the LBA accepts some input x iff $\tau^{k^n}(X) = \top^n$, where X is the the configuration coding x, k is the size of the alphabet, and n is the size of X. Thus, if P $\neq$ PSPACE, the

rule τ will not be predictable. Conversely, since $\rho^t(X)$ is always computable in linear space, the existence of a non-predictable rule would imply that P = PSPACE. Thus, all rules are predictable iff P = PSPACE.

As we have seen in Proposition 2.4, PREP drops down to NP for predictable rules. In any finite cellular automaton the orbit of a configuration must necessarily consist of a transient part followed by a periodic part:

$$\forall 0 \leqslant i < j < t + p(\rho^i(X) \neq \rho^j(X)),$$
$$\rho^{t_0+p}(X) = \rho^{t_0}(X).$$

The proof of the next theorem hinges on the fact that for the local map constructed there, the transient part of the orbit of any configuration has length $t_0 = O(n)$ and the configurations in the periodic part can be easily determined, essentially by modular arithmetic. As a consequence, the rule is predictable.

THEOREM 3.2. *There is a predictable one-dimensional rule ρ such that the pattern reachability problem is* NP*-complete for finite cellular automata using ρ. Moreover, the reachability problem for ρ is solvable in polynomial time if the target pattern is total.*

Proof. We present the construction of the cellular automaton in two stages. In the first stage, we build a one-dimensional rule that nearly has the required properties: the rule is predictable and it is NP-hard to determine whether one of a simple collection of target patterns is matched by a configuration in the orbit of the source configuration. In stage two we then show how to modify the construction to obtain a single target pattern.

Stage One. We begin with the construction of a local map ρ that allows one to define configurations that act as a collection of mod-counters, i.e., counters that are incremented by 1 at each step and are reset to 0 whenever a threshold value is reached. Rule ρ uses alphabet $\Sigma = \{\#, 0, L, R\}$ and has width 5. On configurations of the form $\#0^iL0^j\#$, the rule causes cyclic behavior with period $2(i+j+1)$: L behaves like a particle that moves to the left, rebounds at the first $\#$, and changes into a new particle R.

TABLE I

Z	$\rho(Z)$
$ab\#cd$	$\#$
$0R0ab$	R
$ab0L0$	L
$00R\#a$	L
$a\#L00$	R

TABLE II

t	$\rho^t(X)$
0	$\#R0000\#0R00\#$
1	$\#0R000\#00R0\#$
2	$\#00R00\#000R\#$
3	$\#000R0\#000L\#$
4	$\#0000R\#00L0\#$
5	$\#0000L\#0L00\#$
6	$\#000L0\#L000\#$
7	$\#00L00\#R000\#$
8	$\#0L000\#0R00\#$
9	$\#L0000\#00R0\#$
10	$\#R0000\#000R\#$

R then moves to the right, rebounds at the other $\#$, turns back into an L, and so forth. A local configuration $Z \in \Sigma^5$ is admissible for this construction iff Z contains at most one separator symbol $\#$, and, on either side of the $\#$, at most one L and R. ρ maps all inadmissible configurations to $\#$. So suppose that Z is admissible. Then $\rho(Z)$ is defined by Table I, where $a, b, c, d \in \Sigma$. The configuration $X = \#R0^4\#R0^3\#$ simulates a pair of counters, the first conting modulo 10, initialized to 0, and the second counting modulo 8 and initialized 1. The first 10 steps in the evolution of X are shown in Table II. The full orbit of X has length $\operatorname{lcm}(10, 8) = 40$.

Let $W(i, j, 0) := \#0^iR0^j$ and $W(i, j, 1) := \#0^iL0^j$. Then the configuration

$$W(i_1, j_1, d_1)\, W(i_2, j_2, d_2) \cdots W(i_m, j_m, d_m)\, \# \qquad (*)$$

consists entirely of admissible configurations and has period $2 \cdot \operatorname{lcm}(i_v + j_v + 1 \mid v \in [m])$.

The NP-hardness of PREP can now be established by embedding the combinatorial problem of *simultaneous incongruences*, see [5]. An instance of simultaneous incongruences is a list of natural numbers $a_1, b_1, \ldots, a_m, b_m$, where $0 \leqslant a_v < b_v$ and one has to determine whether there exists a natural number $x \geqslant 0$ such that $x \not\equiv a_v \bmod b_v$ for all $v = 1, \ldots, m$. Equivalently, we may consider a variation of the problem where one has to determine the existence of a number x such that $x + a_v \not\equiv \bmod b_v$ for i.e., it remains NP-complete even if all numbers are specified in unary. Furthermore, it is safe to assume that all the b_i are even and that $b_i \geqslant 8$.

To represent an instance of simultaneous incongruences, we use a source configuration X as defined in $(*)$, where $i_v + j_v + 1 = b_v/2$ and $i_v = a_v$, $d_v = 0$, in case $0 \leqslant a_v < b_v/2$, and $i_v = b_v - a_v - 1$, $d_v = 1$, in case $b_v/2 \leqslant a_v < b_v$. Now consdider the set of cells immediately following a separator symbol $\#$: $C_0 := \{2 + \sum_{\mu<v} (b_\mu/2 + 1) \mid v = 1, \ldots, m\}$. Define a family of target patterns Y by $Y(c) = 0$ or

$Y(c) = L$, for all c in C_0. It is easy to see that configuration $\rho'(X)$ matches one of these patterns Y iff $t \not\equiv a_v \bmod b_v$ for all $v = 1, \ldots, m$.

To see that rule ρ is indeed predictable, first note that any cell in state # will remain in this state forever. Thus it suffices to consider patterns of the form $\#Z\#$, where $Z \in \{0, L, R\}^*$. First assume that Z has length at least 4. If Z contains neither L nor R then $\#Z\#$ is a fixed point. If Z contains either exactly one L or one R then $\rho'(\#Z\#)$ can easily be computed in polynomial time. If, on the other hand, Z has length less than 4 or contains several L's and/or R's, then after $O(|Z|)$ steps at least one other cell in Z permanently enters state #. The pattern $\#Z\#$ is thus partitioned into at least two blocks, $\rho'(\#Z\#) = \#Z_1\#Z_2\#$, and the same argument can be used again. It follows that after $O(n)$ steps any configuration X has either evolved to a fixed point under ρ or has the form indicated in (*), possibly interspersed with blocks of the form $\#00\cdots0\#$. From then on, the evolution of Z will be periodic and it is easy to compute $\rho'(Z)$ for all t sufficiently large.

Lastly, suppose we have a total target pattern, i.e., a configuration Y. We can test explicitly whether Y occurs during the first $O(n)$ steps in the evolution of X. After that, the orbit is cyclic and testing whether it contains Y comes down to an application of the Chinese remainder theorem. This concludes the argument.

Stage Two. To reduce the collection of target patterns to a single pattern we augment the local map ρ in the following way. Each cell is divided into two parts so that a configuration of length n can be thought of as a 2 by n matrix. Each row in this matrix will be referred to as a track. The top track uses the same alphabet Σ as in Stage One and the evolution of the top track is the same as in the last construction. However, instead of moving all the particles in parallel at each step of the cellular automaton, a particle now only moves when a special "signal" in the bottom track passes by. The bottom track uses alphabet $\Sigma' := \{\#, 0, R, R', R'', L, L'\}$, where # is a separator, 0 stands for a blank, and the other symbols are signals that move along the bottom track. The full alphabet is $\Sigma \times \Sigma'$. We will call the new local map τ.

The new source configuration X' is obtained from the initial configuration X from above as follows. Retain X as the top track of X' and place $\#R0^{n-3}\#$ into the bottom track. Thus, signal R is originally postioned in cell number 2. Then R moves to the right, and every time it reaches a particle in the top track, this particle will move ahead by one step, as described above. When R reaches the right #, it turns into a signal L and returns back to cell 2. The top track remains unchanged during this second sweep. In the third sweep, a test signal R' is sent down the track. If it encounters only patterns $\#0$ or $\#L$ in the top track, it arrives at the right boundary marker #, where it is reflected, turning into L'. L' then travels to the left and when it reaches the left boundary marker, it turns into signal R and the cycle continues. If the test signal R' encounters a pattern $\#R$ on the top track, it turns into signal R'' and continuous to move to the right. When it reaches the right boundary, it behaves exactly as signal R' would. Hence, one step with respect to rule ρ now is simulated by $4n$ steps of the new rule τ.

Now define C_0 to be cell number $n-1$, i.e., the cell before the right boundary marker. Consider the pattern $Y(n-1) = R'$. Y matches a configuration in the σ orbit of X if and only if at some point one of the test signals R' reached the right boundary. It is easy to check that this can only happen if X codes a yes-instance of simulations incongruences. Furthermore, rule τ is still predictable if we adopt certain conventions about how the signals travel in the bottom track. For example, the markers # will again be immovable. Colliding signals will annihilate each other and turn into #'s. We omit the details. ∎

Note that it is also possible to modify the rule ρ from the last proof to provide a hardness result for a single target pattern (represent a particle L that approaches a # by modifying the cell containing # and leave the next cell in state 0). However, the slightly more complicated construction with rule τ shows that PREP is NP-hard even if the target pattern is defined only on one cell.

Another way to reduce the complexity of PREP is to introduce a time bound, in particular a polynomial time bound. I.e., given ρ, X, Y, one has to determine whether $\exists t \leqslant p(n)(Y \sqsubseteq \rho'(X))$, where n is the number of cells in the automaton and $p(n)$ is some polynomial. Clearly, this polynomially bounded version of PREP is solvable in polynomial time by brute force. We will show in the next theorem that it is in fact P-complete with respect to log-space reductions. The proof hinges on the fact that boolean circuits can be represented by configurations on a one-dimensional cellular automaton. If, in particular, the circuits are in some normal form, then the transformation from circuit to configuration can be accomplished in logarithmic space. A similar approach will be used again in Theorem 4.1.

THEOREM 3.3. *There is a one-dimensional rule ρ such that the polynomially bounded pattern reachability problem is* P*-complete (with respect to log-space reductions) for finite cellular automata using ρ.*

Proof. We will express the evaluation of a boolean circuit as a configuration reachability problem. More precisely, we use boolean circuits consisting only of AND and OR gates and without feedback. Given such a circuit and values for the input variables the problem to determine whether a certain output variable assumes the value true is referred to as the circuit value problem (CVP). CVP is well known to be P-complete; see [6, 16]. An instance of CVP

may be represented conveniently by a sequence of boolean assignments of the form

$$\begin{aligned} X_0 &:= 0 \\ X_1 &:= 1 \\ X_i &:= X_{L_i} \diamond_i X_{R_i}, \quad i = 2, \ldots, m, \end{aligned} \tag{1}$$

where the operator $\diamond_i$ is either AND or OR. Also, $1 \leqslant L_i < R_i < i$ and every variable X_i occurs exactly once on the left-hand side of an assignment. We have to determine the value of X_m. Note that the size of an instance is $\Theta(m \log m)$.

For the sake of this construction it is convenient to think each cell as being subdivided into six cells, so that a configuration of length n resembles a rectangular matrix of size 6 by n. Again, we will refer to the rows of this matrix as tracks. The top track holds an instance of CVP, properly coded as a word over $\Sigma_1 = \{0, 1, \vee, \wedge, b, \#\}$. Tracks three and five are communication channels used to send boolean values. Tracks two, four, and six play an auxiliary role. The alphabet used in track i is Σ_i, where $\Sigma_2 = \{b, 0, 1\}$, $\Sigma_3 = \Sigma_5 = \{b, T, F\}$, and $\Sigma_4 = \Sigma_6 = \{b, r, a, w\}$. The alphabet for the whole automaton is $\Sigma = \Sigma_1 \times \Sigma_2 \times \cdots \times \Sigma_6 \cup \{\top, \bot\}$. Symbol b will be referred to as blank.

Let $k := \lceil \log m \rceil$ and define $\underline{x}$ to be the r-digit binary expansion of x, $0 \leqslant x < 2^r$. For any word W let $\mathrm{rev}(W)$ denote its reversal. Instance (1) is coded by a configuration $Z(1)$ of length $n = 3(m-1)(k+1)$. The first track of $Z(1)$ has the form

$$bb \cdots b\underline{L_2} \# \underline{R_2} \diamond_2 \underline{2} \# \underline{L_3} \# \underline{R_3} \diamond_3 \underline{3} \# \cdots \# \underline{R_m} \diamond_m .$$

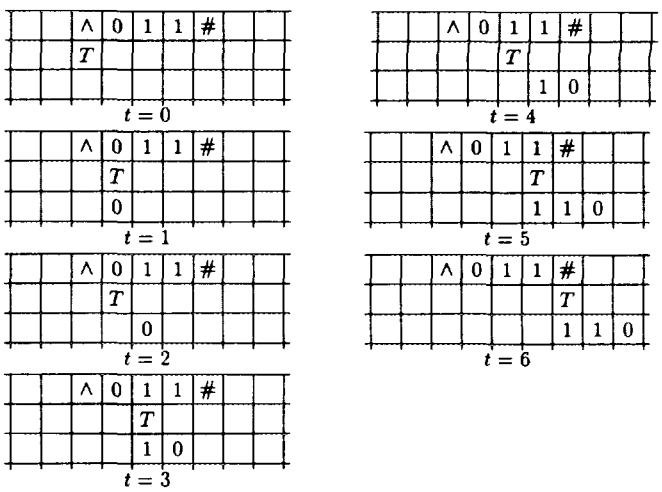
The other tracks contain only blanks except for some cells on the left in tracks 3 and 5 and some markers w placed in tracks 4 and 6 in cells containing a $\#$; see below.

We now give an informal description of the rule ρ. Rule ρ has width 3. In track 1, all symbols are left unchanged by ρ. In tracks 3 and 5 all nonblank symbols are moved one cell to the right at each step and traverse that part of the configuration that lies to the right of the point of origin of the signal. Specifically, signal $\mathrm{rev}(\underline{i}) \in \{0, 1\}^k$ in track 3 indicates that variable X_i evaluates to true. Similarly, if the signal occurs in track 5 variable X_i evaluates to false. The symbolds in the auxiliary tracks 2, 4, and 6 should be thought of as markers that can be placed in a cell and then moved (or removed) according to certain local conditions. There are three procedures that are implemented by ρ: evaluation, sending, and receiving.

Evaluation. Consider the block $\#\underline{L_i}\#\underline{R_i} \diamond_i \underline{i} \#$ that represents equation $X_i = X_{L_i} \diamond_i X_{r_i}$. Suppose that the value of variable X_{L_i} has already been obtained and stored in track 2 in the cell to the left of $\diamond_i$ that (by means of a marker T or F placed there; T stands for true and F for false). When the value for variable X_{R_i} is received in track 3 or 5, X_i can be determined. Correspondingly, a marker T or F will be placed in track 2 under $\diamond_i$. This marker then causes a signal of the form $\mathrm{rev}(\underline{i}) \in \{0, 1\}^k$ to be transmitted in track 3 (if the marker is T) or in track 5 (if the marker is F).

Sending. Suppose the value of a variable X_i has just been obtained as described above. Suppose for the sake of simplicity that X_i is true and therefore the cell containing $\diamond_i$ holds a marker T in track 2. All the following comments apply similarly to a marker F. This marker then moves to the right and causes symbol $\sigma \in \{0, 1\}$ from track 1 to be copied into track 3. Marker T never moves into a cell that already contains a non-blank symbol in track 3 and stops at the first $\#$ in track 1.

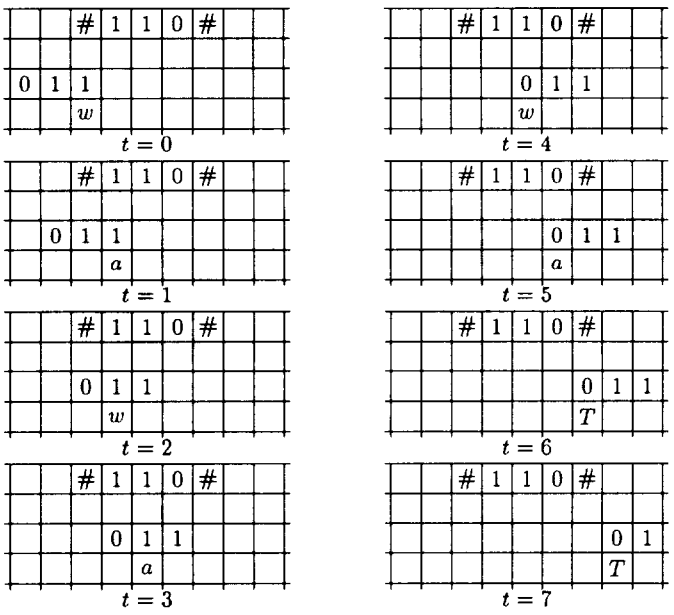
The following figure shows how a signal 110 is generated. Only the first three tracks are shown and the blanks are omitted. The logical operator here is $\wedge$.



Receiving. Suppose a signal $W \in \{0, 1\}^k$ is moving down, say, track 3. The block representing equation number i has to test whether W matches up with the binary expansions of L_i or R_i as stored in track 1. To this end a marker w (for wait) is positioned in track 4 under the $\#$ preceding the binary number. If and when the signal arrives the marker starts moving to the right. Whenever a match occurs between tracks 1 and 3 the marker moves one cell to the right and changes to a. In the next step, a returns to w (without movement). This is necessary to synchronize the marker with the advancing signal. If a mismatch occurs w turns into the reset marker r. The reset marker then moves to the left and stops when it reaches the first $\#$. There it turns into w and remains stationary until the next signal arrives.

The following figure shows how signal 110 in track 3

(corresponding to the instruction: set variable number 6 true) is received in a group cell representing the left variable in an equation. Only tracks 1, 3, and 4 are shown and the blanks are omitted.



At time 6, a marker T is placed in track 2 under the second #. The marker then moves to the right and stops in the last cell before the next operator symbol $\diamond_i$ in track 1. When the value for the second variable is received this allows us to evaluate the ith equation as described above.

All local configurations other than the ones described will be mapped to $(b, ..., b)$ by ρ. To initialize the process of receiving–evaluation–sending, a signal 0^k and 10^{k-1} (corresponding to the reverse binary expansions of 0 and 1) is sent in tracks 3 and 5, respectively. These signals start in the leftmost cells containing only blanks in track 1.

In $O(m \log m)$ steps the evolution of configuration $Z(I)$ under rule ρ will simulate the execution of the assignments in I: starting at $i = 2$ the value of X_i is computed and communicated via tracks 3 and 5 to all equations with numbers larger than i, allowing these equations to be evaluated, to send out their results in turn, and so forth. Evaluation takes constant time once both input values have been obtained. Sending and receiving in any block takes $O(\log m)$ steps, where the constants are independent of the instance. Thus after $O(m \log m)$ a pattern occurs in which the last variable has evaluated to true iff variable X_m in instance (I) evaluates to true. It is not hard to see that, due to the uniformity of the given construction, the configuration $Z(I)$ can be computed from the equations in (I) in logarithmic space. Hence linearly bounded RPEP is P-complete (with respect to log-space reductions) for the rule ρ is just described. ∎

It is straightforward to modify the construction to apply total target patterns. For example, in the last block that represents equation number m, the cell will switch to a special state σ if the last variable evaluates to true. The special state the propagates in linear time throught the whole configuration.

4. HIGHER-DIMENSIONAL CELLULAR AUTOMATA

As we have shown in the last section it is NLOG-complete to determine the existence of predecessor configurations on finite one-dimensional cellular automata. The problem becomes signigicantly more difficult (assuming that NLOG ≠ NP) for two-dimensional cellular automata: PEP is NP-complete in this case. Our results trivially extend to all higher dimensions.

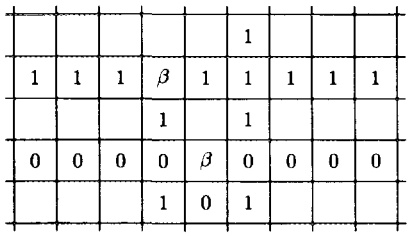
THEOREM 4.1. *There is a two-dimensional rule ρ such that the predecessor existence problem is* NP*-complete for finite cellular automata using ρ.*

Proof. Membership in NP was discussed in Section 2. We will show NP-hardness by embedding a variation of 3-Satisfiability called *monotone* 3-SAT; see [5]. An instance of monotone 3-SAT is a boolean formula Φ in 3-conjunctive normal form, where each clause contains either only negated variables or only unnegated variables. Let $\{x_1, ..., x_n\}$ be the boolean variables used in Φ. Thus $\Phi = \phi_1 \wedge \cdots \wedge \phi_m$, where $\phi_\nu = x_{\nu,1} \vee x_{\nu,2} \vee x_{\nu,3}$ for $1 \leqslant \nu \leqslant m_0$ and $\phi_\nu = \bar{x}_{\nu,1} \vee \bar{x}_{\nu,2} \vee \bar{x}_{\nu,3}$ for $m_0 < \nu \leqslant m$.

The key idea is to construct a rule ρ that tests satisfiability in one step: Φ can be translated into a configuration Y_Φ such that Y_Φ has a predecessor X_Φ iff Φ is satisfiable. The predecessor configuration X_Φ will code a satisfying truth-assignment and a proof that the assignment really satisfies Φ.

Our alphabet is $\Sigma = \{0, 1, \#, \$, \beta, T, F\}$ and the size of configuration Y_Φ is $2n + 1$ by $6m$. It is easier to describe first a typical configuration X_Φ. Rows 1, 3, ..., $2n - 1$ in X_Φ contain either only 0's or only 1's plus possibly a number of β's. These rows correspond to the boolean variables in Φ and will be referred to as truth-setting rows. The truth-setting rows are separated by rows containing #'s everywhere, except at certain columns called signal columns. Signal columns are used to transmit the value of a truth-setting row to the bottom row of X_Φ. The cell where a signal column connects to a truth-setting row contains the special symbol β and is called the base point of the signal column. These are m groups of three immediately adjacent signal columns. The νth group specifies the values of the variables in clause ϕ_ν. Lastly, the bottom row contains $m_0 T$'s in positions 3, 9, ..., $3 + 6(m_0 - 1)$ and $m - m_0$ F's in positions $3 + 6m_0$, ..., $3 + 6(m - 1)$. All other cells in the last row contain a #. A typical segment of configuration X_Φ is shown in the figure below. The segment contains three signal columns

and two base points. Cells containing # are left blank for the sake of clarity.



We now describe the local map ρ used in the automaton. Rule ρ has width 5 and a local configuration is admissible if it can occur as a subconfiguration of any of the configurations X_Φ just described. It is straightforward—although quite tedious—to give an explicit description of admissible local configurations. The default symbol for missing cells is #; all inadmissible configurations are mapped to \$. Every admissible local configuration $W: [-2, 2]^2 \to \Sigma$ with $W(0, 0) = \sigma \in \{\beta, \#\}$ is mapped to σ. Thus, the truth-values chosen in X_Φ are destroyed by ρ by the layout of X_Φ, including base points and separators, remains visible. The actual truth-testing occurs on configurations W with $W(0, 0) \in \{T, F\}$ that are centered at positions $(2n+1, 3+6i)$ in the last row of X_Φ. For $W(0, 0) = T$ we set

$$\rho(W) := \begin{cases} T, & \text{if } W(-1, j) = 1 \text{ for some } j \in \{-1, 0, 1\}, \\ \$, & \text{otherwise.} \end{cases}$$

Similarly for $W(0, 0) = F$, let

$$\rho(W) := \begin{cases} T, & \text{if } W(-1, j) = 0 \text{ for some } j \in \{-1, 0, 1\}, \\ \$, & \text{otherwise.} \end{cases}$$

Hence a local configuration of the form

1	1	1	β	1
#	0	0	1	#
#	#	T	#	#
#	#	#	#	#
#	#	#	#	#

is mapped to T by rule ρ.

Now define Y_k exactly like X_Φ except that all 1's are replaced by 0's. Suppose that X is any predecessor configuration of X_Φ. Since Y_Φ contains no \$'s, configuration X cannot contain inadmissible local configurations. In fact, it is not difficult to see that the only differences between X and X_Φ can occur in cells c such that $X_\Phi(c) = 0$: in that case $X(c) = \{0, 1\}$. Define a truth assignment $\alpha: \{x_1, \ldots, x_n\} \to \{0, 1\}$ by $\alpha(x_i) = \text{true}$ iff row $2i-1$ contains only 1's (and some β's). It is easy to see that α satisfies Φ. Conversely, any satisfying truth assignment for Φ can be translated into a predecessor configuration for Y_Φ. Hence monotone 3-SAT is polynomial time reducible to PEP for this rule ρ and PEP is NP-complete for dimension 2. We note in passing that the reduction just given is in fact in log-space rather than just polynomial time. ∎

TABLE III

Type of cellular automaton	Complexity of PEP
finite, 1-dim	NLOG-complete
finite, 2-dim	NP-complete
infinite, 1-dim	P
infinite, 2-dim, finite predecessor	r.e.-complete
infinite, 2-dim, infinite predecessor	co-r.e.-complete

Observe that the last hardness result holds for one fixed rule ρ—unlike Theorem 3.1 which establishes hardness only for the uniform problem.

5. CONCLUSION

We have shown that various decision problems associated with the evolution of configurations on finite cellular automata are complete in their natural complexity classes. In particular, determining the existence of a predecessor configuration is complete in the nondeterministic classes NLOG and NP, depending on the dimension of the automaton. Together with the results in [25], the complexity of the predecessor existence problem is summarized in the Table III. Note that no lower bound is known for 0-finite configurations on infinite, one-dimensional cellular automata. Of course, the problem remains undecidable for all dimensions higher than one.

The pattern reachability problem is, in general, PSPACE-complete for one-dimensional cellular automata. For predictable rules, PREP is NP-complete for target patterns. We do not know whether there are predictable rules for which PREP remains NP-complete when the target pattern is required to be a configuration. Nonpredictable rules exist iff P is different from PSPACE. Thus, any attempt at a classification of finite cellular automata along the lines of Wolfram and Culik and Yu will not only lead to undecidable classes but will also produce a hierarchy that may collapse; see [11, 22, 19].

REFERENCES

1. J. Albert and K. Culik II, A simple universal cellular automaton and its one-way and totalistic version, *Complex Systems* **1**, No. 1 (1987), 1-16.

2. S. Amoroso and Y. N. Patt, Decision procedures for surjectivity and injectivity of parallel maps for tesselation structures, *J. Comput. System Sci.* **6** (1972), 448–464.
3. A. W. Burks, "Essays on Cellular Automata," Univ. of Illinois Press, Urbana, 1970.
4. S. R. Buss, C. H. Papadimitriou, and J. N. Tsitsiklis, On the predictability of coupled automata: An allegory about chaos, *Complex Systems* **5** (1991).
5. M. R. Garey and D. S. Johnson, "Computers and Intractability," Freeman, San Francisco, 1979.
6. L. M. Goldschlager, The monotone and planar circuit value problems are log-space complete for *p*, *SIGACT News* **9**, No. 2 (1977), 25–29.
7. U. Golze, Differences between 1- and 2-dimensional cell spaces, *in* "Automata, Languages and Development" (A. Lindenmayer and G. Rozenberg, Eds.), pp. 369–384, North- Holland, Amsterdam, 1976.
8. F. Green, NP-complete problems in cellular automata, *Complex Systems* **1**, No. 3 (1987), 453–474.
9. L. P. Hunt, Formal language characterizations of cellular automata, *Complex Systems* **1**, No. 1 (1987), 69–80.
10. K. Culik II, On invertible cellular automata, *Complex Systems* **1**, No. 6 (1987), 1035–1044.
11. K. Cullik II and S. Yu, Undecidability of CA classification schemes, *Complex Systems* **2**, No. 2 (1988), 177–190.
12. N. D. Jones, Space-bounded reducibility among combinatorial problems, *J. Comput. System Sci.* **11** (1975), 68–85.
13. J. Kari, Reversibility of 2D cellular automata is undecidable, *Physica D* **45** (1990), 397–385.
14. O. Martin, A. M. Odlyzko, and S. Wolfram, Algebraic properties of cellular automata, *Commun. Math. Phys.* **93** (1984), 219–258.
15. M. Nasu, Local maps inducing surjective global maps in one-dimensional tesselation automata, *Math. Systems Theory* **11** (1978), 327–351.
16. L. J. Stockmeyer, Classifying the computational complexity of problems, *J. Symbolic Logic* **52**, No. 1 (1987), 1–43.
17. K. Sutner, On σ-automata, *Complex Systems* **2**, No. 1 (1988), 1–28.
18. K. Sutner, A note on Culik–Yu classes, *Complex Systems* **3**, No. 1 (1989), 107–115.
19. K. Sutner, Classifying circular cellular automata, *Physica D* **45**, Nos. (1–3) (1990), 386–395.
20. K. Sutner, De Bruijn graphs and linear cellular automata, *Complex Systems* **5**, No. 1 (1991), 19–30.
21. K. Sutner, "Linear Cellular Automata and Their Fischer Automata," Report Series 93-46, RISC-Linz, August 1993.
22. S. Wolfram, Computation theory of cellular automata, *Comm. Math. Phys.* **96**, No. 1 (1984), 15–57.
23. S. Wolfram, Computer software in science and mathematics, *Sci. Amer.* **251**, No. 3 (1984), 188–203.
24. S. Wolfram, Universally and complexity in cellular automata, *Physica D* **10** (1984), 1–35.
25. T. Yaku, The constructibility of a configuration in a cellular automaton, *J. Comput. System Sci.* **7** (1973), 481–496.