ON THE PREDICTABILITY OF COUPLED AUTOMATA: AN ALLEGORY ABOUT CHAOS[1]

Sam Buss[2], **Christos H. Papadimitriou**[3], and **John Tsitsiklis**[4]

Abstract

We show a sharp dichotomy between systems of identical automata with a symmetric global control whose behavior is easy to predict, and those whose behavior is hard to predict. The division pertains to whether the global control rule is invariant with respect to permutations of the states of the automaton. On the other hand, we show that testing whether the global control rule has this invariance property is an undecidable problem. We argue that there is a natural analogy between complexity in our model and chaos in dynamical systems.

---

1. Research supported by the NSF under Grant ECS–8552419, with matching funds from Bellcore Inc. and Du Pont Inc., the NSF under Grant DMS–8902480, and the ARO under Grant DAAL03–86–K–0171. A preliminary version of this paper was presented at the 31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, October 1990.

2. Department of Computer Science and Engineering, University of California at San Diego, La Jolla, California 92093.

3. Department of Computer Science and Engineering, University of California at San Diego, La Jolla, California 92093. Part of this author's research was performed while visiting the Center for Intelligent Control Systems at M.I.T.

4. Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.

1. INTRODUCTION AND MOTIVATION

Consider an array of five automata identical to the one in Figure 1. The automata operate in unison, and at each time step they all have identical inputs. The input is determined from the states of the automata in terms of the following global control rule:

Figure 1: Automaton M.

"global input is 1 *if at least one of the* n *automata is in state* q_1*, and no more than two automata are in the same state ; otherwise, it is* 0*."*

Suppose that the system starts at the state $(q_2, q_3, q_1, q_2, q_3)$ (or $(q_1, q_2, q_2, q_3, q_3)$, since the rule does not depend on the identities of the automata there is no difference). What is the state of the system after ten moves? A thousand moves?

This is an instance of the state prediction problem studied in this paper. We are given n identical automata, a global control rule, an initial state n-vector, and an integer T, and we are asked to compute the state vector after T steps. The global control rule is given in terms of a first-order sentence. For example, the rule in the example above can be written $(N(q_1) \geq 1) \wedge \forall x(N(x) \leq 2)$, where $N(x)$ stands for the number of automata at state x. (We refer to $N(x)$ as the *multiplicity* of state x.) As in this example, we will only consider global control rules that only depend on the multiplicities of the different states; that is, *the control rule is independent of the identities of the automata*.

We wish to study how the complexity of this problem (intuitively, the predictability of the system) depends on the nature of the global control rule. We consider a system predictable if the answer to the above question can be obtained in time polynomial in the number of states, the number of automata, and the logarithm of T. In contrast, if the prediction problem is PSPACE-complete, this would mean essentially that the system is not easily predictable, and that there seems to be no better prediction method other than simulation.

In this paper we *draw a sharp boundary* between rules in our language that are polynomial-time predictable, and those that are PSPACE-complete. We show that, *all constant-free rules* (that is, rules that do not involve state constants such as q_1 in our example) *are polynomial-time predictable*, while *all rules that inherently use constant symbols lead to PSPACE-complete prediction problems*. Our polynomial algorithm uses a simple monotonocity property of constant-free rules. It is explained in Section 2. Our lower bound uses an increasingly accurate characterization of non-constant-free rules to

essentially reduce any such rule to a rule of the form $N(q_1) = n_1$, where n_1 is an integer constant. We then show that a rule of the latter form leads to a PSPACE-complete prediction problem. These results are presented in Section 3. Finally, in Section 4, we show that testing for constant–freeness is an undecidable problem.

Motivation

In the remainder of this section, we shall discuss dynamical systems and chaos, the subject that has motivated this work.

It is well-known that dynamical systems differ dramatically in several related important aspects of their behavior, such as periodicity, predictability, stability, dependence on initial values, etc. Systems that are "nasty" in these respects (in a fairly well-defined sense) are called "chaotic" ([1], [2]). There are many important, and sometimes deep, facts known about chaotic dynamical systems. Unfortunately, there seems to be no clear characterization of the circumstances that give rise to chaos, and systems that appear very similar have very different properties in this respect. We wish to shed some light in this problem by studying discrete computational analogs of chaos in dynamical systems. We have been influenced in our direction and precise model by several precursors and considerations, briefly explained below.

There have been interesting attempts to use discrete, computational analogs to understand chaos. Most notably, Wolfram [4] has used cellular arrays of automata (generalizations of the Game of Life), and observed similar behavior: Some arrays are easy and predictable, while some others are difficult to figure out and predict, just as in chaotic dynamical systems. There is rather informed and competent discussion in [4] of important computational issues (including randomness and PSPACE-completeness) in relation to this paradigm. However, what is lacking from the analysis of [4] is *a reasonably sharp dichotomy* between cellular automata that exhibit chaotic behavior and those that do not. Such a result would have made the analogy much more valuable. The difficulty in proving such a result is not hard to understand: Cellular arrays have essentially a single parameter (the automaton), and it is very unlikely that finite automata show a sharp division between those that can simulate space-bounded computation and those that exhibit a periodic, predictable bahavior.

Roughly speaking, chaotic behavior seems to appear in systems of very few degrees of freedom (e.g., the Lorenz oscillator) in which nonlinearities have subtle effects, as well as in systems with a very large number of degrees of freedom. Natural discrete analogs of the first class would be complex centralized models of computation such as Turing machines, but of course such models do not yield themselves to syntactic characterizations. (We do discuss below, however, how our model can be thought of as akin to discrete-time dynamical systems with one degree of freedom.) Seeking a discrete analog of the second class, we decided that a large number of degrees of freedom is best reflected in a large number of interacting automata. In fact, such systems are characterized by two parameters (the automaton and the interaction rule) therefore making a sharp dichotomy more likely.

Chaos often arises in coupled oscillators [1]. It is not unreasonable to view a finite automaton as the discrete analog of an oscillator, since, when operating in isolation, it is guaranteed to be eventually periodic. The coupling of oscillators can be, in general, of two different types: Local coupling between nearest neighbors, or of a more global nature. The

first type of coupling is captured by cellular arrays (as in [4]), whereas our formulation captures the second type. As an example of the second type, one could imagine that a set of otherwise decoupled oscillators (automata) generate a "mean field" which in turn affects the behavior of each automaton. Our identity–independence assumption can be viewed as an assumption that the "mean–field" is spatially homogeneous and is independent of the spatial configuration of the oscillators that generate it. Our results imply that such a system would be inherently unpredictable.

A different but related framework for studying chaos is that of discrete-time dynamical systems, such as $x_{t+1} := x_t \cdot (1 - x_t)$. State variable x_t can be thought of as a large sequence of digits in some large base. Suppose that we wish to study rules that treat each such digit independently (at a loss of continuity, of course). That is, the next state x_{t+1} is computed by looking at the values of each digit of x_t, and then modifying each digit, according to some law. Informally speaking, the laws that are most chaos-prone are those that treat all digits the same, independently of the order: Such evenhandedness implies sensitive dependence on the initial data, as the first few digits fail to convey most of the information about x_t. This corresponds precisely to our genre of rules. Our results suggest that almost all such laws lead to chaotic behavior (except for those that are insensitive to the magnitude of the digits treated).

Finally, the model considered here can be tied to problems of supervisory control of discrete–event dynamical systems [3]. Suppose that the automata are identical machines operating in a cyclical fashion (following the 1 arrows) except that whenever any machine enters a special state, some corrective action has to be taken (e.g., temporarily cut the power supply) which causes an abnormal transition at all machines. Our results show that the long–run effects of such supervisory control can be very hard to predict by methods other than simulation. To go even further in this direction, imagine that we wish to study the effect of government decisions on a certain population. We may wish to model each individual as being in one of a set of states (happy, risk-prone, conservative, subversive, etc.). Decisions will be based on opinion polls of the population (the $N(q_i)$'s). Unless we know the nature of the state space and the transitions, the only rules that will lead to predictable behavior seem to be ones that do not distinguish between different moods of the population!

An intriguing aspect of our results is that on the one hand we derive a fairly simple property characterizing those global control rules that lead to unpredictable behaviour; on the other hand, verifying this property is an undecidable problem. This seems to suggest that it could be impossible to derive effective criteria for deciding whether a dynamical system exhibits chaotic behavior or not.

2. POLYNOMIAL PREDICTABILITY

We have an array of n automata, each identical to $M = (K, \{0,1\}, \delta)$, where $K = \{q_1, \ldots, q_{|K|}\}$ is a finite set of *states*, the *input alphabet* is, for simplicity, always $\{0,1\}$, and $\delta : K \times \{0,1\} \mapsto K$ is the *transition function* of M.

The automata are controlled by a *global control rule* R. R is a sentence in a first-order language defined next. Our language has constants $q_1, q_2, q_3, \ldots$ and variables $x, y, z, \ldots$; both range over K, the set of states of M. Terms are of the form $N(s)$, where s is a

constant or a variable and N is a special symbol. (The meaning of $N(s)$ is "the number of automata in state s.") Atomic formulae are linear equations and inequalities of terms, such as: $N(x) + 2 \cdot N(q_3) - 3 \cdot N(y) \leq 5$ or $N(x) = N(q_1)$. Formulae then are formed from atomic ones by Boolean connectives and quantifiers of the form $\forall x$ and $\exists y$. A rule is a formula without free variables (standard definition). Examples of rules are these:

R_1 = "$N(q_1) = 0$";

R_2 = "$\forall x(2 \cdot N(x) \leq N(q_3))$";

R_3 = "$\forall x \exists y((N(x) + N(y) \geq 3) \vee (N(x) = 0))$";

R_4 = "$\forall x(N(x) = N(q_1))$".

Suppose that M is an automaton and R is a rule. We say that M is *appropriate for* R if all constants mentioned in R occur as states of M. A *global state* of a system consisting of n copies of M is an element of K^n. A global state S gives rise to its *poll* $N(S) = (N(q_1), \ldots, N(q_{|K|}))$, a sequence of $|K|$ nonnegative integers adding up to n, where $N(q_i)$ is the number of occurrences of state q_i in S (the multiplicity of q_i). Such a poll is said to be appropriate for R if it is obtained from an automaton M which is appropriate for R. If N is a poll appropriate to R, we write $N \models R$ if the multiplicities $N(q_i)$ of the states of M satisfy sentence R (the standard inductive definition of satisfaction). We say that two rules R and R' are *equivalent* if for all N appropriate to both R and R' we have: $N \models R$ iff $N \models R'$.

Notice that R_1 and R_2 above explicitly mention constants, whereas R_3 does not. R_4 does mention a constant, but this is not inherent; R_4 is equivalent to R'_4 = "$\forall x \forall y(N(x) = N(y))$". We call a rule *constant-free* if it is equivalent to a rule that does not contain constants in its text.

The operation of the system is the following: The global state $S(t) = (s_1(t), \ldots, s_n(t))$ determines its poll $N(S(t))$, which in turn determines the *global input* $I(t)$; in particular, $I(t) = 1$ if $N(S(t)) \models R$, and $I(t) = 0$ otherwise. $I(t)$ then determines the next state $s_i(t+1) = \delta(s_i(t), I(t))$ in each automaton, and so on. We thus arrive at the following computational problem:

STATE PREDICTION-R: We are given a positive integer n, an automaton $M = (K, \{0,1\}, \delta)$, an initial state vector $S(0) = (s_1(0), \ldots, s_n(0)) \in K^n$, and an integer $T > 0$. We are asked to determine $S(T)$.

Theorem 1: If R is constant-free, STATE PREDICTION-R can be solved in polynomial time.

Proof: Suppose that $S = (s_1, \ldots, s_n)$ is a global state. The *type* of S, $\tau(S)$, is the sorted poll of S. That is, the type of S captures the multiplicities of states in S, but without identifying each multiplicity with a state. For example, if $K = \{a, b, c\}$ and $S = (a, a, b, a, b)$, then the type of S is $\{0, 2, 3\}$. We say that S' is a *degradation* of S if the type of S' can be obtained from that of S by replacing some groups of non-zero numbers by their respective sums, and with sufficient 0's to make $|K|$ numbers. For example, $\{0, 0, 0, 1, 5\}$ is a degradation of $\{0, 1, 1, 1, 3\}$. If R is constant-free, then it is easy to see that whether $N(S) \models R$ only depends on $\tau(S)$.

Suppose that we know $S(t)$, the global state at time t. We then know whether $N(S(t)) \models R$, and thus we know $I(t)$, and it is easy to compute $S(t+1)$. It is easy to see that, since all automata are identical and they obtain the same input, the next state in each is uniquely determined by the current one. Hence, either the type of $S(t+1)$ is the same as that of $S(t)$ (if $I(t)$ happens to map all states present at $S(t)$ in a one-to-one fashion to new states), or $S(t+1)$ is a degradation of $S(t)$. If we encounter a degradation, we pause (the current stage is finished). Otherwise, we keep on simulating the system for $|K|$ moves, with the same input (since the type remains the same).

At this point (that is, after $|K|$ moves), each automaton has entered a loop, since there are only $|K|$ states. All these loops are disjoint, and therefore there will never be a change in the type of the global state (and hence in the global input). Each automaton has become periodic, with period at most K, and we can solve the state prediction problem very easily.

Suppose now that we have a degradation. We repeat the same method, simulating the system either for $|K|$ moves, or until a degradation occurs. This must end after $|K|$ such stages, since each degradation introduces a new zero in the type of $S(t)$. Therefore, we can predict the state after simulating the system for at most $|K|^2$ moves. □

On the subject of polynomial algorithms, it is easy to show the following:

Proposition 1: If there is a constant k such that the number of states of M is at most k *or* the number of copies of M is at most k, then STATE PREDICTION-R is polynomially solvable for all R. □

Thus, our constructions of PSPACE-completeness in the next section will necessarily employ an unbounded number of copies of large automata.

3. PSPACE-COMPLETENESS

We shall show that all non-constant-free rules in some sense reduce to non-constant-free rules of a very simple form. We must first understand the "model theory" of non-constant-free rules.

Definition: The *range* of a poll N is the set $\{N(q_1), \ldots, N(q_k)\}$ of state multiplicities. Two polls are said to be *compatible* if they have the same range. If N is a poll on states $\{q_1, \ldots, q_k\}$ then N' is an *extension* of N iff N' is a poll on states $\{q_1, \ldots q_m\}$ with $k \leq m$ and $N(q_i) = N'(q_i)$ for all $i \leq k$.

The notation $R(q_{i_1}, \ldots, q_{i_k})$ will be used only when $q_{i_1}, \ldots, q_{i_k}$ indicates all of the occurences of state names in the rule R. Thus, for instance, $\exists x_1 \cdots \exists x_k R(x_1, \ldots, x_k)$ is constant-free. However, the notation does not require $q_{i_1}, \ldots, q_{i_k}$ to be distinct.

Lemma 1: Suppose N and N' are compatible polls and $R(q_{i_1}, \ldots, q_{i_k})$ is a rule and that $j_1, \ldots, j_k$ are such that $N(q_{i_n}) = N'(q_{j_n})$ for all $1 \leq n \leq k$. Then $N \models R(q_{i_1}, \ldots, q_{i_k})$ if and only if $N' \models R(q_{j_1}, \ldots, q_{j_k})$.

Proof: By induction on the number of logical connectives in R. For R atomic, Lemma 1 is obvious since atomic formulas are linear combinations of $N(q_i)$'s. The case where R has outermost connective a propositional connective is immediate from the induction

hypothesis. If R is $\exists x S(x, q_{i_1}, \ldots, q_{i_k})$, then

$$\begin{aligned} N \models R &\iff N \models S(q_{i_0}, q_{i_1}, \ldots, q_{i_k}) \quad \text{for some } i_0 \\ &\iff N' \models S(q_{j_0}, q_{j_1}, \ldots, q_{j_k}) \quad \text{for some } j_0 \\ &\iff N' \models R(q_{j_1}, \ldots, q_{j_k}) \end{aligned}$$

where the middle equivalence follows from the compatibility of N and N' and the induction hypothesis. The case $R = \forall x S$ is handled by noting that R is equivalent to $\neg\exists x \neg S$. □

Lemma 2: If $R(q_1, \ldots, q_k)$ is a rule, N is a poll of length $\geq k$, and N' is a compatible extension of N then $N \models R$ iff $N' \models R$.
Proof: Lemma 2 is an immediate corollary of Lemma 1. □

Definition: A rule R is *preserved by permutations* iff for any polls N and N' with N' a permutation of N,

$$N \models R \iff N' \models R.$$

More generally, R is *preserved by compatible polls* iff for all compatible N and N',

$$N \models R \iff N' \models R.$$

Lemma 3: Let $R = R(q_1, \ldots, q_k)$. The following are equivalent:
(a) R is constant-free.
(b) R is preserved by permutations.
(c) R is preserved by compatible polls.
(d) $R(q_1, \ldots, q_k)$ and $\forall x_1 \cdots \forall x_k R(x_1, \ldots, x_k)$ and $\exists x_1 \cdots \exists x_k R(x_1, \ldots, x_k)$ are equivalent rules.

Proof: The implication (d)⇒(a) is obvious. To show (a)⇒(b) suppose R is constant-free and that N and N' are compatible polls. Since R is constant free, there is a rule S containing no constants which is equivalent to R. By Lemma 1, $N \models S$ if and only if $N' \models S$; i.e., $N \models R$ if and only if $N' \models R$. Thus R is preserved by compatible polls.

Next we show (b)⇒(c). Suppose R is preserved by permutations. Let N and N' be compatible polls which we can express as $N = (G)$ and $N' = (G')$ where G and G' are sequences of state multiplicities. Then

$$\begin{aligned} (G) \models R &\iff (G, G') \models R \quad \text{by Lemma 2} \\ &\iff (G', G) \models R \quad \text{by preservation by permutations} \\ &\iff (G') \models R \qquad \text{by Lemma 2.} \end{aligned}$$

Thus $N \models R$ if and only if $N' \models R$.

To show (c)⇒(d), we shall first assume that (c) holds and that N is a poll such that $N \models \exists \vec{x} R(\vec{x})$ and show that $N \models \forall \vec{x} R(\vec{x})$. Because $N \models \exists \vec{x} R(\vec{x})$, there are (not necessarily distinct) states $q_{i_1}, \ldots, q_{i_k}$ such that $N \models R(q_{i_1}, \ldots, q_{i_k})$. Letting $q_{j_1}, \ldots, q_{j_k}$ be arbitrary states, we need to show that $N \models R(q_{j_1}, \ldots, q_{j_k})$. For this purpose, find an extension N' of N such that there are distinct indices $m_1, \ldots, m_k$ so that $N'(q_{m_n}) = N(q_{i_n})$ for all $1 \leq n \leq k$, and find a poll N'' compatible with N' so that $N''(q_{m_n}) = N(q_{j_n})$ for all n.

(Note that the sole reason for introducing N' was because of the non-distinctness of the i_n's; the definition of N'' is possible only since the m_n's are distinct.) Now,

$$\begin{aligned} N \models R(q_{i_1}, \ldots q_{i_k}) &\iff N' \models R(q_{m_1}, \ldots q_{m_k}) \quad \text{by Lemma 1} \\ &\iff N'' \models R(q_{m_1}, \ldots q_{m_k}) \quad \text{preservation by compatible polls} \\ &\iff N \models R(q_{j_1}, \ldots q_{j_k}) \quad \text{by Lemma 1.} \end{aligned}$$

(For the middle equivalence above we know that $R(q_{m_1}, \ldots, q_{m_n})$ is preserved by compatible pools since $R(q_1, \ldots, q_k)$ is and since the property of being preserved by compatible polls is preserved by renaming of states.) Thus we have established that if (c) holds then $\exists \vec{x} R(\vec{x})$ implies $\forall \vec{x} R(\vec{x})$; hence, the following chain of implications holds for every poll:

$$\begin{aligned} \forall x_1 \cdots \forall x_k R(x_1, \ldots, x_k) &\implies R(q_1, \ldots, q_k) \\ &\implies \exists x_1 \cdots \exists x_k R(x_1, \ldots, x_k) \\ &\implies \forall x_1 \cdots \forall x_k R(x_1, \ldots, x_k). \end{aligned}$$

So the three rules are equivalent. □

Lemma 4: If R is not constant free, then there are polls $N = (G, n_1, G')$ and $N' = (G, n_2, G')$ such that $N \models R$ and $N' \not\models R$ and such that n_1 and n_2 both occur in G'.
Proof: By Lemma 3, R is not preserved by permutations and since permutations are generated by 2-cycles, there must be polls $N_1 = (G_1, n_1, G_2, n_2, G_3)$ and $N_2 = (G_1, n_2, G_2, n_1, G_3)$ such that $N_1 \models R$ and $N_2 \not\models R$. Thus, by Lemma 2,

$$(G_1, n_1, G_2, n_2, G_3, n_1, n_2) \models R$$

and

$$(G_1, n_2, G_2, n_1, G_3, n_1, n_2) \not\models R.$$

Now consider the poll $(G_1, n_1, G_2, n_1, G_3, n_1, n_2)$. If this poll makes R true then the lemma holds with G equal to G_1 and G' equal to G_2, n_1, G_3, n_1, n_2. On the other hand, if the poll makes R false, then the lemma holds with G equal to G_1, n_1, G_2 and G' equal to G_3, n_1, n_2 (and the roles of n_1 and n_2 interchanged). □

For notational convenience, we can assume that the two polls of Lemma 4 are of the form (n_1, N) and (n_2, N), where $N = (\bar{n}_2, \ldots, \bar{n}_D)$ denotes the remaining poll. (This can be always achieved by renaming of the states.) In the proof that follows, we will construct a system of automata whose poll at any time will be of the form (n_1, N, N') or (n_2, N, N'). The segment N of the poll will never change, and will stay equal to $(\bar{n}_2, \ldots, \bar{n}_D)$. The entries of N' will change with time but they will be taking values only in the set $\{n_1, n_2\}$. Since both n_1 and n_2 occur in N (Lemma 4), at any time we will be dealing with an extension of (n_1, N) or (n_2, N). By Lemma 2, it follows that R will be satisfied at exactly those times when the system's poll is of the form (n_1, N, N').

Theorem 2. Let R be a non-constant-free rule. Then STATE PREDICTION-R is PSPACE-complete.

Proof: Let M be a Turing Machine that operates on a circular tape with B tape squares. Let A be the alphabet size, and let the alphabet elements be $0, 1, \ldots, A-1$. Let Q be the number of states of M. We assume that each transition of M moves the tape head to the right by one square. Finally, we assume that the Turing Machine has a special "halting" configuration: once the tape and machine get to that configuration the machine state and the tape contents never change. It is easily shown that the problem of determining whether the above described Turing Machine ever reaches the halting configuration is PSPACE–complete.

The transitions of M can be described in terms of $P = AQ$ many *transition rules* of the form: "if M is in state m and the tape symbol is a, then a gets overwritten by a' and the new state of M is m'." Thus, at any step the machine tries each one of the transition rules, until it finds one which applies ("fires"), and then makes a transition. Notice, that the identity of the transition rule r to be fired determines completely the value of m and a. Furthermore, the transition rule r' to be fired at the next transition is completely determined by the transition rule r being fired now and the value in the tape square which is to the right of the tape head. (This is because r uniquely determines the state of M right after r is fired.) We assume that the transition rules have been numbered from 0 to $P-1$.

We now construct an instance of STATE PREDICTION–R that will encode and simulate the computation of M on the B tape squares. Our instance consists of a number (to be specified later) of identical finite state automata (FSAs), which we now construct.

There are certain states $q_2, q_3, ..., q_D$ that "do not move". (If an FSA starts at one of those states, it always stays there.) The initial multiplicities of these states are exactly the numbers $\bar{n}_i$ that correspond to N, where N was defined in the discussion following Lemma 3. For all of the remaining states, the state multiplicities will be initialized at either n_1 or n_2. Furthermore, the transitions of the FSAs will be specified so that the multiplicity at any one of these remaining states is always n_1 or n_2. It is useful to think of the states with multiplicity n_1 as "carrying a token." Thus, our transition rule R can be interpreted as: "R is true if and only if there is a token at state q_1."

We now specify the remaining states of the FSAs. We will have:

(i) States of the form (a, p, r), where a corresponds to a tape symbol ($0 \leq a < A$), p corresponds to a tape square ($1 \leq p < B$), and r corresponds to a transition rule ($0 \leq r < P$). State (a, p, r) is interpreted as follows: If R is true (i.e., if there is a token at q_1), then the presence of a token at state (a, p, r) (i.e., a multiplicity of n_1) indicates that there is a symbol "a" p squares to the right of the tape head and that transition rule r is about to be fired.

(ii) A special state $(-1, -1, -1)$ which will be needed later in order to apply the Chinese Remainder Theorem.

(iii) States of the form (s), where $0 \leq s \leq S = P(AP+1)^2$. These states are used for "synchronization." The state (0) is identified with the special state q_1. (That is, R is true if and only if (0) has multiplicity n_1, that is, when it has a token.) The transition rules of the automata will be defined so that exactly one of these states has a token and a transition of the Turing Machine will be simulated each time that this token gets to state (0) and global control rule R becomes true.

We initialize the FSAs so that state (0) has multiplicity n_1, and all states (s), with $s \neq 0$, have multiplicity n_2. Given an initial configuration of the Turing Machine, we encode this configuration as follows. Let r^* be the first transition rule to be applied. Then, a state (a, p, r) will have multiplicity n_1 if and only if $r = r^*$ and the symbol p squares to the right of the tape head is a. All other states of the form (a, p, r), as well as the state $(-1, -1, -1)$ have multiplicity n_2. Note that there is no set of states used to encode the contents of the tape square under the head; this information is already given by the transition rule r^*.

We now describe the transition rules for the FSAs.

1. If R is not satisfied (state (0) has multiplicity n_2):

1a: $(a, p, r) \longrightarrow (a, p, r + 1 \bmod P)$, if $1 \leq p < B - 1$, called "incrementing r mod P."
1b: $(a, B - 1, r) \longrightarrow (a, B - 1, r + 1)$, if $r + 1 < P$.
1c: $(a, B - 1, P - 1) \longrightarrow (a + 1, B - 1, 0)$, if $a < A - 1$.
1d: $(A - 1, B - 1, P - 1) \longrightarrow (-1, -1, -1)$.
1e: $(-1, -1, -1) \longrightarrow (0, B - 1, 0)$.
1f: $(s) \longrightarrow (s - 1 \bmod S)$.

According to rules 1b, 1c, 1d, and 1e, when $p = B - 1$, the automaton cycles through all states of the form $(a, B - 1, r)$, together with state $(-1, -1, -1)$. The number of states in the cycle is $AP + 1$ and we refer to these four rules as "incrementing mod $AP + 1$."

2. If R is satisfied (state (0) has multiplicity n_1):

2a: $(a, p, r) \longrightarrow (a, p - 1, r)$, if $p \neq 1$. This captures the movement of the tape head to the right, since the symbol a that was p squares to the right of the tape head is now $p - 1$ squares to the right.
2b: $(0) \longrightarrow (0, B - 1, 0)$.
2c: $(a, 1, r) \longrightarrow (N_{a,r})$.
2d: $(N_{a,r}) \longrightarrow (a, B - 1, r)$, if $(a, r) \neq (0, 0)$.
2e: $(N_{0,0}) \longrightarrow (0)$.
2f: $(s) \longrightarrow (s)$ if $s \neq 0$ and s is not of the form $(N_{a,r})$.

In rules 2c, 2d, 2e, and 2f, the numbers $N_{a,r}$, where $a = 0, 1, \ldots, A - 1$, and $r = 0, \ldots, P - 1$, are distinct positive integers which are chosen so as to have the following properties. Suppose that when transition rule r (of the Turing Machine) is fired, it writes a' in the tape square under the head. Furthermore, assuming that a is the tape symbol immediately to the right of the tape head, let r' be the transition rule to be fired at the next step. (As argued earlier, r' is uniquely determined by r and a.) Then:
(i) $N_{a,r} = (r' - r) \bmod P$. (So, incrementing any (a, p, r) mod P a total of $N_{a,r}$ times gives (a, p, r').
(ii) Incrementing $(0, B - 1, 0)$ mod $AP + 1$ a total of $N_{a,r}$ times yields $(a', B - 1, r')$.

Numbers $N_{a,r}$ with the above mentioned properties exist, by the Chinese Remainder Theorem, because $AP + 1$ and P are relatively prime.

We now explain how the FSAs simulate the Turing Machine. First, it is easily verified that all states (other than $q_2, \ldots, q_D$) have multiplicities n_1 or n_2 at all times. Consider a time when (0) has multiplicity n_1. Then, there are tokens at states (a, p, r) encoding the symbols in the tape squares (except for the symbol under the tape head), and also indicating that transition rule r of the Turing Machine is being fired. Rule R is satisfied

and the FSA transition rules 2a–2f are used. Right after that, the states (s) have all multiplicity n_2 except for one state ($N_{a,r}$) which receives a token from state $(a, 1, r)$, where a is the symbol one position to the right of the tape head at the time that r is fired. The next time that R will be satisfied will be when that token reaches state (0). Until then, the FSA transition rules 1a–1f are followed; due to rule 1f, it takes $N_{a,r}$ steps for the token to reach state (0). Because of rule 1a, a token at (a, p, r), for $p < B - 1$, gets incremented by $(r' - r)$ mod P leading to state (a, p, r'), as desired. Regarding the set of states of the form $(a, B - 1, r)$, when r was fired, the FSA transition rule 2b sent a token to state $(0, B - 1, 0)$. After $N_{a,r}$ steps, this token has moved to state $(a', B - 1, r')$ which correctly gives the status of the tape cell left behind by the tape head, as well as of the next transition rule to be fired.

We conclude that the FSAs correctly simulate the Turing Machine. In particular, each time that the global control rule R is satisfied (at least once every $P(AP + 1)^2$ time steps) a new configuration of the Turing Machine is generated. Let T be a large enough time so that if the Turing Machine ever reaches the halting state, it does so earlier than time T. To determine whether this will be so, it suffices to determine the global state of the FSAs at time $P(AP + 1)^2T$. Note that T can be chosen so that $\log T$ is bounded by a polynomial in A, P, and B. PSPACE–completeness of the state prediction-R problem follows. □

4. UNDECIDABILITY OF VALIDITY AND CONSTANT–FREENESS

Recall that a rule R is said to be "constant-free" if and only if it is equivalent to a formula in which no constants appear. In the above, we showed that the state prediction problem is PSPACE–complete for non-constant-free rules and is polynomial time for constant-free rules. In this section we show that it is undecidable if a given rule is constant-free; thus, it is undecidable if a given rule has state prediction problem which is polynomial time computable (if PSPACE and PTIME are distinct). We shall prove this undecidability by showing that the recognition of constant-free rules is equivalent to the recognition of valid rules ("valid" means true for all polls). Then we show that the set of satisfiable rules (true for some poll) is undecidable.

Lemma 5: The problem of recognizing constant-free rules is equivalent to the problem of recognizing valid rules (under many-one polynomial-time reductions).
Proof: By Lemma 3, $R(q_1, \ldots, q_n)$ is constant-free if and only if

$$\exists x_1 \ldots \exists x_n R(x_1, \ldots x_n) \Leftrightarrow \forall x_1 \ldots \forall x_n R(x_1, \ldots x_n)$$

is valid. On the other hand, R is valid if and only if (1) $(\forall x \forall y N(x) = N(y)) \rightarrow R$ is valid, and (2) $R \vee N(q_{k+1}) = N(q_{k+2})$ is constant-free, where q_{k+1} and q_{k+2} are not mentioned in R. Note that (1) is easily seen to be decidable in polynomial time because it is readily reduced to a Boolean combination of inequalities of the form $pN < q$ in the variable N. □

Theorem 3. The set of constant-free rules is undecidable.

Proof: By Lemma 5, it suffices to show that the set of satisfiable rules is undecidable (since a rule is valid iff its negation is not satisfiable). By the Matijasevič-Davis-Putnam-Robinson theorem, it is undecidable if a diophantine equation has a solution. Using the fact that multiplication can be expressed in terms of squaring since $x \cdot y = z$ if and only if $2 \cdot z + x^2 + y^2 = (x + y)^2$ and the fact that squaring can be defined in terms of least common multiple since $x^2 + x = lcm(x, x + 1)$, we have that the satisfiability of purely existential formulas without negations in the language 1, + and $lcm(-,-)$ is undecidable.

Thus it is undecidable if a rule of the form $\exists x_1 \ldots \exists x_N S(\vec{x})$ is satisfiable where S is a conjunction of formulas of the forms $N(x_i) = 1$, $N(x_i) + N(x_j) = N(x_k)$ and

$$N(x_i) = lcm(N(x_j), N(x_k)).$$

If we can replace $N(x_i) = lcm(N(x_j), N(x_k))$ by a formula which is satisfiable if and only if $N(x_i)$ is the least common multiple of $N(x_j)$ and $N(x_k)$, then Theorem 3 will be proved. We first define $DIV(N(x_i), N(x_j))$ to be a formula which is satisfiable if and only if $N(x_i)$ divides $N(x_j)$ by

$$\begin{aligned} DIV(N(x_i), N(x_j)) \Leftrightarrow \exists y \exists z \big[& N(z) - N(y) = N(x_j) \wedge \\ & \forall w \big(N(y) < N(w) \leq N(z) \rightarrow \big\{ (\exists v (N(y) \leq N(v) = N(w) - N(x_i))) \\ & \wedge (N(w) \neq N(z) \rightarrow \exists v (N(w) + N(x_i) = N(v) \leq N(z))) \big\} \big) \big]. \end{aligned}$$

Let $CodesMults(N(x_i), N(y), N(z))$ be the subformula of DIV of the form $\forall w \big(\cdots \big)$; this expresses the fact that the range $N(y)$ to $N(z)$ contains precisely those values $N(w)$ such that $N(w) - N(y)$ is a multiple of $N(x_i)$. Let $NDIV(N(x_i), N(x_j))$ be the following formula, which is satisfiable if and only if $N(x_i)$ does not divide $N(x_j)$:

$$\exists y \exists z \big[N(z) < N(x_j) < N(z) + N(x_i) \wedge CodesMults(N(x_i), N(y), N(z)) \big].$$

Now $LCM(N(x_i), N(x_j), N(x_k))$ can be defined by

$$\begin{aligned} DIV(N(x_j), N(x_i)) \wedge \exists y \exists z \big[& CodesMults(N(x_k), N(y), N(z)) \wedge N(z) = N(y) + N(x_i) \\ & \wedge \forall u (N(y) < N(u) < N(z) \rightarrow NDIV(N(x_k), N(y))) \big]. \end{aligned}$$

By construction, $LCM(N(x_i), N(x_j), N(x_k))$ will be satisfiable in some extension (of any poll) if and only if $N(x_i)$ is the least common multiple of $N(x_j)$ and $N(x_k)$. □

REFERENCES

1. P. Berge, Y. Pomeau, and C. Vidal, *Order Within Chaos*, J. Wiley, New York, 1984.
2. R. L. Devaney, *An Introduction to Chaotic Dynamical Systems*, Benjamin/Cummings, Menlo Park, 1986.
3. P. J. Ramadge, and W. M. Wonham, "Supervisory Control of a Class of Discrete Event Processes," *SIAM J. Control and Optimization*, 25, 1987, pp. 206–230.
4. S. Wolfram (editor), *Theory and Applications of Cellular Automata*, World Scientific, Singapore, 1986.