A MATHEMATICAL FORMALISM FOR AGENT-BASED MODELING

REINHARD LAUBENBACHER, ABDUL S. JARRAH, HENNING MORTVEIT, AND S. S. RAVI

ABSTRACT. Many complex systems can be modeled as multiagent systems in which the constituent entities (agents) interact with each other. The global dynamics of such a system is determined by the nature of the local interactions among the agents. Since it is difficult to formally analyze complex multiagent systems, they are often studied through computer simulations. While computer simulations can be very useful, results obtained through simulations do not formally validate the observed behavior. Thus, there is a need for a mathematical framework which one can use to represent multiagent systems and formally establish their properties. This work contains a brief exposition of some known mathematical frameworks that can model multiagent systems. The focus is on one such framework, namely that of finite dynamical systems. Both, deterministic and stochastic versions of this framework are discussed. The paper contains a sampling of the mathematical results from the literature to show how finite dynamical systems can be used to carry out a rigorous study of the properties of multiagent systems and it is shown how the framework can also serve as a universal model for computation.

CONTENTS

Glossary

Agent-based simulation. An agent-based simulation of a complex system is a computer model that consists of a collection of agents/variables that can take on a typically finite collection of states. The state of an agent at a given point in time is determined through a collection of rules that describe the agent's interaction with other agents. These rules may be deterministic or stochastic. The agent's state depends on the agent's previous state and the state of a collection of other agents with whom it interacts.

Mathematical framework. A mathematical framework for agent-based simulation consists of a collection of mathematical objects that are considered mathematical abstractions of agent-based simulations. This collection of objects should be general enough to capture the key features of most simulations, yet specific enough to allow the development of a mathematical theory with meaningful results and algorithms.

Finite dynamical system. A finite dynamical system is a time-discrete dynamical system on a finite state set. That is, it is a mapping from a cartesian product of finitely many copies of a finite set to itself. This finite set is often considered to be a field. Dynamics is generated by iteration of the mapping.

1. Definition of the subject and its importance

Agent-based simulations are generative or computational approaches used for analyzing "complex systems." What is a "system?" Examples of systems include a collection of molecules in a container, the population in an urban area, and the brokers in a stock market. The entities or *agents* in these three systems would be molecules, individuals and stock brokers, respectively. The agents in such systems interact in the sense that molecules collide, individuals come into contact with other individuals and brokers trade shares. Such systems, often called *multiagent systems*, are not necessarily complex. The label "complex" is typically attached to a system if the number of agents is large, if the agent interactions are involved, or if there is a large degree of heterogeneity in agent character or their interactions.

This is of course not an attempt to define a complex system. Currently there is no generally agreed upon definition of complex systems. It is not the goal of this article to provide such a definition – for our purposes it will be sufficient to think of a complex system as a collection of agents interacting in some manner that involves one or more of the complexity components just mentioned, that is, with a large number of agents, heterogeneity in agent character and interactions and possibly stochastic aspects to all these parts. The global properties of complex systems, such as their global dynamics, emerge from the totality of local interactions between individual agents over time. While these local interactions are well understood in many cases, little is known about the emerging global behavior. Thus, it is typically difficult to construct global mathematical models such as systems of ordinary or partial differential equations, whose properties one could then analyze. Agent-based simulations are one way to create computational models of complex systems that take their place.

An *agent-based simulation*, sometimes also called an *individual-based* or *interaction-based simulation* (which we prefer), of a complex system is in essence a computer program that realizes some (possibly approximate) model of the system to be studied, incorporating the agents and their rules of interaction. The simulation might be deterministic (i.e., the evolution of agent-states is governed by deterministic rules) or stochastic. The typical way in which such simulations are used is to initialize the computer program with a particular assignment of agent states and to run it for some time. The output is a temporal sequence of states for all agents, which is then used to draw conclusions about the complex system one is trying to understand. In other words, the computer

program is the model of the complex system, and by running the program repeatedly, one expects to obtain an understanding of the characteristics of the complex system.

There are two main drawbacks to this approach. First, it is difficult to validate the model. Simulations for most systems involve quite complex software constructs that pose challenges to code validation. Second, there are essentially no rigorous tools available for an analysis of model properties and dynamics. There is also no widely applicable formalism for the comparison of models. For instance, if one agent-based simulation is a simplification of another, then one would like to be able to relate their dynamics in a rigorous fashion. We are currently lacking a mathematically rich formal framework that models agent-based simulations. This framework should have at its core a class of mathematical objects to which one can map agent-based simulations. The objects should have a sufficiently general mathematical structure to capture key features of agent-based simulations and, at the same time, should be rich enough to allow the derivation of substantial mathematical results. This chapter presents one such framework, namely the class of *time-discrete dynamical systems over finite state sets.*

The building blocks of these systems consist of a collection of variables (mapping to agents), a dependency graph that captures the dependence relations of agents on other agents, a local update function for each agent that encapsulates the rules by which the state of each agent evolves over time, and an update discipline for the variables (e.g. parallel or sequential). We will show that this class of mathematical objects is appropriate for the representation of agent-based simulations and, therefore, complex systems, and is rich enough to pose and answer relevant mathematical questions. This class is sufficiently rich to be of mathematical interest in its own right and much work remains to be done in studying it. We also remark that many other frameworks such as probabilistic Boolean networks [80] fit inside the framework described here.

2. Introduction

Computer simulations have become an integral part of today's research and analysis methodologies. The ever-increasing demands arising from the complexity and sheer size of the phenomena studied constantly push computational boundaries, challenge existing computational methodologies, and motivate the development of new theories to improve our understanding of the potential and limitations of computer simulation. Interaction-based simulations are being used to simulate a variety of biological systems such as ecosystems and the immune system, social systems such as urban populations and markets, and infrastructure systems such as communication networks and power grids.

To model or describe a given system, one typically has several choices in the construction and design of agent-based models and representations. When agents are chosen to be simple, the simulation may not capture the behavior of the real system. On the other hand, the use of highly sophisticated agents can quickly lead to complex behavior and dynamics. Also, use of sophisticated agents may lead to a system that *scales poorly.* That is, a linear increase in the number of agents in the system may require a non-linear (e.g., quadratic, cubic, or exponential) increase in the computational resources needed for the simulation.

Two common methods, namely *discrete event simulation* and *time-stepped simulation*, are often used to implement agent-based models [1; 45; 67]. In the discrete event simulation method, each event that occurs in the system is assigned a time of occurrence. The collection of events is kept in increasing order of their occurrence times. (Note that an event occurring at a certain time may give rise to events which occur later.) When all the events that occur at a particular time instant have been carried out, the simulation clock is advanced to the next time instant in the order. Thus, the differences between successive values of the simulation clock may not be uniform. Discrete event simulation is typically used in contexts such as queuing systems [58]. In the time-stepped method of simulation, the simulation clock is always advanced by the same amount. For each value of the simulation clock, the states of the system components are computed using equations that

model the system. This method of simulation is commonly used for studying, e.g., fluid flows or chemical reactions. The choice of model (discrete event versus time-stepped) is typically guided by an analysis of the computational speed they can offer, which in turn depends on the nature of the system being modeled, see, e.g., [37].

Tool-kits for general purpose agent-based simulations include Swarm [29; 57] and Repast [68]. Such tool-kits allow one to specify more complex agents and interactions than would be possible using, e.g., ordinary differential equations models. In general, it is difficult to develop a software package that is capable of supporting the simulation of a wide variety of physical, biological, and social systems.

Standard or classical approaches to modeling are often based on continuous techniques and frameworks such as ordinary differential equations (ODEs) and partial differential equations (PDEs). For example, there are PDE based models for studying traffic flow [38; 47; 85]. These can accurately model the emergence of traffic jams for simple road/intersection configurations through, for example, the formation of shocks. However, these models fail to scale to the size and the specifications required to accurately represent large urban areas. Even if they hypothetically were to scale to the required size, the answers they provide (e.g. car density on a road as a function of position and time) cannot answer questions pertaining to specific travelers or cars. Questions of this type can be naturally described and answered through agent-based models. An example of such a system is TRANSIMS (see Section 2.1.1), where an agent-based simulation scheme is implemented through a cellular automaton model. Another well-known example of the change in modeling paradigms from continuous to discrete is given by lattice gas automata [32] in the context of fluid dynamics.

Stochastic elements are inherent in many systems, and this usually is reflected in the resulting models used to describe them. A stochastic framework is a natural approach in the modeling of, for example, noise over a channel in a simulation of telecommunication networks [8]. In an economic market or a game theoretic setting with competing players, a player may sometimes decide to provide incorrect information. The state of such a player may therefore be viewed and modeled by a random variable. A player may make certain probabilistic assumptions about other players' environment. In biological systems, certain features and properties may only be known up to the level of probability distributions. It is only natural to incorporate this stochasticity into models of such systems.

Since applications of stochastic discrete models are common, it is desirable to obtain a better understanding of these simulations both from an application point of view (reliability, validation) and from a mathematical point of view. However, an important disadvantage of agent-based models is that there are few mathematical tools available at this time for the analysis of their dynamics.

2.1. **Examples of Agent-Based Simulations.** In order to provide the reader with some concrete examples that can also be used later on to illustrate theoretical concepts we describe here three examples of agent-based descriptions of complex systems, ranging from traffic networks to the immune system and voting schemes.

2.1.1. *TRANSIMS (*TR*ansportation* AN*alysis* SIM*ulation* S*ystem).* TRANSIMS is a large-scale computer simulation of traffic on a road network [63; 66; 76]. The simulation works at the resolution level of individual travelers, and has been used to study large US metropolitan areas such as Portland,OR, Washington D.C. and Dallas/Fort Worth. A TRANSIMS-based analysis of an urban area requires (*i*) a population, (*ii*) a location-based activity plan for each individual for the duration of the analysis period and (*iii*) a network representation of all transportation pathways of the given area. The data required for (*i*) and (*ii*) are generated based on, e.g., extensive surveys and other information sources. The network representation is typically very close to a complete description of the real transportation network of the given urban area.

TRANSIMS consists of two main modules: the *router* and the cellular automaton based *micro-simulator*. The router maps each activity plan for each individual (obtained typically from activity

surveys) into a travel route. The micro-simulator executes the travel routes and sends each individual through the transportation network so that its activity plan is carried out. This is done in such a way that all constraints imposed on individuals from traffic driving rules, road signals, fellow travelers, and public transportation schedules are respected. The time scale is typically 1 second.

The micro-simulator is the part of TRANSIMS responsible for the detailed traffic dynamics. Its implementation is based on *cellular automata* which are described in more detail in Section 3.1. Here, for simplicity, we focus on the situation where each individual travels by car. The *road network representation* is in terms of *links* (e.g. road segments) and *nodes* (e.g. intersections). The network description is turned into a cell-network description by discretizing each lane of every link into cells. A cell corresponds to a 7.5 meter lane segment, and can have up to four neighbor cells (front, back, left and right).

The *vehicle dynamics* is specified as follows. Vehicles travel with discrete velocities 0, 1, 2, 3, 4 or 5 which are constant between time steps. Each update time step brings the simulation one time unit forward. If the time unit is one second then the maximum speed of $v_{\max} = 5$ cells per time unit corresponds to an actual speed of $5 \times 7.5\,\mathrm{m/s} = 37.5\,\mathrm{m/s}$ which is $135\,\mathrm{kmh}$ or approximately $83.9\,\mathrm{mph}$.

Ignoring intersection dynamics, the micro-simulator executes three functions for each vehicle in every update: (*a*) lane-changing, (*b*) acceleration and (*c*) movement. These functions can be implemented through four cellular automata, one each for lane change decision and execution, one for acceleration and one for movement. For instance, the acceleration automaton works as follows. A vehicle in TRANSIMS can increase its speed by at most 1 cell per second, but if the road ahead is blocked, the vehicle can come to a complete stop in the same time. The function that is applied to each cell that has a car in it uses the gap ahead and the maximal speed to determine if the car will increase or decrease its velocity. Additionally, a car may have its velocity decreased one unit as determined by a certain deceleration probability. The random deceleration is an important element of producing realistic traffic flow. A major advantage of this representation is that it leads to very light-weight agents, a feature that is critical for achieving efficient scaling.

2.1.2. *CImmSim.* Next we discuss an interaction-based simulation that models certain aspects of the human immune system. Comprised of a large number of interacting cells whose motion is constrained by the body's anatomy, the immune system lends itself very well to agent-based simulation. In particular, these models can take into account three-dimensional anatomical variations as well as small-scale variability in cell distributions. For instance, while the number of T-cells in the human body is astronomical, the number of antigen-specific T-cells, for a specific antigen, can be quite small, thereby creating many spatial inhomogeneities. Also, little is known about the global structure of the system to be modeled.

The first discrete model to incorporate a useful level of complexity was *ImmSim* [22; 23], developed by Seiden and Celada as a stochastic cellular automaton. The system includes B-cells, T-cells, antigen presenting cells (APCs), antibodies, antigens, and antibody-antigen complexes. Receptors on cells are represented by bit strings, and antibodies use bit strings to represent their epitopes and peptides. Specificity and affinity are defined by using bit string similarity. The bit string approach was initially introduced in [31]. The model is implemented on a regular two-dimensional grid, which can be thought of as a slice of a lymph node, for instance. It has been used to study various phenomena, including the optimal number of human leukocyte antigens in human beings [22], the autoimmunity and T-lymphocyte selection in the thymus [60], antibody selection and hypermutation [24], and the dependence of the selection and maturation of the immune response on the antigen-to-receptor's affinity [15]. The computational limitations of the Seiden-Celada model have been overcome by a modified model, *CimmSim* [20], implemented on a parallel architecture. Its complexity is several orders of magnitude larger than that of its predecessor. It has been used to model hypersensitivity to chemotherapy [19] and the selection of escape mutants from immune

recognition during HIV infection [14]. In [21] the *CimmSim* framework was applied to the study of mechanisms that contribute to the persistence of infection with the Epstein-Barr virus.

2.1.3. *A Voting Game.* The following example describes a hypothetical voting scheme. The voting system is constructed from a collection of voters. For simplicity, it is assumed that only two candidates, represented by 0 and 1, contest in the election. There are N voters represented by the set $\{v_1, v_2, \ldots, v_N\}$. Each voter has a candidate preference or a *state*. We denote the state of voter v_i by x_i. Moreover, each voter knows the preferences or states of some of his or her friends (fellow voters). This friendship relation is captured by the *dependency graph* which we describe later in Section 4.1.

Starting from an initial configuration of preferences, the voters cast their votes in some order. The candidate that receives the most votes is the winner. A number of rules can be formulated to decide how each voter chooses a candidate. We will provide examples of such rules later, and as will be seen, the outcome of the election is governed by the order in which the votes are cast as well as the structure of the dependency graph.

3. Existing mathematical frameworks

The field of agent-based simulation currently places heavy emphasis on implementation and computation rather than on the derivation of formal results. Computation is no doubt a very useful way to discover potentially interesting behavior and phenomena. However, unless the simulation has been set up very carefully, its outcome does not formally validate or guarantee the observed phenomenon. It could simply be caused by an artifact of the system model, an implementation error, or some other uncertainty.

A first step in a theory of agent-based simulation is the introduction of a formal framework that on the one hand is precise and computationally powerful, and, on the other hand, is natural in the sense that it can be used to effectively describe large classes of both deterministic and stochastic systems. Apart from providing a common basis and a language for describing the model using a sound formalism, such a framework has many advantages. At a first level, it helps to clarify the key structure in a system. Domain specific knowledge is crucial to deriving good models of complex systems, but domain specificity is often confounded by domain conventions and terminology that eventually obfuscate the real structure.

A formal, context independent framework also makes it easier to take advantage of existing general theory and algorithms. Having a model formulated in such a framework also makes it easier to establish results. Additionally, expressing the model using a general framework is more likely to produce results that are widely applicable. This type of framework also supports implementation and validation. Modeling languages like UML [16] serve a similar purpose, but tend to focus solely on software implementation and validation issues, and very little on mathematical or computational analysis.

3.1. **Cellular Automata.** In this section we discuss several existing frameworks for describing agent-based simulations. Cellular automata (CA) were introduced by Ulam and von Neumann [84] as biologically motivated models of computation. Early research addressed questions about the computational power of these devices. Since then their properties have been studied in the context of dynamical systems [40], language theory [52], and ergodic theory [51] to mention just a few areas. Cellular automata were popularized by Conway [35] (Game of Life) and by Wolfram [55; 86; 88]. Cellular automata (both deterministic and stochastic) have been used as models for phenomena ranging from lattice gases [32] and flows in porous media [77] to traffic analysis [33; 64; 65].

A cellular automaton is typically defined over a regular grid. An example is a two-dimensional grid such as $\mathbf{Z}^2$. Each grid point (i, j) is referred to as a *site* or *node*. Each site has a state $x_{i,j}(t)$ which is often taken to be binary. Here t denotes the time step. Furthermore, there is a notion of a *neighborhood* for each site. The neighborhood N of a site is the collection of sites that can

influence the future state of the given site. Based on its current state $x_{i,j}(t)$ and the current states of the sites in its neighborhood N, a function $f_{i,j}$ is used to compute the next state $x_{i,j}(t+1)$ of the site (i,j). Specifically, we have

(3.1)
$$x_{i,j}(t+1) = f_{i,j}(\bar{x}_{i,j}(t)) \,,$$

where $\bar{x}_{i,j}(t)$ denotes the tuple consisting of all the states $x_{i',j'}(t)$ with $(i',j') \in N$. The tuple consisting of the states of all the sites is the CA *configuration* and is denoted $x(t) = (x_{i,j}(t))_{i,j}$. Equation (3.1) is used to map the configuration $x(t)$ to $x(t+1)$. The cellular automaton map or dynamical system, is the map Φ that sends $x(t)$ to $x(t+1)$.

A central part of CA research is to understand how configurations evolve under iteration of the map Φ and what types of dynamical behavior can be generated. A general introduction to CA can be found in [43].

3.2. **Hopfield Networks.** Hopfield networks were proposed as a simple model of associative memories [42]. A discrete Hopfield neural network consists of an undirected graph $Y(V, E)$. At any time t, each node $v_i \in V$ has a state $x_i(t) \in \{+1, -1\}$. Further, each node $v_i \in V$ has an associated *threshold* $\tau_i \in \mathbf{R}$. Each edge $\{v_i, v_j\} \in E$ has an associated weight $w_{i,j} \in \mathbf{R}$. For each node v_i, the neighborhood N_i of v_i includes v_i and the set of nodes that are adjacent to v_i in Y. Formally,

$$N_i = \{v_i\} \cup \{v_j \in V \ : \ \{v_i, v_j\} \in E\}.$$

States of nodes are updated as follows. At time t, node v_i computes the function f_i defined by

$$f_i(t) = \mathrm{sgn}\left(-\tau_i + \sum_{v_j \in N_i} w_{i,j}\, x_j(t)\right),$$

where sgn is the map from $\mathbf{R}$ to $\{+1, -1\}$, defined by

$$\mathrm{sgn}(x) = \begin{cases} 1, & \text{if } x \geq 0 \text{ and} \\ -1 & \text{otherwise.} \end{cases}$$

Now, the state of v_i at time $t+1$ is

$$x_i(t+1) = f_i(t).$$

Many references on Hopfield networks (see for example [42; 78]) assume that the underlying undirected graph is complete; that is, there is an edge between every pair of nodes. In the definition presented above, the graph need not be complete. However, this does not cause any difficulties since the missing edges can be assigned weight 0. As a consequence, such edges will not play any role in determining the dynamics of the system. Both synchronous and asynchronous update models of Hopfield neural networks have been considered in the literature. For theoretical results concerning Hopfield networks see [69; 70] and the references cited therein. Reference [78] presents a number of applications of neural networks. In [54], a Hopfield model is used to study polarization in dynamic networks.

3.3. **Communicating Finite State Machines.** The model of communicating finite state machines (CFSM) was proposed to analyze protocols used in computer networks. In some of the literature, this model is also referred to as "concurrent transition systems" [36].

In the CFSM model, each agent is a process executing on some node of a distributed computing system. Although there are minor differences among the various CFSM models proposed in the literature [17; 36], the basic set-up models each process as a finite state machine (FSM). Thus, each agent is in a certain state at any time instant t. For each pair of agents, there is a bidirectional channel through which they can communicate. The state of an agent at time $t+1$ is a function of the current state and the input (if any) on one or more of the channels. When an agent (FSM) undergoes a transition from one state to another, it may also choose to send a message to another agent or receive a message from an agent. In general, such systems can be synchronous or asynchronous.

As can be seen, CFSMs are a natural formalism for studying protocols used in computer networks. The CFSM model has been used extensively to prove properties (e.g. deadlock freedom, fairness) of a number of protocols used in practice (see [17; 36] and the references cited therein).

Other frameworks include interacting particle systems [50], and Petri nets [59]. There is a vast literature on both, but space limitations prevent a discussion here.

4. Finite Dynamical Systems

Another, quite general, modeling framework that has been proposed is that of *finite dynamical systems*, both synchronous and asynchronous. Here the proposed mathematical object representing an agent-based simulation is a time-discrete dynamical system on a finite state set. The description of the systems is modeled after the key components of an agent-based simulation, namely agents, the dependency graph, local update functions, and an update order. This makes a mapping to agent-based simulations natural. In the remainder of this chapter we will show that finite dynamical systems satisfy our criteria for a good mathematical framework in that they are general enough to serve as a broad computing tool and mathematically rich enough to allow the derivation of formal results.

4.1. **Definitions, Background, and Examples.** Let $x_1, \ldots, x_n$ be a collection of variables, which take values in a finite set X. (As will be seen, the variables represent the entities in the system being modeled and the elements of X represent their states.) Each variable x_i has associated to it a "local update function" $f_i : X^n \longrightarrow X$, where "local" refers to the fact that f_i takes inputs from the variables in the "neighborhood" of x_i, in a sense to be made precise below. By abuse of notation we also let f_i denote the function $X^n \longrightarrow X^n$ which changes the i-th coordinate and leaves the other coordinates unchanged. This allows for the sequential composition of the local update functions. These functions assemble to a dynamical system

$$\Phi = (f_1, \ldots, f_n) : X^n \longrightarrow X^n,$$

with the dynamics generated by iteration of Φ. As an example, if $X = \{0, 1\}$ with the standard Boolean operators AND and OR, then Φ is a Boolean network.

The assembly of Φ from the local functions f_i can be done in one of several ways. One can update each of the variables simultaneously, that is,

$$\Phi(x_1, \ldots, x_n) = (f_1(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n)) .$$

In this case one obtains a *parallel dynamical system*.

Alternatively, one can choose to update the states of the variables according to some fixed update order, for example, a permutation or a word on the set $\{1, \ldots, n\}$. That is, let $\pi = (\pi_1, \ldots, \pi_t)$ be a word using the alphabet $\{1, \ldots, n\}$. The function

$$(4.1) \qquad\qquad\qquad \Phi_\pi = f_{\pi_t} \circ f_{\pi_{n-1}} \circ \cdots \circ f_{\pi_1} ,$$

is called a *sequential dynamical system (SDS)* and, as before, the dynamics of Φ_π is generated by iteration. The case when π is a permutation on $\{1, \ldots, n\}$ has been studied extensively [9; 10; 11; 12]. It is clear that using a different permutation γ may result in a different dynamical system Φ_γ.

Remark 4.1. Notice that for a fixed π, the function Φ_π is a parallel dynamical system: once the update order π is chosen and the local update functions are composed according to π, that is, the function Φ_π has been computed, then $\Phi_\pi(x_1, \ldots, x_n) = g(x_1, \ldots, x_n)$ where g is a parallel update dynamical system. However, the maps g_i are not local functions.

The dynamics of Φ is usually represented as a directed graph on the vertex set X^n, called the *phase space* of Φ. There is a directed edge from $\mathbf{v} \in X^n$ to $\mathbf{w} \in X^n$ if and only if $\Phi(\mathbf{v}) = \mathbf{w}$. A second graph that is usually associated with a finite dynamical system is its *dependency graph* $Y(V, E)$. In general, this is a directed graph, and its vertex set is $V = \{1, \ldots, n\}$. There is a

directed edge from i to j if and only if x_i appears in the function f_j. In many situations, the interaction relationship between pairs of variables is symmetric; that is, variable x_i appears in f_j if and only if x_j appears in f_i. In such cases, the dependency graph can be thought of as an undirected graph. We recall that the dependency graphs mentioned in the context of the voting game (Section 2.1.3) and Hopfield networks (Section 3.2) are undirected graphs. The dependency graph plays an important role in the study of finite dynamical systems and is sometimes listed explicitly as part of the specification of Φ.

Example 4.2. Let $X = \{0, 1\}$. Suppose we have four variables and the local Boolean update functions are

$$
\begin{aligned}
f_1 &= x_1 + x_2 + x_3 + x_4, \\
f_2 &= x_1 + x_2, \\
f_3 &= x_1 + x_3, \\
f_4 &= x_1 + x_4,
\end{aligned}
$$

where "$+$" represents XOR, the exclusive OR function. The dynamics of the function $\Phi = (f_1, \ldots, f_4) : X^4 \longrightarrow X^4$ is the directed graph on the left in Figure 1 while the dependency graph is on the right.
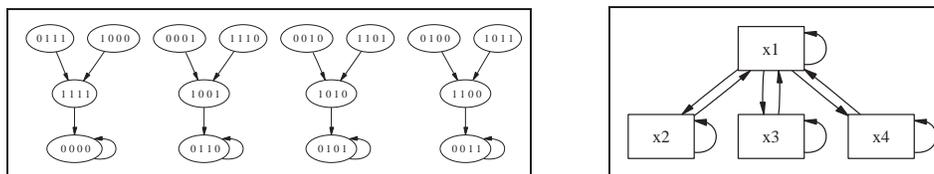

FIGURE 1. The phase space of the parallel system Φ is on the left and dependency graph of the Boolean function from Example 4.2 is on the right.

Example 4.3. Consider the local functions in the Example 4.2 above and let $\pi = (2, 1, 3, 4)$. Then

$$
\Phi_\pi = f_4 \circ f_3 \circ f_1 \circ f_2 : X^4 \longrightarrow X^4.
$$

The phase space of Φ_π is the directed graph on the left in Figure 2, while the phase space of Φ_γ, where $\gamma = id$ is on the right in Figure 2.
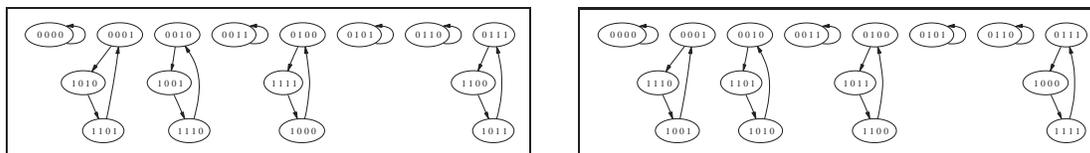

FIGURE 2. The phase spaces from Example 4.3: Φ_π (left) and Φ_{id} (right)

Notice that the phase space of any function is a directed graph in which every vertex has out-degree one; this is a characteristic property of deterministic functions.

Making use of Boolean arithmetic is a powerful tool in studying Boolean networks, which is not available in general. In order to have available an enhanced set of tools it is often natural to make an additional assumption regarding X, namely that it is a finite number system, a *finite field* [49]. This amounts to the assumption that there are "addition" and "multiplication" operations defined

on X that satisfy the same rules as ordinary addition and multiplication of numbers. Examples include $\mathbf{Z}_p$, the integers modulo a prime p. This assumption can be thought of as the discrete analog of imposing a coordinate system on an affine space.

When the set X is a finite field, it is easy to show that for any local function g, there exists a polynomial h such that $g(x_1, \ldots, x_n) = h(x_1, \ldots, x_n)$ for all $(x_1, \ldots, x_n) \in X^n$. To be precise, suppose X is a finite field with q elements. Then

$$(4.2) \qquad g(x_1, \ldots, x_n) = \sum_{(c_1, \ldots, c_n) \in X^n} g(c_1, \ldots, c_n) \prod_{i=1}^{n} (1 - (x_i - c_i)^{q-1}).$$

This observation has many useful consequences, since polynomial functions have been studied extensively and many analytical tools are available.

Notice that cellular automata and Boolean networks, both parallel and sequential, are special classes of polynomial dynamical systems. In fact, it is straightforward to see that

$$(4.3) \qquad x \wedge y = x \cdot y,\ x \vee y = x + y + xy \text{ and } \neg x = x + 1.$$

Therefore, the modeling framework of finite dynamical systems includes that of cellular automata, discussed earlier. Also, since a Hopfield network is a function $X^n \longrightarrow X^n$, which can be represented through its local constituent functions, it follows that Hopfield networks also are special cases of finite dynamical systems.

4.2. **Stochastic Finite Dynamical Systems.** The deterministic framework of finite dynamical systems can be made stochastic in several different ways, making one or more of the system's defining data stochastic. For example, one could use one or both of the following criteria.

- Assume that each variable has a nonempty set of local functions assigned to it, together with a probability distribution on this set, and each time a variable is updated, one of these local functions is chosen at random to update its state. We call such systems *probabilistic finite dynamical systems* (PFDS), a generalization of probabilistic Boolean networks [81].
- Fix a subset of permutations $T \subseteq S_n$ together with a probability distribution. When it is time for the system to update its state, a permutation $\pi \in T$ is chosen at random and the agents are updated sequentially using π. We call such systems *stochastic finite dynamical systems* (SFDS).

Remark 4.4. By Remark 4.1, each system Φ_π is a parallel system. Hence a SFDS is nothing but a set of parallel dynamical systems $\{\Phi_\pi : \pi \in T\}$, together with a probability distribution. When it is time for the system to update its state, a system Φ_π is chosen at random and used for the next iteration.

To describe the phase space of a stochastic finite dynamical system, a general method is as follows. Let Ω be a finite collection of systems $\Phi_1, \ldots, \Phi_t$, where $\Phi_i : X^n \longrightarrow X^n$ for all i, and consider the probabilities $p_1, \ldots, p_t$ which sum to 1. We obtain the stochastic phase space

$$(4.4) \qquad \Gamma_\Omega = p_1 \Gamma_1 + p_2 \Gamma_2 + \cdots + p_t \Gamma_t,$$

where Γ_i is the phase space of Φ_i. The associated probability space is $\mathcal{F} = (\Omega, 2^\Omega, \mu)$, where the probability measure μ is induced by the probabilities p_i. It is clear that the stochastic phase space can be viewed as a *Markov chain* over the state space X^n. The adjacency matrix of Γ_Ω directly encodes the Markov transition matrix. This is of course not new, and has been done in, e.g.,[28; 81; 83]. But we emphasize the point that even though SFDS give rise to Markov chains *our study of SFDS is greatly facilitated by the rich additional structure* available in these systems. To understand the effect of structural components such as the topology of the dependency graph or the stochastic nature of the update, it is important to study them not as Markov chains but as SFDS.

Example 4.5. Consider Φ_π and Φ_γ from Example 4.3 and let Γ_π and Γ_γ be their phases spaces as shown in Figure 2. Let $p_1 = p_2 = \dfrac{1}{2}$. The phase space $\dfrac{1}{2}\Gamma_\pi + \dfrac{1}{2}\Gamma_\gamma$ of the stochastic sequential dynamical system obtained from Φ_π and Φ_γ (with equal probabilities) is presented in Figure 3.
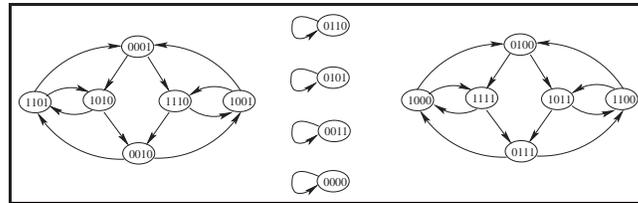

FIGURE 3. The stochastic phase space for Example 4.5 induced by the two deterministic phase spaces of Φ_π and Φ_γ from Figure 2. For simplicity the weights of the edges have been omitted.

4.3. **Agent-based Simulations as Finite Dynamical Systems.** In the following we describe the generic structure of the systems typically modeled and studied through agent-based simulations. The central notion is naturally that of an *agent*.

Each agent carries a state that may encode its preferences, internal configuration, perception of its environment, and so on. In the case of TRANSIMS, for instance, the agents are the cells making up the road network. The cell state contains information about whether or not the cell is occupied by a vehicle as well as the velocity of the vehicle. One may assume that each cell takes on states from the same set of possible states, which may be chosen to support the structure of a finite field.

The agents interact with each other, but typically an agent only interacts with a small subset of agents, its *neighbors*. Through such an interaction an agent may decide to change its state based on the states (or other aspects) of the agents with which it interacts. We will refer to the process where an agent modifies its state through interaction as an *agent update*. The precise way in which an agent modifies its state is governed by the nature of the particular agent. In TRANSIMS the neighbors of a cell are the adjacent road network cells. From this adjacency relation one obtains a dependency graph of the agents. The local update function for a given agent can be obtained from the rules governing traffic flow between cells.

The updates of all the agents may be scheduled in different ways. Classical approaches include synchronous, asynchronous and event-driven schemes. The choice will depend on system properties or particular considerations about the simulation implementation.

In the case of *CImmSim*, the situation is somewhat more complicated. Here the agents are also the spatial units of the system, each representing a small volume of lymph tissue. The total volume is represented as a 2-dimensional CA, in which every agent has 4 neighbors, so that the dependency graph is a regular 2-dimensional grid. The state of each agent is a collection of counts for the various immune cells and pathogens that are present in this particular agent (volume). Movement between cells is implemented as diffusion. Immune cells can interact with each other and with pathogens while they reside in the same volume. Thus, the local update function for a given cell of the simulation is made up of the two components of movement between cells and interactions within a cell. For instance, a B cell could interact with the Epstein-Barr virus in a given volume and transition from uninfected to infected by the next time step. Interactions as well as movement are stochastic, resulting in a stochastic finite dynamical system. The update order is parallel.

Example 4.6 (The voting game (Section 2.1.3)). The following scenario is constructed to illustrate how implementation choices for the system components have a clear and direct bearing on the dynamics and simulation outcomes.

Let the voter dependency graph be the star graph on 5 vertices with center vertex a and surrounding vertices b, c, d and e. Furthermore, assume that everybody votes opportunistically using the majority rule: the vote cast by an individual is equal to the preference of the majority of his/her friends with the person's own preference included. For simplicity, assume candidate 1 is preferred in the case of a tie.

If the initial preference is $x_a = 1$ and $x_b = x_c = x_d = x_e = 0$ then if voter a goes first he/she will vote for candidate 0 since that is the choice of the majority of the neighbor voters. However, if b and c go first they only know a's preference. Voter b therefore casts his/her vote for candidate 1 as does c. Note that this is a tie situation with an equal number of preferences for candidate 1 (a) and for candidate 2 (b). If voter a goes next, then the situation has changed: the preference of b and c has already changed to 1. Consequently, voter a picks candidate 1. At the end of the day candidate 1 is the election winner, and the choice of update order has tipped the election!

This example is of course constructed to illustrate our point. However, in real systems it can be much more difficult to detect the presence of such sensitivities and their implications. A solid mathematical framework can be very helpful in detecting such effects.

5. Finite dynamical systems as theoretical and computational tools

If finite dynamical systems are to be useful as a modeling paradigm for agent-based simulations it is necessary that they can serve as a fairly universal model of computation. We discuss here how such dynamical systems can mimic Turing Machines (TMs), a standard universal model for computation. For a more thorough exposition, we refer the reader to the series of papers by Barrett et al. [2; 3; 4; 5; 6]. To make the discussion reasonably self-contained, we provide a brief and informal discussion of the TM model. Additional information on TMs can be found in any standard text on the theory of computation (e.g. [82]).

5.1. **A Computational View of Finite Dynamical Systems: Definitions.** In order to understand the relationship of finite dynamical systems to TMs, it is important to view such systems from a computational stand point. Recall that a finite dynamical system $\Phi : X^n \longrightarrow X^n$, where X is a finite set, has an underlying dependency graph $Y(V, E)$. From a computational point of view, the nodes of the dependency graph (the agents in the simulation) are thought of as devices that compute appropriate functions. For simplicity, we will assume in this section that the dependency graph is undirected, that is, all dependency relations are symmetric. At any time, the state value of each node $v_i \in V$ is from the specified domain X. The inputs to f_i are the current state of v_i and the states of the neighbors of v_i as specified by Y. The output of f_i, which is also a member of X, becomes the state of v_i at the next time instant. The discussion in this section will focus on sequentially updated systems (SDS), but all results discussed apply to parallel systems as well.

Each step of the computation carried out by an SDS can be thought as consisting of n "mini steps;" in each mini step, the value of the local transition function at a node is computed and the state of the node is changed to the computed value. Given an SDS Φ, a *configuration* $\mathcal{C}$ of Φ is a vector $(c_1, c_2, \ldots, c_n) \in X^n$. It can be seen that each computational step of an SDS causes a transition from one configuration to another.

5.2. **Configuration Reachability Problem for SDSs.** Based on the computational view, a number of different problems can be defined for SDSs (see for example, [4; 6; 7]). To illustrate how SDSs can model TM computations, we will focus on one such problem, namely the *configuration reachability* (CR) problem: Given an SDS Φ, an initial configuration $\mathcal{C}$ and another configuration $\mathcal{C}'$, will Φ, starting from $\mathcal{C}$, ever reach configuration $\mathcal{C}'$? The problem can also be expressed in terms of the phase space of Φ. Since configurations such as $\mathcal{C}$ and $\mathcal{C}'$ are represented by nodes in the phase space, the CR problem boils down to the question of whether there is a directed path in the phase space from $\mathcal{C}$ to $\mathcal{C}'$. This abstract problem can be mapped to several problems in the simulation of multiagent systems. Consider for example the TRANSIMS context. Here, the initial

configuration $\mathcal{C}$ may represent the state of the system early in the day (when the traffic is very light) and $\mathcal{C}'$ may represent an "undesirable" state of the system (such as heavy traffic congestion). Similarly, in the context of modeling an infectious disease, $\mathcal{C}$ may represent the initial onset of the disease (when only a small number of people are infected) and $\mathcal{C}'$ may represent a situation where a large percentage of the population is infected. The purpose of studying computational problems such as CR is to determine whether one can efficiently predict the occurrence of certain events in the system from a description of the system. If computational analysis shows that the system can indeed reach undesirable configurations as it evolves, then one can try to identify steps needed to deal with such situations.

5.3. **Turing Machines: A Brief Overview.** A Turing machine (TM) is a simple and commonly used model for general purpose computational devices. Since our goal is to point out how SDSs can also serve as computational devices, we will present an informal overview of the TM model. Readers interested in a more formal description may consult [82].

A TM consists of a set Q of states, a one-way infinite input tape and a read/write head that can read and modify symbols on the input tape. The input tape is divided into cells and each cell contains a symbol from a special finite alphabet. An input consisting of n symbols is written on the leftmost n cells of the input tape. (The other cells are assumed to contain a special symbol called *blank*.) One of the states in Q, denoted by q_s, is the designated *start* state. Q also includes two other special states, denoted by q_a (the *accepting* state) and q_r (the *rejecting* state). At any time, the machine is in one of the states in Q. The *transition function* for the TM specifies for each combination of the current state and the current symbol under the head, a new state, a new symbol for the current cell (which is under the head) and a movement (i.e., left or right by one cell) for the head. The machine starts in state q_s with the head on the first cell of the input tape. Each step of the machine is carried out in accordance with the transition function. If the machine ever reaches either the accepting or the rejecting state, it halts with the corresponding decision; otherwise, the machine runs forever.

A *configuration* of a TM consists of its current state, the current tape contents and the position of the head. Note that the transition function of a TM specifies how a new configuration is obtained from the current configuration.

The above description is for the basic TM model (also called *single tape* TM model). For convenience in describing some computations, several variants of the above basic model have been proposed. For example, in a *multi-tape* TM, there are one or more *work tapes* in addition to the input tape. The work tapes can be used to store intermediate results. Each work tape has its own read/write head and the definitions of configuration and transition function can be suitably modified to accommodate work tapes. While such an enhancement to the basic TM model makes it easier to carry out certain computations, it does not add to the machine's computational power. In other words, any computation that can be carried out using the enhanced model can also be carried out using the basic model.

As in the case of dynamical systems, one can define a *configuration reachability* (CR) problem for TMs: Given a TM M, an initial configuration $\mathcal{I}_M$ and another configuration $\mathcal{C}_M$, will the TM starting from $\mathcal{I}_M$ ever reach $\mathcal{C}_M$? We refer to the CR problem in the context of TMs as CR-TM. In fact, it is this problem for TMs that captures the essence of what can be effectively computed. In particular, by choosing the state component of $\mathcal{C}_M$ to be one of the halting states (q_a or q_r), the problem of determining whether a function is computable is transformed into an appropriate CR-TM problem. By imposing appropriate restrictions on the resources used by a TM (e.g. the number of steps, the number of cells on the work tapes), one obtains different versions of the CR-TM problem which characterize different computational complexity classes [82].

5.4. **How SDSs Can Mimic Turing Machines.** The above discussion points out an important similarity between SDSs and TMs. Under both of these models, each computational step causes

a transition from one configuration to another. It is this similarity that allows one to construct a discrete dynamical system Φ that can simulate a TM. Typically, each step of a TM is simulated by a short sequence of successive iterations Φ. As part of the construction, one also identifies a suitable mapping between the configurations of the TM being simulated and those of the dynamical system. This is done in such a way that the answer to CR-TM problem is "yes" if and only if the answer to the CR problem for the dynamical system is also "yes."

To illustrate the basic ideas, we will informally sketch a construction from [4]. This construction produces an SDS Φ that simulates a restricted version of TMs; the restriction being that for any input containing n symbols, the number of work tape cells that the machine may use is bounded by a linear function of n. Such a TM is called a *linear bounded automaton* (LBA) [82]. Let M denote the given LBA and let n denote the length of the input to M. The domain X for the SDS Φ is chosen to be a finite set based on the allowed symbols in the input to the TM. The dependency graph is chosen to be a simple path on n nodes, where each node serves as a representative for a cell on the input tape. The initial and final configurations $\mathcal{C}$ and $\mathcal{C}'$ for Φ are constructed from the corresponding configurations of M. The local transition function for each node of the SDS is constructed from the given transition function for M in such a way that each step of M corresponds to exactly one step of Φ. Thus, there is a simple bijection between the sequence of configurations that M goes through during its computation and the sequence of states that Φ goes through as it evolves. Using this bijection, it is shown in [4] that the answer to the CR-TM problem is "yes" if and only if Φ reaches $\mathcal{C}'$ starting from $\mathcal{C}$. Reference [4] also presents a number of sophisticated constructions where the resulting dynamical system is very simple; for example, it is shown that an LBA can be simulated by an SDS in which X is the Boolean field, the dependency graph is d-regular for some constant d and the local transition functions at all the nodes are *identical*. Such results point out that one does not need complicated dynamical systems to model TM computations.

Barrett et al. [5] have also considered *stochastic* SDS (SSDS), where the local transition function at each node is stochastic. For each combination of inputs, a stochastic local transition function specifies a probability distribution over the domain of state values. It is shown in [5] that SSDSs can effectively simulate computations carried out by probabilistic TMs (i.e., TMs whose transition functions are stochastic).

5.5. **TRANSIMS related questions.** Section 2.1.1 gave an overview of some aspects of the TRANSIMS model. The micro-simulator module is specified as a functional composition of four cellular automata of the form $\Delta_4 \circ \Delta_3 \circ \Delta_2 \circ \Delta_1$. (We only described Δ_3 which corresponds to velocity updates.) Such a formulation has several advantages. First, it has created an abstraction of the essence of the system in a precise mathematical way without burying it in contextual domain details. An immediate advantage of this specification is that it makes the whole implementation process more straightforward and transparent. While the local update functions for the cells are typically quite simple, for any realistic study of an urban area the problem size would typically require a sequential implementation, raising a number of issues that are best addressed within a mathematical framework like the one considered here.

6. Mathematical results on finite dynamical systems

In this section we outline a collection of mathematical results about finite dynamical systems that is representative of the available knowledge. The majority of these results are about deterministic systems, as the theory of stochastic systems of this type is still in its infancy. We will first consider synchronously updated systems.

Throughout this section, we make the assumption that the state set X carries the algebraic structure of a finite field. Accordingly, we use the notation k instead of X. It is a standard result that in this case the number q of elements in k has the form $q = p^t$ for some prime p and $t \geq 1$. The

reader may keep the simplest case $k = \{0, 1\}$ in mind, in which case we are considering Boolean networks.

Recall Equation (4.2). That is, any function $g : k^n \longrightarrow k$ can be represented by a multivariate polynomial with coefficients in k. If we require that the exponent of each variable be less than q, then this representation is unique. In particular Equation (4.3) implies that every Boolean function can be represented uniquely as a polynomial function.

6.1. **Parallel update systems.** Certain classes of finite dynamical systems have been studied extensively, in particular cellular automata and Boolean networks. Many of these studies have been experimental in nature, however. Some more general mathematical results about cellular automata can be found in the papers of Wolfram and collaborators [87]. The results there focus primarily on one-dimensional Boolean cellular automata with a particular fixed initial state. Here we collect a sampling of more general results.

We first consider linear and affine systems

$$\Phi = (f_1, \ldots, f_n) : k^n \longrightarrow k^n$$

That is, we consider systems for which the coordinate functions f_i are linear, resp. affine, polynomials. (In the Boolean case this includes functions constructed from XOR and negation.) When each f_i is a linear polynomial of the form $f_i(x_1, \ldots, x_n) = a_{i1}x_1 + \cdots + a_{in}x_n$, the map Φ is nothing but a *linear transformation* on k^n over k, and, by using the standard basis, Φ has the matrix representation

$$\Phi \left(\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \right) = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix},$$

where $a_{ij} \in k$ for all i, j.

Linear finite dynamical systems were first studied by Elspas [30]. His motivation came from studying feedback shift register networks and their applications to radar and communication systems and automatic error correction circuits. For linear systems over finite fields of prime cardinality, that is, q is a prime number, Elspas showed that the exact number and length of each limit cycle can be determined from the *elementary divisors* of the matrix A. Recently, Hernandez [41] re-discovered Elspas' results and generalized them to arbitrary finite fields. Furthermore, he showed that the structure of the *tree of transients* at each node of each limit cycle is the same, and can be completely determined from the nilpotent elementary divisors of the form x^a. For *affine* Boolean networks (that is, finite dynamical systems over the Boolean field with two elements, whose local functions are linear polynomials which might have constant terms), a method to analyze their cycle length has been developed in [56]. After embedding the matrix of the transition function, which is of dimension $n \times (n + 1)$, into a square matrix of dimension $n + 1$, the problem is then reduced to the linear case. A fast algorithm based on [41] has been implemented in [44], using the symbolic computation package *Macaulay2*.

It is not surprising that the phase space structure of Φ should depend on invariants of the matrix $A = (a_{ij})$. The rational canonical form of A is a block-diagonal matrix and one can recover the structure of the phase space of A from that of the blocks in the rational form of A. Each block represents either an invertible or a nilpotent linear transformation. Consider an invertible block B. If $\mu(x)$ is the minimal polynomial of B, then there exists s such that $\mu(x)$ divides $x^s - 1$. Hence $B^s - I = 0$ which implies that $B^s \mathbf{v} = \mathbf{v}$. That is, every state vector $\mathbf{v}$ in the phase space of B is in a cycle whose length is a divisor of s.

Definition 6.1. For any polynomial $\lambda(x)$ in $k[x]$, the *order* of $\lambda(x)$ is the least integer s such that $\lambda(x)$ divides $x^s - 1$.

The cycle structure of the phase space of Φ can be completely determined from the orders of the irreducible factors of the minimal polynomial of Φ. The computation of these orders involves in particular the factorization of numbers of the form $q^r - 1$, which makes the computation of the order of a polynomial potentially quite costly. The nilpotent blocks in the decomposition of A determine the tree structure at the nodes of the limit cycles. It turns out that all trees at all periodic nodes are identical. This generalizes a result in [55] for additive cellular automata over the field with two elements.

While the fact that the structure of the phase space of a linear system can be determined from the invariants associated with its matrix may not be unexpected, it is a beautiful example of how the right mathematical viewpoint provides powerful tools to completely solve the problem of relating the structure of the local functions to the resulting (or emerging) dynamics. Linear and affine systems have been studied extensively in several different contexts and from several different points of view, in particular the case of cellular automata. For instance, additive cellular automata over more general rings as state sets have been studied, e.g., in [25]. Further results on additive CAs can also be found there. One important focus in [25] is on the problem of finding CAs with limit cycles of maximal length for the purpose of constructing pseudo random number generators.

Unfortunately, the situation is more complicated for nonlinear systems. For the special class of Boolean synchronous systems whose local update functions consist of monomials, there is a polynomial time algorithm that determines whether a given monomial system has only fixed points as periodic points [27]. This question was motivated by applications to the modeling of biochemical networks. The criterion is given in terms of invariants of the dependency graph Y. For a strongly connected directed graph Y (i.e., there is a directed path between any pair of vertices), its *loop number* is the greatest common divisor of all directed loops at a particular vertex. (This number is independent of the vertex chosen.)

Theorem 6.2 ([27])**.** *A Boolean monomial system has only fixed points as periodic points if and only if the loop number of every strongly connected component of its dependency graph is equal to 1.*

In [26] it is shown that the problem for general finite fields can be reduced to that of a Boolean system and a linear system over rings of the form $\mathbf{Z}/p^r\mathbf{Z}$, p prime. Boolean monomial systems have been studied before in the cellular automaton context [13].

6.2. **Sequential update systems.** The update order in a sequential dynamical system has been studied using combinatorial and algebraic techniques. A natural question to ask here is how the system depends on the update schedule. In [9; 10; 61; 72] this was answered on several levels for the special case where the update schedule is a permutation. We describe these results in some detail. Results about the more general case of update orders described by words on the indices of the local update functions can be found in [34].

Given local update functions $f_i : k^n \longrightarrow k$ and permutation update orders σ, π, a natural question is when the two resulting SDS Φ_σ and Φ_π are identical and, more generally, how many different systems one obtains by varying the update order over all permutations. Both questions can be answered in great generality. The answer involves invariants of two graphs, namely the acyclic orientations of the dependency graph Y of the local update functions and the *update graph* of Y. The update graph $U(Y)$ of Y is the graph whose vertex set consists of all permutations of the vertex set of Y [72]. There is an (undirected) edge between two permutations $\sigma = (\sigma_1, \ldots, \sigma_n)$ and $\tau = (\tau_1, \ldots, \tau_n)$ if they differ by a transposition of two adjacent entries σ_i and σ_{i+1} such that there is no edge in Y between σ_i and σ_{i+1}.

The update graph encodes the fact that one can commute two local update functions f_i and f_j without affecting the end result Φ if i and j are not connected by an edge in Y. That is, $\cdots f_i \circ f_j \cdots = \cdots f_j \circ f_i \cdots$ if and only if i and j are not connected by an edge in Y.

All permutations belonging to the same connected component in $U(Y)$ give identical SDS maps. The number of (connected) components in $U(Y)$ is therefore an upper bound for the number of functionally inequivalent SDS that can be generated by just changing the update order. This can also be characterized in terms of acyclic orientations of the graph Y: each component in the update graph induces a unique acyclic orientation of the graph Y. Moreover, we have the following result:

Proposition 6.3 ([72]). *There is a bijection*

$$F_Y \colon S_Y / \sim_Y \longrightarrow \mathrm{Acyc}(Y) \,,$$

where $S_Y / \sim_Y$ denotes the set of connected components of $U(Y)$ and $\mathrm{Acyc}(Y)$ denotes the set of acyclic orientations of Y.

This upper bound on the number of functionally different systems has been shown in [72] to be sharp for Boolean systems, in the sense that for a given Y one construct this number of different systems, using appropriate combinations of NOR functions.

For two permutations σ and τ it is easy to determine if they give identical SDS maps: one can just compare their induced acyclic orientations. The number of acyclic orientations of the graph Y tells how many functionally different SDS maps one can obtain for a fixed graph and fixed vertex functions. The work of Cartier and Foata [18] on partially commutative monoids studies a similar question, but their work is not concerned with finite dynamical systems.

Note that permutation update orders have been studied sporadically in the context of cellular automata on circle graphs [71] but not in a systematic way, typically using the order $(1, 2, \ldots, n)$ or the even-odd/odd-even orders. As a side note, we remark that this work also confirms our findings that switching from a parallel update order to a sequential order turns the complex behavior found in Wolfram's "class III and IV" automata into much more regular or mundane dynamics, see e.g. [79].

The work on functional equivalence was extended to dynamical equivalence (topological conjugation) in [10; 61]. The automorphism group of the graph Y can be made to act on the components of the update graph $U(Y)$ and therefore also on the acyclic orientations of Y. All permutations contained in components of an orbit under $\mathrm{Aut}(Y)$ give rise to dynamically equivalent sequential dynamical systems, that is, to isomorphic phase spaces. However, here one needs some more technical assumptions, i.e., the local functions must be symmetric and induced, see [11]. This of course also leads to a bound for the number of dynamically inequivalent systems that can be generated by varying the update order alone. Again, this was first done for permutation update orders. The theory was extended to words over the vertex set of Y in [34; 75].

The structure of the graph Y influences the dynamics of the system. As an example, graph invariants such as the independent sets of Y turn out to be in a bijective correspondence with the invariant set of sequential systems over the Boolean field k that have $\mathrm{nor}_t \colon k^t \to k_2$ given by $\mathrm{nor}_t(x_1, \ldots, x_t) = (1 + x_1) \cdots (1 + x_t)$ as local functions [73]. This can be extended to other classes such as those with order independent invariant sets as in [39]. We have already seen how the automorphisms of a graph give rise to equivalence [61]. Also, if the graph Y has non-trivial covering maps we can derive simplified or reduced (in an appropriate sense) versions of the original SDS over the image graphs of Y, see e.g. [62; 74].

Parallel and sequential dynamical systems differ when it comes to invertibility. Whereas it is generally computationally intractable to determine if a CA over $\mathbf{Z}^d$ is invertible for $d \geq 2$ [46], it is straightforward to determine this for a sequential dynamical system [61]. For example, it turns out that the only invertible Boolean sequential dynamical systems with symmetric local functions are the ones where the local functions are either the parity function or the logical complement of the parity function [10].

Some classes of sequential dynamical systems such as the ones induced by the nor-function have desirable stability properties [39]. These systems have minimal invariant sets (i.e. periodic states)

that do not depend on the update order. Additionally, these invariant sets are stable with respect to configuration perturbations.

If a state $\mathbf{c}$ is perturbed to a state $\mathbf{c}'$ that is not periodic this state will evolve to a periodic state $\mathbf{c}''$ in one step; that is, the system will quickly return to the invariant set. However, the states $\mathbf{c}$ and $\mathbf{c}''$ may not necessarily be in the same periodic orbit.

6.3. **The category of sequential dynamical systems.** As a general principle, in order to study a given class of mathematical objects it is useful to study transformations between them. In order to provide a good basis for a mathematical analysis the objects and transformations together should form a *category*, that is, the class of transformations between two objects should satisfy certain reasonable properties (see, e.g., [53]). Several proposed definitions of a transformation of SDS have been published, notably in [48] and [74]. One possible interpretation of a transformation of SDS from the point of view of agent-based simulation is that the transformation represents the approximation of one simulation by another or the embedding/projection of one simulation into/onto another. These concepts have obvious utility when considering different simulations of the same complex system.

One can take different points of view in defining a transformation of SDS. One approach is to require that a transformation is compatible with the defining structural elements of an SDS, that is, with the dependency graph, the local update functions, and the update schedule. If this is done properly, then one should expect to be able to prove that the resulting transformation induces a transformation at the level of phase spaces. That is, transformations between SDS should preserve the local and global dynamic behavior. This implies that transformations between SDS lead to transformations between the associated global update functions.

Since the point of view of SDS is that global dynamics emerges from system properties that are defined locally, the notion of SDS transformation should focus on the local structure. This is the point of view taken in [48]. The definition given there is rather technical and the details are beyond the scope of this article. The basic idea is as follows. Let $\Phi_\pi = f_{\pi(n)} \circ \cdots \circ f_{\pi(1)}$ and $\Phi_\sigma = g_{\sigma(m)} \circ \cdots \circ g_{\sigma(1)}$ with the dependency graphs Y_π and Y_γ, respectively. A transformation $F : \Phi_\pi \longrightarrow \Phi_\sigma$ is determined by

- a graph mapping $\varphi : Y_\pi \longrightarrow Y_\gamma$ (reverse direction);
- a family of maps $k_{\phi(v)} \longrightarrow k_v, v \in Y_\pi$;
- an order preserving map $\sigma \longmapsto \pi$ of update schedules.

These maps are required to satisfy the property that they "locally" assemble to a coherent transformation. Using this definition of transformation, it is shown [48, Theorem 2.6] that the class of SDS forms a category. One of the requirements, for instance, is that the composition of two transformations is again a transformation. Furthermore, it is shown [48, Theorem 3.2] that a transformation of SDS induces a map of directed graphs on the phase spaces of the two systems. That is, a transformation of the local structural elements of SDS induces a transformation of global dynamics. One of the results proven in [48] is that every SDS can be decomposed uniquely into a direct product (in the categorical sense) of indecomposable SDS.

Another possible point of view is that a transformation

$$F : (\Phi_\pi : k^n \to k^n) \longrightarrow (\Phi_\gamma : k^m \to k^m)$$

is a function $F : k^n \longrightarrow k^m$ such that $F \circ \Phi_\pi = \Phi_\gamma \circ F$, without requiring specific structural properties. This is the approach in [74]. This definition also results in a category, and a collection of mathematical results. Whatever definition chosen, much work remains to be done in studying these categories and their properties.

7. Future directions

Agent-based computer simulation is an important method for modeling many complex systems, whose global dynamics emerges from the interaction of many local entities. Sometimes this is the only feasible approach, especially when available information is not enough to construct global dynamic models. The size of many realistic systems, however, leads to computer models that are themselves highly complex, even if they are constructed from simple software entities. As a result it becomes challenging to carry out verification, validation, and analysis of the models, since these consist in essence of complex computer programs. This chapter argues that the appropriate approach is to provide a formal mathematical foundation by introducing a class of mathematical objects to which one can map agent-based simulations. These objects should capture the key features of an agent-based simulation and should be mathematically rich enough to allow the derivation of general results and techniques. The mathematical setting of dynamical systems is a natural choice for this purpose.

The class of finite dynamical systems over a state set X which carries the structure of a finite field satisfies all these criteria. Parallel, sequential, and stochastic versions of these are rich enough to serve as the mathematical basis for models of a broad range of complex systems. While finite dynamical systems have been studied extensively from an experimental point of view, their mathematical theory should be considered to be in its infancy, providing a fruitful area of research at the interface of mathematics, computer science, and complex systems theory.

8. Bibliography

Primary Litreture

[1] R. L. Bagrodia, *Parallel languages for discrete-event simulation models*, IEEE Computational Science and Engineering, 5 (1998), pp. 27–38.

[2] C. L. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns, *On some special classes of sequential dynamical systems.*, Annals of Combinatorics, 7 (2003), pp. 381–408.

[3] ——, *Reachability problems for sequential dynamical systems with threshold functions.*, Theor. Comput. Sci., 295 (2003), pp. 41–64.

[4] ——, *Complexity of reachability problems for finite discrete sequential dynamical systems.*, J. Computer and System Sciences, 72 (2006), pp. 1317–1345.

[5] C. L. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, R. E. Stearns, and M. Thakur, *Computational aspects of analyzing social network dynamics.*, in Proc. International Joint Conference on Artificial Intellligence (IJCAI 2007), 2007, pp. 2268–2273.

[6] ——, *Predecessor existence problems for finite discrete dynamical systems*, 2007. To appear in *Theoretical Computer Science*.

[7] C. L. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, R. E. Stearns, and P. Tosic, *Garden of eden and fixed point configurations in sequential dynamical systems*, in Proc. International Conference on Combinatorics, Computation and Geometry (DM-CCG'2001), 2001, pp. 95–110.

[8] C. L. Barrett, M. V. Marathe, J. P. Smith, and S. S. Ravi, *A mobility and traffic generation framework for modeling and simulating ad hoc communication networks*, in SAC'02 Madrid, Spain, ACM, 2002, pp. 122–126.

[9] C. L. Barrett, H. S. Mortveit, and C. M. Reidys, *Elements of a theory of simulation II: Sequential dynamical systems*, Appl. Math. and Comput., 107 (2000), pp. 121–136.

[10] ——, *Elements of a theory of simulation III: Equivalence of SDS*, Appl. Math. and Comput., 122 (2001), pp. 325–340.

[11] ——, *Elements of a theory of computer simulation. IV. sequential dynamical systems : fixed points, invertibility and equivalence*, Appl. Math. Comput., 134 (2003), pp. 153–171.

[12] C. L. BARRETT AND C. M. REIDYS, *Elements of a theory of simulation I: Sequential CA over random graphs*, Appl. Math. and Comput., 98 (1999), pp. 241–259.

[13] R. BARTLETT AND M. GARZON, *Monomial cellular automata*, Complex Systems, 7 (1993), pp. 367–388.

[14] M. BERNASCHI AND F. CASTIGLIONE, *Selection of escape mutants from immune recognition during hiv infection*, Immunology and Cell Biology, 80 (2002), pp. 307–313.

[15] M. BERNASCHI, S. SUCCI, AND F. CASTIGLIONE, *Large-scale cellular automata simulations of the immune system response*, Phys. Rev. E, 61 (2000).

[16] G. BOOCH, J. RUMBAUGH, AND I. JACOBSON, *Unified Modeling Language User Guide*, Addison-Wesley Professional, 2nd ed., 2005.

[17] D. BRAND AND P. ZAFIROPULO, *On communicating finite-state machines*, J. ACM, 30 (1983), pp. 323–342.

[18] P. CARTIER AND D. FOATA, *Problemes combinatoires de commutation et reárrangements*, vol. 85 of Lecture Notes in Mathematics, Springer Verlag, 1969.

[19] F. CASTIGLIONE AND Z. AGUR, *Analyzing hypersensitivity to chemotherapy in a cellular automata model of the immune system*, in Cancer modeling and simulation, L. Preziosi, ed., Chapman and Hall/CRC Press, London, 2003.

[20] F. CASTIGLIONE, M. BERNASCHI, AND S. SUCCI, *Simulating the Immune Response on a Distributed Parallel Computer*, International Journal of Modern Physics C, 8 (1997), pp. 527–545.

[21] F. CASTIGLIONE, K. DUCA, A. JARRAH, R. LAUBENBACHER, D. HOCHBERG, AND D. THORLEY-LAWSON, *Simulating Epstein-Barr virus infection with C-ImmSim*, Bioinformatics, 23 (2007), pp. 1371–1377.

[22] F. CELADA AND P. SEIDEN, *A computer model of cellular interactions in the immune syste*, Immunology Today, 13 (1992), pp. 56–62.

[23] ——, *A model for simulating cognate recognition and response in the immune system*, J Theor Biol, 158 (1992), pp. 235–270.

[24] ——, *Affinity maturation and hypermutation in a simulation of the humoral immune response*, Eur J Immunol, 26 (1996), pp. 1350–1358.

[25] P. P. CHAUDHURI, *Additive Cellular Automata. Theory and Applications*, vol. 1, IEEE Computer Society Press, 1997.

[26] O. COLÓN-REYES, A. JARRAH, R. LAUBENBACHER, AND B. STURMFELS, *Monomial dynamical systems over finite fields*, Complex Systems, 16 (2006), pp. 333–342.

[27] O. COLÓN-REYES, R. LAUBENBACHER, AND B. PAREIGIS, *Boolean monomial dynamical systems*, Annals of Combinatorics, 8 (2004), pp. 425–439.

[28] D. DAWSON, *Synchronous and asynchronous reversible markov systems*, Canad. Math. Bull., 17 (1974), pp. 633–649.

[29] W. EBELING AND F. SCHWEITZER, *Swarms of particle agents with harmonic interactions*, Theory in Biosciences, 120-3/4 (2001), pp. 207–224.

[30] B. ELSPAS, *The theory of autonomous linear sequential networks*, IRE Transaction on Circuit Theory, (1959), pp. 45–60.

[31] J. FARMER, N. PACKARD, AND A. PERELSON, *The immune system, adaptation, and machine learning*, Phys. D, 2 (1986), pp. 187–204.

[32] U. FRISH, B. HASSLACHER, AND P. Y., *Lattice-gas automata for the Navier-Stokes equations*, Physical Review Letters, 56 (1986), pp. 1505–1508.

[33] H. FUKŚ, *Probabilistic cellular automata with conserved quantities*, Nonlinearity, 17 (2004), pp. 159–173.

[34] L. D. GARCIA, A. S. JARRAH, AND R. LAUBENBACHER, *Sequential dynamical systems over words*, Appl. Math. Comput., 174 (2006), pp. 500–510.

[35] M. GARDNER, *The fantastic combinations of John Conway's new solitaire game "life"*, Scientific American, 223 (1970), pp. 120–123.

[36] M. GOUDA AND C. CHANG, *Proving liveness for networks of communicating finite-state machines*, ACM Transactions on Programming Languages and Systems, 8 (1986), pp. 154–182.

[37] Y. GUO, W. GONG, AND D. TOWSLEY, *Time-stepped hybrid simulation (tshs) for large scale networks*, in INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, vol. 2, 2000, pp. 441–450.

[38] A. GUPTA AND V. KATIYAR, *Analyses of shock waves and jams in traffic flow*, Journal of Physics A, 38 (2005), pp. 4069–4083.

[39] A. Å. HANSSON, H. S. MORTVEIT, AND C. M. REIDYS, *On asynchronous cellular automata*, Advances in Complex Systems, (2005). submitted.

[40] G. HEDLUND, *Endomorphisms and automorphisms of the shift dynamical system*, Math. Syst. Theory, 3 (1969), pp. 320– 375.

[41] A. HERNÁNDEZ-TOLEDO, *Linear finite dynamical systems*, Communications in Algebra, 33 (2005), pp. 2977–2989.

[42] J. HOPFIELD, *Neural networks and physical systems with emergent collective computational properties*, Proc. National Academy of Sciences of the USA, 79 (1982), pp. 2554–2588.

[43] A. ILICHINSKY, *Cellular Automata: A Discrete Universe*, World Scientific Publishing Company, July 2001.

[44] A. JARRAH, R. LAUBENBACHER, M. STILLMAN, AND P. VERA-LICONA, *An efficient algorithm for the phase space structure of linear dynamical systems over finite fields*. Submitted, 2007.

[45] D. R. JEFFERSON, *Virtual time*, ACM Transactions on Programming Languages and Systems, 7 (1985), pp. 404–425.

[46] J. KARI, *Theory of cellular automata: A survey*, Theoretical Computer Science, 334 (2005), pp. 3–33.

[47] B. L. KEYFITZ, *Hold that light! modeling of traffic flow by differential equations.*, Stud. Math. Libr., 26 (2004), pp. 127–153.

[48] R. LAUBENBACHER AND B. PAREIGIS, *Decomposition and simulation of sequential dynamical systems*, Adv. Appl. Math., 30 (2003), pp. 655–678.

[49] R. LIDL AND H. NIEDERREITER, *Finite Fields*, Cambridge University Press, New York, 1997.

[50] T. M. LIGGETT, *Interacting particle systems*, Classics in Mathematics, Springer-Verlag, Berlin, 2005. Reprint of the 1985 original.

[51] D. A. LIND, *Applications of ergodic theory and sofic systems to cellular automata*, Physica D, 10D (1984), pp. 36–44.

[52] K. LINDGREN, C. MOORE, AND M. NORDAHL, *Complexity of two-dimensional patterns*, Journal of Statistical Physics, 91 (1998), pp. 909–951.

[53] S. MAC LANE, *Category Theory for the Working Mathematician*, no. 5 in GTM, Springer Verlag, 2nd ed., 1998.

[54] M. W. MACY, J. A. KITTS, AND A. FLACHE, *Dynamic social network nodeling and analysis*, The National Academies Press, 2003, ch. Polarization in Dynamic Networks: A Hopfield Model of Emergent Structure, pp. 162–173.

[55] O. MARTIN, A. ODLYZKO, AND S. WOLFRAM, *Algebraic properties of cellular automata*, Commun. Math. Phys., 93 (1984), pp. 219–258.

[56] D. MILLIGAN AND M. WILSON, *The behavior of affine boolean sequential networks*, Connection Science, 5 (1993), pp. 153–167.

[57] N. MINAR, R. BURKHART, C. LANGTON, AND A. MANOR, *The swarm simulation system: A toolkit for building multi-agent simulations*, Santa Fe Institute preprint series, (1996).

`http://www.santafe.edu/sfi/publications/96wplist.html`.

[58] J. Misra, *Distributed discrete-event simulation*, ACM Computing Surveys, 18 (1986), pp. 39–65.

[59] T. Moncion, G. Hutzler, and P. Amar, *Verification of biochemical agent-based models using petri nets*, in International Symposium on Agent Based Modeling and Simulation, ABModSim'06, T. Robert, ed., Austrian Society for Cybernetics Studies, 2006, pp. 695–700.

[60] D. Morpurgo, R. Serentha, P. Seiden, and F. Celada, *Modelling thymic functions in a cellular automaton*, International Immunology, 7 (1995), pp. 505–516.

[61] H. S. Mortveit and C. M. Reidys, *Discrete, sequential dynamical systems*, Discrete Mathematics, 226 (2001), pp. 281–295.

[62] ——, *Reduction of discrete dynamical systems over graphs*, Advances in Complex Systems, 7 (2004), pp. 1–20.

[63] K. Nagel, M. Rickert, and C. L. Barrett, *Large-scale traffic simulation*, Lecture notes in computer science, 1215 (1997), pp. 380–402.

[64] K. Nagel and M. Schreckenberg, *A cellular automaton model for freeway traffic*, Journal de physique I, 2 (1992), pp. 2221–2229.

[65] K. Nagel, M. Schreckenberg, A. Schadschneider, and N. Ito, *Discrete stochastic models for traffic flow*, Physical Review E, 51 (1995), pp. 2939–2949.

[66] K. Nagel and P. Wagner, *Traffic Flow: Approaches to Modelling and Control*, John Wiley & Sons, 2006.

[67] R. E. Nance, *A history of discrete event simulation programming languages*, ACM SIGPLAN Notices, 28 (1993), pp. 149–175.

[68] M. J. North, N. T. Collier, and J. R. Vos, *Experiences creating three implementations of the repast agent modeling toolkit.*, ACM Trans. Modeling and Computer Simulation, 16 (2006), pp. 1–25.

[69] P. Orponen, *Computational complexity of neural networks: A survey*, Nordic Journal of Computing, 1 (1994), pp. 94–110.

[70] ——, *The computational power of discrete hopfield networks with hidden units*, Neural Computation, 8 (1996), pp. 403–415.

[71] J. K. Park, K. Steiglitz, and W. P. Thruston, *Soliton-like behavior in automata*, Physica D, 19D (1986), pp. 423–432.

[72] C. Reidys, *Acyclic Orientations of Random Graphs*, Advances in Applied Mathematics, 21 (1998), pp. 181–192.

[73] C. M. Reidys, *On acyclic orientations and sequential dynamical systems*, Adv. Appl. Math., 27 (2001), pp. 790–804.

[74] ——, *On Certain Morphisms of Sequential Dynamical Systems*, Discrete Mathematics, 296 (2005), pp. 245–257.

[75] ——, *Sequential dynamical systems over words*, Annals of Combinatorics, 10 (2006).

[76] M. Rickert, K. Nagel, M. Schreckenberg, and A. Latour, *Two lane traffic simulations using cellular automata*, Physica A, 231 (1996), pp. 534–550.

[77] D. H. Rothman, *Cellular-automaton fluids: A model for flow in porous media*, Geophysics, 53 (1988), pp. 509–518.

[78] S. Russell and P. Norwig, *Artificial Intelligence: A Modern Approach*, Prentice-Hall, 2003.

[79] B. Schönfisch and A. de Roos, *Synchronous and asynchronous updating in cellular automata*, BioSystems, 51 (1999), pp. 123–143.

[80] I. Shmulevich, E. R. Dougherty, S. Kim, and W. Zhang, *Probabilistic boolean networks: A rule-based uncertainty model for gene regulatory networks*, Bioinformatics, 18 (2002), pp. 261–274.

[81] I. Shmulevich, E. R. Dougherty, and W. Zhang, *From boolean to probabilistic boolean networks as models of genetic regulatory networks*, Proceedings of the IEEE, 90 (2002),

pp. 1778–1792.

[82] M. Sipser, *Introduction to the Theory of Computation*, PWS Publishing Co., 1997.

[83] L. Vasershtein, *Markov processes over denumerable products of spaces describing large system of automata*, Problemy Peredachi Informatsii, 5 (1969), pp. 64–72.

[84] J. von Neumann, *Theory of Self-Reproducing Automata*, University of Illinois Press, 1966. Edited and completed by Arthur W. Burks.

[85] G. Whitham, *Linear and Nonlinear Waves*, Pure and Applied Mathematics: A Wiley-Interscience Series of Texts, Monographs and Tracts, Wiley-Interscience, reprint edition ed., 1999.

[86] S. Wolfram, *Statistical mechanics of cellular automata*, Rev. Mod. Phys., 55 (1983), pp. 601–644.

[87] ——, *Theory and Applications of Cellular Automata*, vol. 1 of Advanced Series on Complex Systems, World Scientific Publishing Company, 1986.

[88] ——, *A New Kind of Science*, Wolfram Media, 2002.

Books and Reviews

[1] Hopcroft, J. E., R. Motwani, and J. D. Ullman (2000). *Automata Theory, Languages and Computation.* Reading: Addison Wesley.

[2] Kozen, D. C. (1997). *Automata and Computability.* New York: Springer-Verlag.

[3] Wooldridge, M. (2002). *Introduction to Multiagent Systems.* Chichester, UK: John Wiley & Sons.

Virginia Bioinformatics InstituteVirginia Polytechnic Institute and State University
E-mail address, Reinhard Laubenbacher: `reinhard@vbi.vt.edu`
E-mail address, Abdul S. Jarrah: `ajarrah@vbi.vt.edu`
E-mail address, Henning Mortveit: `henning@vbi.vt.edu`

Department of Computer Science, University at Albany–State University of New York
E-mail address, S. S. Ravi: `ravi@cs.albany.edu`