Science and Engineering of Large Scale Socio-Technical Simulations [1]

Chris Barrett Stephen Eubank Madhav Marathe
Henning Mortveit Christian Reidys

Abstract: *Computer simulation is a computational approach whereby* ***global system*** *properties are produced as dynamics by direct computation of interactions among representations of* ***local*** *system* ***elements****. A mathematical theory of simulation consists of an account of the formal properties of sequential evaluation and composition of interdependent local mappings. When certain local mappings and their interdependencies can be related to particular real world objects and interdependencies, it is common to compute the interactions to derive a symbolic model of the global system made up of the corresponding interdependent objects. The formal mathematical and computational account of the simulation provides a particular kind of theoretical explanation of the global system properties and, therefore, insight into how to engineer a complex system to exhibit those properties.*

This paper considers the mathematical foundations and engineering principles necessary for building large scale simulations of socio-technical systems. Examples of such systems are urban regional transportation systems, the national electrical power markets and grids, the world-wide Internet, vaccine design and deployment, theater war, etc. These systems are composed of large numbers of interacting human, physical and technological components. Some components adapt and learn, exhibit perception, interpretation, reasoning, deception, cooperation and noncooperation, and have economic motives as well as the usual physical properties of interaction. The systems themselves are large and the behavior of socio-technical systems is tremendously complex.

The state of affairs for these kinds of systems is characterized by very little satisfactory formal theory, a good deal of very specialized knowledge of subsystems, and a dependence on experience-based practitioners' art. However, these systems are vital and require policy, control, design, implementation and investment. Thus there is motivation to improve the ability to comprehend them by use of whatever means, including computer simulation. Moreover, the general theoretical understanding of the system properties provided by the formalization of simulation, is of great value given the otherwise poor state of understanding of the systems themselves.

Theoretically, ***Sequential Dynamical Systems*** *(SDS) are introduced as a mathematical model of discrete simulation. Sequential dynamical systems are compositions of local maps. The order of composition reflects casual relationships between individual agents abstracted as functions and the locality of the functions reflects limited interaction and knowledge of the entire system available to each agent. The properties of SDS are very general and allow much deeper understanding of both the simulation and the simulated system.*

The last part considers the engineering principles derived from such a theory. These engineering principles allow us to specify, design, and analyze simulations of extremely large systems and implement them on massively parallel architectures. These ideas are illustrated by several socio-technical simulations being developed at Los Alamos National Laboratory.

1 Introduction and Motivation

This paper considers theoretical foundations and associated engineering science for simulating large scale socio-technical systems. The name *socio-technical systems* indicates the fact that such systems are defined by essential physical, technological, and human/societal components. Examples of such systems include: transportation systems, financial systems, electrical power markets, communication systems, health care systems, supply chain management systems and war fighting systems. As an example of this general class, note that all critical national infrastructures are large scale socio-technical systems.

[1] Basic and Applied Simulations Sciences, Los Alamos National Laboratory, MS M997, P.O. Box 1663, Los Alamos, NM 87545. Email: {barrett, eubank, marathe, henning, reidys }@lanl.gov. The work is supported by the Department of Energy under Contract W-7405-ENG-36.

These systems provide an excellent setting for the study of simulation for at least two reasons. First, because they are so distributed, diverse and display such complexity, there is ample motivation for using computer simulation to represent them. Second, for the same reasons there is little theoretical understanding of particular socio-technical systems themselves. This, somewhat ironically perhaps, can serve to focus attention on the general properties of simulation that might structure understanding of all of such systems rather than an immediate retreat to consideration of a computational treatment of a particular system-specific theory. Such restriction of focus of computational methods is common, for example, in the use of simulation in the purely physical sciences. Indeed, a thesis of this paper is that many properties of actual socio-technical systems are *formally simulated* in the sense that these properties are dynamical manifestations of composed, interacting local subsystems, e.g., virtual circuits in packet-switched networks, closing prices in markets, congestion in vehicular traffic, etc. Thus, not withstanding that computer simulation is a valuable means with which to represent socio-technical systems, the formalization of simulation serves to provide generalities and principles over these systems themselves which otherwise have little theory associated with them.

The concept of a society of individuals – a fundamental aspect of socio-technical systems is largely a matter of conventions for interactions and transactions, i.e. traffic and information dissemination of one sort or another. Thus, goods of various sort including informational, financial and vehicular whose movement gives rise to traffic and the design as well as regulation of network resources on which this traffic moves are important issues that need to be tackled effectively. Traffic (of all sorts) and infrastructures are also important to scientists, engineers and business persons. They may seek to understand, design, modify the actual in-place systems, develop relevant technology, or market products and services.

Just as with socio-technical systems, computer simulation is also really a matter of (procedural) interactions and (symbolic) transformations and transactions, conventions and (data) traffic. As a result, it seems intuitively plausible that computer simulations can represent certain essentials of socio-technical systems and thus perhaps assist us to understand or design them. Examination of this intuition gives useful insight regarding simulation as a formal means of system representation. There is much new as of yet to be discovered regarding both the prospects and limitations of computer simulation. Some of our theoretical results already address whether/when simulation is merely optional, whether it is, computationally speaking, universal or a more restricted class of computation, and many other important and pertinent issues.

From the highest levels of government and the private sector to system designers, operators and students, computer simulations are used to combine data, knowledge, and situational context and assist human reasoning and decision-making in complex socio-technical environments (e.g., critical infrastructure systems, military command and control, etc.) The level of national interest in social infrastructure in representation, analysis, and technical support for policy-level decision making and situation management is found in PDD39 (countering chem-bio terrorism) and PDD63 (protection of critical/cyber infrastructure).

We will describe a formal conceptual organization of these issues as they relate to simulations generally and the use of simulations of socio-technical systems for practical purposes.

2 Computer Simulations and Discrete Dynamical Systems

Informally, computer simulation is the art and science of using computers to calculate interactions and transactions among many separate algorithmic representations, each of which might be associated with identifiable "things" in the real world (at least in a world outside the simulation program). In most cases it is fair to assume that the interactions between individual objects is local. Simulations compose these "local mappings" and thereby generate "global phenomena".

Because of the widespread use of computer simulations, it is difficult to give a formal definition of a computer simulation that is applicable to all the various settings where it is used. Nevertheless, it is clear that simulation has two essential aspects. First, a simulation is a dynamics generator. A computer simulation program is a means by which to compose many distinct, data local, iterated procedures. Each procedure's dynamics is affected by the composition of procedures and in that sense the composition produces global dynamics. The second, aspect of simulation is mimicry of the dynamics of another system by the dynamics of the simulation program. After mimicry is established, the simulation program is often called a simulation model but this is a separate collection of issues. We will discuss the first part in detail, but also touch upon the second part in the subsequent sections. Thus we view simulations as comprised of the following: (i) a collection of entities with state values and local rules for state transitions, (ii) an interaction graph capturing the local dependency of an entity on its neighboring entities and (iii) an update sequence or schedule such that the causality in the system is represented by the composition of local mappings.

2.1 An Elementary Theory of Simulation: SDS

An elementary theory of simulation should yield theorems that are applicable to a class of simulations rather than to only particular members of this class. In a series of papers [11, 12, 13, 15, 17, 20, 21, 22, 52, 74, 87, 89] we have begun to develop such an elementary theory. The theory is based on a discrete dynamical systems viewpoint of simulation – indeed the goal of simulations is to produce dynamical behavior via composition of local interactions. Sequential Dynamical Systems (SDS) are a new class of discrete finite dynamical systems. A formal definition of such a system is given in Section 4. Informally, an SDS $\mathcal{S}$ is a 3-tuple $(Y, \mathcal{F}, \pi)$. Here $Y(V, E)$ is an undirected graph with n nodes. Associated with each node is a state value drawn from a finite domain. $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$ is a set of functions such that the function f_i is associated with node v_i. Finally, π is a permutation of (or a total order on) the nodes in V. A **configuration** of an SDS is an n vector $(c_1, c_2, \ldots, c_n)$, where c_i is the value of the state of node v_i ($1 \le i \le n$). A single SDS transition from one configuration to another is obtained by updating the state of each node v_i using the corresponding function f_i. These updates are carried out in the order specified by π. An SDS is denoted by $[F_Y, \pi]$, if all the local functions f_i are of a particular kind say Nor, we will sometimes refer to these SDS as $[\mathrm{Nor}_Y, \pi]$ or Nor-SDS. In general if we have a fixed function $\mathcal{F}$ at each node or if the local transition functions are drawn from a set $\mathcal{B}$ then we sometimes refer to such SDS as $[\mathcal{F}_Y, \pi]$ (or $\mathcal{F}$-SDS), $[\mathcal{B}_Y, \pi]$ (or $\mathcal{B}$-SDS) respectively.

Example 1: Consider the graph $Y = \mathrm{Circ}_4$ as shown in figure 1. To each vertex i in Y we associate a state $x_i \in \{0,1\} = \mathbb{F}_2$. The parity function $\mathrm{par}_3 : \mathbb{F}_2^3 \to \mathbb{F}_2$ is defined by $\mathrm{par}_3(x_1, x_2, x_3) = x_1 + x_2 + x_3 \mod 2$. In

Figure 1: The circle graph on 4 vertices, Circ_4.

the usual enumeration scheme of elementary CA rules this is rule 150. We introduce the functions $\mathrm{Par}_i : \mathbb{F}_2^4 \to \mathbb{F}_2^4$, $1 \le i \le 4$ by

$$\begin{aligned}
\mathrm{Par}_1(x_1, x_2, x_3, x_4) &= (\mathrm{par}_3(x_1, x_2, x_4), x_2, x_3, x_4),\\
\mathrm{Par}_2(x_1, x_2, x_3, x_4) &= (x_1, \mathrm{par}_3(x_1, x_2, x_3), x_3, x_4),\\
\mathrm{Par}_3(x_1, x_2, x_3, x_4) &= (x_1, x_2, \mathrm{par}_3(x_2, x_3, x_4), x_4),\\
\mathrm{Par}_4(x_1, x_2, x_3, x_4) &= (x_1, x_2, x_3, \mathrm{par}_3(x_1, x_3, x_4)).
\end{aligned}$$

Thus the map Par_i updates the state of vertex i based on the states of i and its neighbors in Y and leaves all other states fixed. We apply these maps to the state $x = (1,1,0,0)$ in the order $\pi = (1,2,3,4)$. At each stage we use the value of the previous function as the input to the next function, i.e.

$$\begin{aligned}
&(1,1,0,0) \overset{\mathrm{Par}_1}{\mapsto} (0,1,0,0) \overset{\mathrm{Par}_2}{\mapsto}\\
&(0,1,0,0) \overset{\mathrm{Par}_3}{\mapsto} (0,1,1,0) \overset{\mathrm{Par}_4}{\mapsto}\\
&(0,1,1,1).
\end{aligned}$$

Thus we have $\mathrm{Par}_4 \circ \mathrm{Par}_3 \circ \mathrm{Par}_2 \circ \mathrm{Par}_1(1,1,0,0) = (0,1,1,1)$. The composition of maps $\mathrm{Par}_4 \circ \mathrm{Par}_3 \circ \mathrm{Par}_2 \circ \mathrm{Par}_1$ is a sequential dynamical system (SDS). Specifically, it is the SDS over the graph Circ_4 induced by the parity function $\mathrm{par}_3 : \mathbb{F}_2^3 \to \mathbb{F}_2$ with ordering $\pi = (1,2,3,4)$. We denote this by $[\mathbf{Par}_{\mathrm{Circ}_4}, \pi]$. Clearly, a different update order may give a different result.

By iterating the map $\phi = [\mathbf{Par}_{\mathrm{Circ}_4}, \pi]$ we obtain the *orbit* of $(1,1,0,0)$, i.e.

$$\begin{tikzcd}
(1,1,0,0) \arrow[rr, "\phi"] & & (0,1,1,1) \arrow[dl, "\phi"] \\
& (0,0,0,1). \arrow[ul, "\phi"] &
\end{tikzcd}$$

The *phase space* of the SDS ϕ is the union of all such cycles and possible transients. It is easy to see that SDS capture the three essential elements of any computer simulation. The use of simple functions to represent each agent/entity is just an equivalent alternate representation of each individual as automata. The fact that each function depends locally on the state values of neighboring agents is intended to capture the intuition that individual objects comprising a real system usually have local knowledge about the state of system. Finally, a permutation is an abstraction of the need to *explicitly* encode causal dependency. Extensions of this basic model will be discussed in the subsequent sections. The informal description of an SDS given above can be seen to capture all of the above features.

2.2 Modeling and Computational Power of SDS

An immediate question that arises is the following: How expressive is the SDS framework, and what is its computational power? Clearly, we need models that provide a delicate balance between modeling power and the associated computational complexity. In this context, the results presented here strongly support the following assertions.

- Large-scale real world distributed systems can be efficiently represented as SDS.
- Typical "state prediction" problems are computationally intractable even for very restricted instances of SDS. Thus, it is unlikely that methods more efficient than computer simulations can be devised for the state prediction problem.
- SDS are "simple" yet computationally universal[2].

Consider the first assertion. References [7, 24, 25, 32] show how simulations of large-scale socio-technical systems can be modeled using appropriate SDS. The local interaction rules for entities and a dependency graph structure are by now accepted as standard aspects of discrete dynamical systems for modeling large-scale systems. The ordering aspect is somewhat new in a formal setting but has recently received attention by other researchers [51, 39, 91]. It is implicit in all discrete event simulations. Consider a simple yet a realistic example of a simulation system that uses this abstraction.

Example 2: **TRANSIMS** is a large scale transportation simulation project at the Los Alamos National Laboratory. In this project, an SDS-based approach was used to micro-simulate every vehicle in an urban transportation network. Each roadway is divided into discrete cells. Each cell is 7.5 meters long and one lane wide. Each cell contains either a vehicle (or a part of a vehicle) or is empty. The micro-simulation is carried out in discrete time steps with each step simulating one second of real traffic. In each time step, a vehicle on the network makes decisions such as accelerate, brake or change lanes, in response to the occupancy of the neighboring cells. We can represent the above model in our SDS framework. For ease of exposition, we assume a single lane road which can be modeled as a one dimensional array of cells, with each cell representing a certain segment of the road. The state of each car (driver) may assume one of $v_{max}+1$ possible integer values; these values correspond to discrete speeds from 0 to v_{max}. The state of each cell may assume one of $v_{max}+2$ different values, the additional value being used to represent an empty cell. In the TRANSIMS system implementation, v_{max} was usually a small integer (such as 5). The rule by which a car updates its state (location, speed and lane) is a simple function of its state and the states of the cars in the neighboring cells. It is now easy to see how the situation can be formulated in terms of an SDS. An important point to note is that unlike cellular automata (CA) which are synchronous, different choices of the order for updating the cells may yield completely different dynamics in the case of SDS. For instance, in the case of the single-lane system, updating the states from front to back acts like a perfect predictor and thus never yields clusters of vehicles. On the other hand, updating from back to front yields more realistic traffic dynamics [7].

Given the above model, a simulation question that arises in practice can be transformed into an appropriate analysis question for SDS. For instance, the question of whether the system starting from a given initial configuration will ever reach a traffic-jam-like configuration can be cast as a reachability problem for an appropriate SDS.

Consider the second assertion. Following [27], we say that a system is predictable if basic phase space properties such as reachability and fixed point reachability can be determined in time which is polynomial in the size of the system specification. Our **PSPACE**-completeness results for predicting the behavior of "very simple" SDS (e.g. SDS in which the domain of state values is Boolean and each node computes the same symmetric Boolean function) essentially imply that the systems are not easily predictable; in fact, our results imply that no prediction method is likely to be more efficient than running the simulation itself.

Finally, we consider the third assertion. We show that SDS are "universal" in that any reasonable model

[2]The notion of universality although a little more subtle can be defined for space and time complexity classes; See [43].

of simulation can be "efficiently locally simulated" by appropriate SDS that can be constructed in polynomial time. The models investigated include cellular automata, communicating finite state machines, multi-variate difference equations, etc. Moreover, in most of these cases SDS can also be locally simulated by these devices. Thus, lower bounds on the computational complexity of deciding some properties of SDS yield as direct corollaries analogous results for those models. The models include the following:

(a) Classical CA (see for example, [101]) and **graph automata** [78, 70], which are a widely studied class of dynamical systems in physics and complex systems.

(b) Discrete Hopfield networks [48, 34], which are a classical model for machine learning, and

(c) Communicating finite state machines [3, 2], which are widely used to model and verify distributed systems.

The main difference between graph automata and SDS is that in the former case node states are updated in parallel while in latter case they are updated in a specified sequential order. The notion of universality is applicable not only in the context of classical computability theory, but is also applicable for for space and time complexity classes; See [43, 101, 80]. It can be shown that appropriately defined SDS are universal for each of the "natural" space and time complexity classes [21, 22, 52]. For instance, SDS with when the interaction graph is defined using natural specifications and the local transition functions are over finite domain yield a universal device for **PSPACE**. When the graph is specified succinctly using periodic specifications yielding graphs exponentially larger than the specifications and with local transition functions over finite domains we get a universal device for **EXPSPACE**. We can also get a universal device for **EXPSPACE** by having polynomially many nodes but exponential amount of memory at each node. Finally, using a simple simulation of a two counter machine, we can show that a two node SDS with each node computing functions over integers are universal in the classical sense – they are as powerful as Turing machines [21, 22].

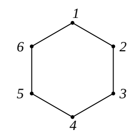

Figure 2: The graph Circ_6.

3 Overview and Discussion of Results

We give a brief overview of our results on SDS and discuss their implications. In Section 5 we give formal statements of these results. Due to lack of space, we only outline some of the results here. Additional results and their discussion can be found in [11, 12, 13, 15, 17, 20, 21, 22, 23, 87, 89].

Equivalence. Theorem 5.1 deals with the question of equivalence of two SDS and is a well-known problem in the context of validating computer simulations. Essentially, we discuss the question on how many functionally different[3] systems can be obtained by just varying the corresponding update schedule?

Example 3: Order dependence of* SDS*: The function $\mathrm{maj}_3 : \mathbb{F}_2^3 \to \mathbb{F}_2$ is defined by

$$\mathrm{maj}(x_1, x_2, x_3) = \begin{cases} 1, & x_1 + x_2 + x_3 \geq 2 \\ 0, & \text{otherwise,} \end{cases}$$

where $''+''$ denotes the addition in the ring $(\mathbb{Z}, +)$. Consider the SDS induced by maj_3 over the graph Circ_6 as shown in figure 2. It is easy to verify that $x_0 = (1, 1, 1, 0, 0, 0)$ is a fixed point for any SDS of the form $[\mathrm{Maj}_{\mathrm{Circ}_6}, \pi]$. (Clearly, fixed points are invariant with respect to schedule.) Using the schedule $\pi_1 = (2, 1, 3, 5, 4, 6)$ we see that the states $(1, 0, 1, 0, 0, 0)$, $(1, 0, 1, 0, 1, 0)$ and $(1, 1, 1, 0, 1, 0)$ are mapped to the fixed point x_0. These are the only states apart from x_0 itself that is mapped to x_0.

By changing the schedule to $\pi_2 = (1, 3, 2, 4, 6, 5)$ *no* states apart from x_0 itself are mapped to x_0. Thus by a change of schedule the "basin of attraction" of the fixed point x_0 has vanished. See figure 3.

In the context of this question various deep relations between the number of functionally different SDS and certain combinatorial quantities associated to their underlying dependency graph arise. A fixed point of an SDS is a configuration $\mathcal{C}$ such that the transition out of $\mathcal{C}$ is to $\mathcal{C}$ itself. A configuration

[3] see figures 4 and 5

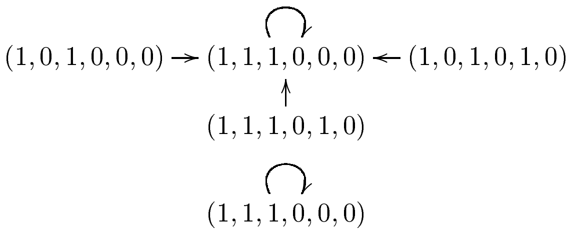

Figure 3: The phase space component of $[\mathrm{Maj}_{\mathrm{Circ}_6}, \pi_1]$ (top) and $[\mathrm{Maj}_{\mathrm{Circ}_6}, \pi_2]$ (bottom) containing the fixed point $(1,1,1,0,0,0)$. The basin of attraction of the fixed point $(1,1,1,0,0,0)$ vanishes under the change of schedule $\pi_1 \mapsto \pi_2$.

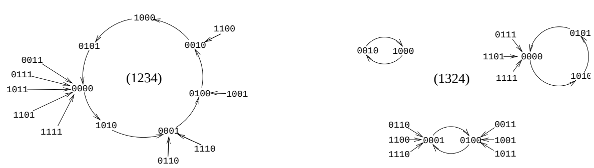

Figure 4: The phase spaces of the SDS $[\mathbf{Nor}_{\mathrm{Circ}_4}, (1234)]$ and $[\mathbf{Nor}_{\mathrm{Circ}_4}, (1324)]$. In these SDS all Boolean functions F_1, F_2, F_3, F_4 are **Nor** functions. On the lhs. the underlying update order is (1234), whereas on the rhs. the order is (1324). Obviously, the phase spaces are not isomorphic as directed graphs.

that has no predecessor is called a Garden of Eden configuration [90, 101].

Fixed Points, Garden of Eden & Nor-systems. The question on the sharpness of a combinatorial upper bound on this number leads to Theorem 5.2 which describes the structure of SDS induced by Boolean nor function. In fact, it appears that these systems play a special role in SDS analysis, and they turn out to be the only SDS which are fixed-point free over arbitrary base graphs. In the figures 4 and 5 we present two complete phase spaces of SDS induced by nor over the square, Circ_4. Both figures allow us to illustrate particular features of SDS induced by nor functions: They exhibit only non-periodic points that are Garden-of-Eden states and changing the underlying update schedule may (see figure 4) or may not (as shown in 5) induce functionally different SDS.

A related question in this regard is to characterize the computational complexity of computing various phase space configurations. Our results show that in general it is **NP**-hard to decide if a given SDS has a Garden-of-Eden configuration (Fixed Point configuration respectively). The hardness results essentially rely on the non-deterministic nature of the problem. Moreover, counting the number of such configurations is also typically #**P**-complete.[4] On the other hand when the underlying graph has a special graph theoretic structure (the graph is of bounded treewidth[5]) then these invariants can be computed in polynomial time.

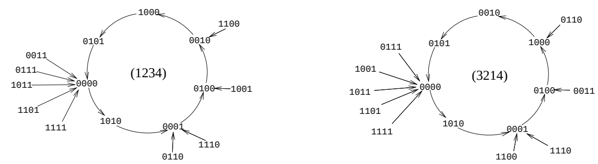

Figure 5: Using the update orders $\sigma = (1234)$ and $\pi = (3214)$ and the Boolean function **Nor** for F_1, F_2, F_3, F_4 the phase spaces of the corresponding SDS $[\mathbf{Nor}_{\mathrm{Circ}_4}, \sigma]$ (rhs.) and $[\mathbf{Nor}_{\mathrm{Circ}_4}, \pi]$ (lhs.) are not identical although their digraphs are isomorphic. The digraphs differ, for instance, at the following states: the preimages of 0000 are not all identical, the preimages of 0001 differ, state 0100 has different preimages and maps into a different state.

Reachability, Universality, and Local Simulation. Another very important question regarding the phase space of an SDS is: Can one reach a given global configuration starting from another given global configuration? This is the well-known reachability problem, and it has direct implications to a host of other combinatorial questions. Not surprisingly, it turns out that the reachability question is **PSPACE**-hard even for SDS whose local transition functions are Boolean symmetric functions. The result is obtained via a sequence of "local simulations" that begin by simulating a linearly bounded Automata (LBA) by an SDS with finite domain. By allowing an exponential memory at each node or allowing exponentially many nodes, one can in fact obtain **EXPSPACE**-hardness results. An important implication of this **stated very informally** is *Simulation of such dynamical systems to discern their dynamical properties is an optimal computational strategy or that simulation is not optional.* Moreover the systems for which the hardness results are so simple (essentially, local transition functions can be simple threshold or inverted thresholds) that any realistic socio-technical system is likely to have

[4]We refer the reader to any of the classical texts on complexity theory for a definitions of these well-known complexity classes and their significance.

[5]Informally, a class of graphs are said to have bounded treewidth if they have a recursive separator of bounded size.

such systems embedded in them. This is important since, one might ask, whether some other analytical method or an efficient algorithm can be used instead of iterated composition of local maps.

In the past, communicating finite state machines (CFSM) [40, 45, 46, 47, 71, 82] have been used extensively to specify and verify important properties such as liveness, deadlock, etc. of communication protocols, models of infectious diseases, driving logics, etc. It is easy to see that CFSMs are closely related to SDS. Ideally, we would like to express the systems in "higher level SDS" (or CFSMs and then translate them into simpler kinds of SDS (akin to compilation). This is because the language (model) that is most convenient to describe the underlying system might not necessarily be the best model for actual simulation of the system on a HPC architecture. Thus it is conceivable that such simpler systems obtained via translation could be mapped on HPC architectures and the resulting maps could be analyzed for performance bottlenecks. Simpler systems can also be potentially used to verify the correctness of the ensuing protocols. To achieve this, such translations should be efficient and should preserve the basic properties across the original and the translated system. In recent years (see [69, 36, 98, 29], several authors have suggested building *cellular automata based computers* for simulating physics. The results presented here are pertinent to this basic theme in two ways. We believe that *SDS based computers* are better suited for simulating socio-technical systems. Second, regardless, of the final model, it is indeed necessary to provide efficient simulations (translations) of problems specified in one model to problem specified in another model. Without such simulations, building such computers is likely to be of very limited use.

A motivation for obtaining complexity theoretic results reported here is derived from the papers of Buss, Papadimitriou and Tsitsiklis [27], Moore [72, 73], Sutner [96] and Wolfram [101]. Specifically, we undertake the computational study of SDS in an attempt to increase our understanding of SDS in particular and the complex behavior of dynamical systems in general. SDS are discrete finite analogs of classical dynamical systems, and we aim to obtain a better understanding of "finite discrete computational analogs of chaos". As pointed out in [27, 72, 73], computational intractability or unpredictability is the closest form of chaotic behavior that such systems can exhibit. Extending the work of [27], we prove a dichotomy result between classes of SDS whose global behavior is easy to predict and others for which the global behavior is hard to predict. In [101], Wolfram posed the following three general questions in the chapter entitled "Twenty Problems in the Theory of Cellular Automata": (i) **Problem 16:** *How common are computational universality and undecidability* in CA? (ii) **Problem 18:** *How common is computational irreducibility in CA*? (iii) **Problem 19:** *How common are computationally intractable problems about CA*? Our results [17, 20, 21, 22] for SDS (and for CA as direct corollaries) show that the answer to all of the above questions is "*quite common*". In other words, it is quite common for synchronous dynamical systems (i.e., CA) as well as sequential dynamical systems to exhibit intractability. In fact, our results show that such intractability is exhibited by *extremely simple* SDS and CA.

Factorization of SDS. We just saw that one could in appropriate cases "locally simulate" one SDS $\mathcal{S}$ by another SDS $\mathcal{T}$. In most cases such transformations yield an SDS $\mathcal{T}$ that are somewhat larger (although the size of $\mathcal{T}$ is of the same order as the size of $\mathcal{S}$). The next set of results aim at creating equivalent but "smaller" SDS. Typically, the phase space of an SDS has more than one attractor or component and consequently, a time series will only visit parts of phase space. Thus, in the context of computer simulations, there will be valid states or regimes that are never realized. Accordingly, one is interested in constructing a "reduced" simulation system capable of producing a somewhat related dynamics in the "essential" regimes and that ideally disposes of the "non-essential" regimes. In this paper we try to address this question by establishing an embedding of SDS-phase spaces under certain conditions. To be more explicit, we will show how to relate an SDS ϕ over a graph Y and an SDS ψ over a smaller sized-graph Z if there exists a covering map $p : Y \to Z$. Let us illustrate this idea by considering the following:

Example 4: Morphism Concepts for **SDS.** We

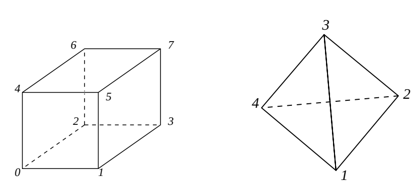

Figure 6: The graphs Q_2^3 and K_4.

will consider sequential dynamical systems over Q_2^3 and K_4 induced by the parity function.

The two graphs Q_2^3 and K_4 are related by the map (graph morphism) $p : Q_2^3 \to K_4$ by $p^{-1}(\{1\}) = \{0, 7\}$, $p^{-1}(\{2\}) = \{1, 6\}$, $p^{-1}(\{3\}) = \{2, 5\}$, and $p^{-1}(\{4\}) = \{3, 4\}$. Note that p is a *covering map*.

The map p naturally induces an embedding $\tau : \mathbb{F}_2^4 \to \mathbb{F}_2^8$ by $(\tau(x))_k = x_{p(k)}$, that is,

$$\tau(x_1, x_2, x_3, x_4) = (x_1, x_2, x_3, x_4, x_4, x_3, x_2, x_1) \ .$$

Let $\pi = (1, 2, 3, 4) \in S_4$, and let $\pi_p = (0, 7, 1, 6, 2, 5, 3, 4)$. We now have a commutative diagram

$$\begin{tikzcd}
\mathbb{F}_2^4 \arrow[r, "{[\mathrm{Par}_{K_4}, (1,2,3,4)]}"] \arrow[d, "\tau"] & \mathbb{F}_2^4 \arrow[d, "\tau"] \\
\mathbb{F}_2^8 \arrow[r, "{[\mathrm{Par}_{Q_2^3}, \pi_p]}"] & \mathbb{F}_2^8
\end{tikzcd} \ .$$

Let $x = (1, 0, 0, 0)$. Straightforward calculations give $[\mathrm{Par}_{K_4}, (1, 2, 3, 4)](1, 0, 0, 0) = (1, 1, 0, 0)$, and further iterations give the orbit

$$\begin{tikzcd}
(1,0,0,0) \arrow[r] & (1,1,0,0) \arrow[r] & (0,1,1,0) \arrow[dl] \\
(0,0,0,1) \arrow[u] & (0,0,1,1) \arrow[l] &
\end{tikzcd} \ .$$

Note that

$$\begin{tikzcd}
(1,0,0,0) \arrow[r, "{[\mathrm{Par}_{K_4}, (1,2,3,4)]}"] \arrow[d, "\tau"] & (1,1,0,0) \arrow[d, "\tau"] \\
(1,0,0,0,0,0,0,1) \arrow[r, "{[\mathrm{Par}_{Q_2^3}, \pi_p]}"] & (1,1,0,0,0,0,1,1)
\end{tikzcd} \ .$$

By applying the map τ to the cycle of $(0, 0, 0, 0)$ under $[\mathrm{Par}_{K_4}, (1, 2, 3, 4)]$ it is easily verified that we obtain the orbit of $(1, 0, 0, 0, 0, 0, 0, 1)$ under $[\mathrm{Par}_{Q_2^3}, \pi_p]$:

$$\begin{tikzcd}
(1,0,0,0,0,0,0,1) \arrow[r] & (1,1,0,0,0,0,1,1) \arrow[d] \\
 & (0,1,1,0,0,1,1,0) \arrow[d] \\
(0,0,0,1,1,0,0,0) \arrow[uu] & (0,0,1,1,1,1,0,0) \arrow[l]
\end{tikzcd} \ .$$

A more lengthy calculation will show that the entire phase space of $[\mathrm{Par}_{K_4}, (1, 2, 3, 4)]$ can be embedded in the phase space of $[\mathrm{Par}_{Q_2^3}, \pi_p]$. The former SDS has one fixed point and three orbits of length 5 while the latter SDS has one fixed point and 51 orbits of length 5.

Accordingly a morphism between two SDS will be a tuple consisting of a corresponding graph morphism between their dependency graphs and a morphism between their phase spaces, therefore explicitly relating their dynamics. Theorem 5.8 will guarantee the existence of such morphisms in some generality.

3.1 Applications

Hopfield Networks. As discussed earlier, our lower bounds for reachability problems for (Bool, AWT)-SDSs directly imply that reachability problems for Hopfield networks with sequential update and asymmetric weights are **PSPACE**-hard. Moreover, the result holds for very small edge weights, bounded degree and bounded pathwidth (and hence treewidth) graphs. To our knowledge, such results have not been reported earlier. Our model of SDS with simple-threshold and simple-inverse-threshold functions and the corresponding **PSPACE**-hard lower bounds for the reachability problems suggest a potentially new variant of Hopfield networks. Finally, for Hopfield networks with 3-simple-threshold functions, we obtain a better upper bound on the length of transients, and hence on the convergence time.

Reference [34] presents a **PSPACE**-hardness result for Hopfield networks with asymmetric edge weights and fully parallel updates. To our knowledge, the result for sequential update was not known earlier. Moreover, discrete dynamical systems with only simple-threshold and simple-inverted-threshold functions have not been considered in the literature to our knowledge. Noting the correspondence between Hopfield networks and the classes of SDS considered here, we get as direct corollaries analogous hardness results for Hopfield networks under sequential updates.

Communicating finite State Machines (CFSMs). CFSMs have been widely studied as models of concurrent processes. As a result, a number of models have been proposed in the literature. Since these models were proposed for different applications, they are not always equivalent. We refer the reader to [3, 28, 40, 44, 46, 71, 82, 84, 94, 97] for definitions, results, applications and the state of current research in this area. The basic setup consists of a collection of finite state machines. These machines communicate with each other via explicit channels [28, 82, 40] or via action symbols [84, 94]. Our results apply to both these variants. To see this, we note the following:

1. Simple-threshold and simple-inverted-threshold functions can be easily represented as finite state machines (FSMs) that essentially emulate a counter. The FSM corresponding to each node of an SDS consists of two parts, namely a control part and a part simulating the threshold function. (For some models, we can sometimes eliminate the control part.)

2. Sequential update of the nodes of an SDS can be simulated by using n distinct (one for each machine) action symbols that in effect imply that each FSM is updated in the order determined from the ordering used for the given SDS. When dealing with explicit channels, this can be done by initializing all the FIFO I/O channels and using the control part to make sure that each machine corresponding to a threshold function makes a transition only after all its inputs have been received. At that point, the transition simply consists of counting how many inputs are 1 and how many of them are 0. After this, the machine posts the result of evaluating the function on each of the output channels.

Given these observations, the remaining details of the simulation are fairly straightforward. Our results show that the **PSPACE**-hardness of reachability problems for CFSMs holds for extremely simple individual automata. Specifically, the automata use a very simple model of concurrency and the underlying graphs are of bounded degree. This points out that a bounded amount of concurrency is sufficient to yield computational intractability results for reachability problems for CFSMs. Thus the results extend some of the earlier results in [94, 84] on the complexity reachability problems for communicating state processes.

The lower bounds can be viewed as a tradeoff between the three basic parameters that characterize communicating finite state processes: (i) the power of individual automata, (ii) the interconnection pattern and (iii) the communication paradigm (e.g. channels, action symbols). For instance, it is easy to prove the **PSPACE**-hardness of reachability problems for a simple chain of communicating automata, where each automaton essentially encodes the transition function of an LBA. On the other hand, we can also show that reachability problems are **PSPACE**-hard, when each individual automaton essentially mimics a simple-threshold or a simple-inverse-threshold function. In either case, the message passing mechanism may be explicit channels or action symbols.

4 Preliminaries

We assume the reader is familiar with basic concepts in computational complexity, algorithms, algebraic graph theory and dynamical systems, otherwise see [1, 26, 30, 38, 80, 99, 90, 101].

A **Sequential Dynamical System** (SDS) $\mathcal{S}$ over a given domain $\mathbb{D}$ of state values is a triple $(Y, \mathcal{F}, \pi)$, whose components are as follows:

1. $Y(V, E)$ is a finite undirected graph without multi-edges or self-loops. Y is referred to as the **underlying graph** or base graph of $\mathcal{S}$. We use n to denote $|V|$ and m to denote $|E|$. The nodes of Y are numbered using the integers 1, 2, ..., n.

2. For each node i of Y, $\mathcal{F}$ specifies a **local transition function**, denoted by f_i. This function maps $\mathbb{D}^{\delta_i+1}$ into $\mathbb{D}$, where δ_i is the degree of node i. Letting $N(i)$ denote the set consisting of node i itself and its neighbors, each parameter of f_i corresponds to a member of $N(i)$.

3. Finally, π is a permutation of $\{1, 2, \ldots, n\}$ specifying the order in which nodes update their states using their local transition functions. Alternatively, π can be envisioned as a total order on the set of nodes.

We set $\delta_i = |N(i)| - 1$, $d(Y) = \max_{1 \le i \le n} \delta_i$. The increasing sequence of elements of $N(i)$ is referred to as

$$\tilde{N}(i) = (j_1, \ldots, i, \ldots, j_{\delta_i}) \ . \qquad (1)$$

Each Y-vertex i has associated a state $x_i \in \mathbb{F}_2$, and for each $k = 1, \ldots, d+1$ we have a symmetric function $f_{(k)} : \mathbb{F}_2^k \to \mathbb{F}_2$. We introduce the map

$$\begin{aligned} \text{proj}[i] : \mathbb{F}_2^n \quad &\to \quad \mathbb{F}_2^{\delta_i+1} \\ (x_1, \ldots, x_n) \quad &\mapsto \quad (x_{j_1}, \ldots, x_i, \ldots, x_{j_{\delta_i}}), \end{aligned}$$

and denote the permutation group over k letters by S_k. For each i there exists a (Y-local) map $F_{i,Y}$ given by

$$\begin{aligned} y_i(x) \quad &= \quad f_{(\delta_i+1)} \circ \text{proj}[i](x), \\ F_{i,Y}(x) \quad &= \quad (x_1, \ldots, x_{i-1}, y_i(x), x_{i+1}, \ldots, x_n). \end{aligned}$$

With this notation, we can alternatively specify a sequential dynamical system as

$$[\mathfrak{F}_Y, \ ] : S_n \longrightarrow \mathbb{F}_{\mathbb{D}}^{n\ \mathbb{F}_{\mathbb{D}}^n}$$

$$[\mathfrak{F}_Y, \pi] = \prod_{i=1}^{n} F_{\pi(i),Y} = F_{\pi(n),Y} \circ \cdots \circ F_{\pi(2),Y} \circ F_{\pi(1),Y} \ .$$

We call $[\mathfrak{F}_Y, \pi]$ the *sequential dynamical system* (SDS) over Y with respect to the ordering π.

A **configuration** $\mathcal{C}$ of $\mathcal{S}$ can be interchangeably regarded as an n-vector $(c_1, c_2, \ldots, c_n)$, where each $c_i \in \mathbb{D}$, $1 \leq i \leq n$, or as a function $\mathcal{C} : V \rightarrow \mathbb{D}$. From the first perspective, c_i is the state value of node i in configuration $\mathcal{C}$, and from the second perspective, $\mathcal{C}(i)$ is the state value of node i in configuration $\mathcal{C}$.

Computationally, each step of an SDS (i.e., the transition from one configuration to another), involves n sub-steps, where the nodes are processed in the *sequential* order specified by permutation π. The "processing" of a node consists of computing the value of the node's local transition function and changing its state to the computed value. The following pseudo code shows the computations involved in one transition.

---

for $i = 1$ **to** n **do**

(i) Node $\pi(i)$ evaluates $f_{\pi(i)}$. (This computation uses the *current* values of the state of $\pi(i)$ and those of the neighbors of $\pi(i)$.) Let x denote the value computed.

(ii) Node $\pi(i)$ sets its state $s_{\pi(i)}$ to x.

end-for

We let $F_{\mathcal{S}}$ denote the **global transition function** associated with $\mathcal{S}$. This function can be viewed either as a function that maps $\mathbb{D}^n$ into $\mathbb{D}^n$ or as a function that maps $\mathbb{D}^V$ into $\mathbb{D}^V$. $F_{\mathcal{S}}$ represents the transitions between configurations, and can therefore be considered as defining the dynamical behavior of the SDS $\mathcal{S}$.

Let $\mathcal{I}$ denote the designated configuration of $\mathcal{S}$ at time 0. Starting with $\mathcal{I}$, the configuration of $\mathcal{S}$ after t steps (for $t \geq 0$) is denoted by $\xi(\mathcal{S}, \mathcal{I}, t)$. Note that $\xi(\mathcal{S}, \mathcal{I}, 0) = \mathcal{I}$ and $\xi(\mathcal{S}, \mathcal{I}, t+1) = F_{\mathcal{S}}(\xi(\mathcal{S}, \mathcal{I}, t))$. Consequently, for all $t \geq 0$, $\xi(\mathcal{S}, \mathcal{I}, t) = {F_{\mathcal{S}}}^t(\mathcal{I})$.

Recall that a configuration $\mathcal{C}$ can be viewed as a function that maps V into $\mathbb{D}$. As a slight extension of this view, we use $\mathcal{C}(W)$ to denote the states of the nodes in $W \subseteq V$.

A **fixed point** of an SDS $\mathcal{S}$ is a configuration $\mathcal{C}$ such that $F_{\mathcal{S}}(\mathcal{C}) = \mathcal{C}$. An SDS $\mathcal{S}$ is said to **cycle** through a (finite) sequence of configurations $\langle \mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_r \rangle$ if $F_{\mathcal{S}}(\mathcal{C}_1) = \mathcal{C}_2$, $F_{\mathcal{S}}(\mathcal{C}_2) = \mathcal{C}_3$, …, $F_{\mathcal{S}}(\mathcal{C}_{r-1}) = \mathcal{C}_r$ and $F_{\mathcal{S}}(\mathcal{C}_r) = \mathcal{C}_1$. A fixed point is a cycle involving only one configuration.

The **phase space** $\mathcal{P}_{\mathcal{S}}$ of an SDS $\mathcal{S}$ is a directed graph defined as follows: There is a node in $\mathcal{P}_{\mathcal{S}}$ for each configuration of $\mathcal{S}$. There is a directed edge from a node representing configuration $\mathcal{C}$ to that representing configuration $\mathcal{C}'$ if $F_{\mathcal{S}}(\mathcal{C}) = \mathcal{C}'$. The phase space of the SDS $\mathcal{S} = [\mathfrak{F}_Y, \pi]$ will be denoted by $\Gamma[\mathfrak{F}_Y, \pi]$. In such a case, we also say that configuration $\mathcal{C}$ is a **predecessor** of configuration $\mathcal{C}'$. Since SDS are deterministic, each node in its phase space has an out degree of 1. In general, the phase space $\mathcal{P}_{\mathcal{S}}$ may have an infinite number of nodes. When the domain $\mathbb{D}$ of state values is finite, the number of nodes in the phase space is $|\mathbb{D}|^n$. Figure 7 shows an example of an SDS and its phase space.

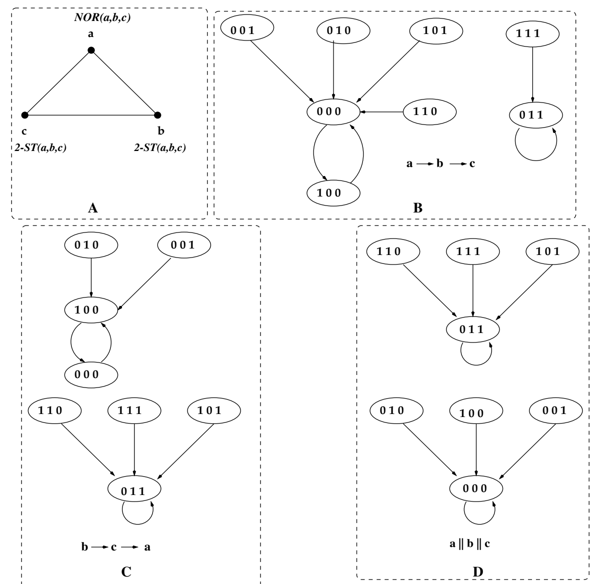

Figure 7: Example of an SDS and its phase space. Figure **A** shows the underlying graph and the local transition functions. Each configuration is specified as a triple (s_a, s_b, s_c). Figure **B** shows the phase space with $\pi = \langle a, b, c \rangle$. Figure **C** shows the phase space with $\pi = \langle b, c, a \rangle$. Finally, Figure **D** shows the phase space when all the nodes are updated in parallel (as in CA).

Fixed points are nodes in phase space with self loops. Cycles with more than one node are called **limit cycles** (or **periodic cycles**) and the nodes on any limit cycle are called **periodic points**. In $\mathcal{P}_{\mathcal{S}}$, a **transient** is a simple directed path such that no edge of the path appears in any cycle in $\mathcal{P}_{\mathcal{S}}$.

Note that a node in the phase space may have multiple predecessors. This means that the time evolution map of an SDS is, in general, *not invertible* but is *contractive*. The existence of configurations with multiple predecessors also implies that certain configurations may have no predecessors. A configuration with no predecessors is referred to as a **Garden-of-Eden** configuration. Such configurations can occur only as initial states and can never be generated during the time evolution of an SDS.

¿From now on, unless otherwise specified, we assume that the domain of a given SDS (or SDS) is finite and that each local transition function can be evaluated in polynomial time.

Monotone and threshold functions are well-studied subclasses of Boolean functions [20, 58]. As SDS under these two classes of local transition functions will be considered throughout this paper, we provide the necessary definitions in this subsection.

Given two Boolean vectors $X = \langle x_1, x_2, \ldots, x_q \rangle$ and $Y = \langle y_1, y_2, \ldots, y_q \rangle$, define the relation "$\leq$" as follows: $X \leq Y$ if $x_i \leq y_i$, $1 \leq i \leq q$. A q-input Boolean function f is **monotone** if $X \leq Y$ implies that $f(X) \leq f(Y)$.

There are several other equivalent definitions of monotone Boolean functions; for example, a Boolean function is monotone if it can be implemented using only operators from the set {And, OR}.

A **k-simple-threshold function** takes on the value 1 if at least k of the Boolean inputs have value 1; otherwise, the value of the function is 0. A **k-simple-inverted-threshold function** takes on the value 0 if at least k of the Boolean inputs have value 1; otherwise, the value of the function is 1.

We use the notation (Bool, ST)-SDS for the class of SDS where each local transition function is a simple-threshold function. When the set of local transition functions of an SDS is allowed to consist of both simple-threshold and simple-inverted-threshold functions, the resulting class of SDS is denoted by (Bool, SIT)-SDS.

Threshold functions, which are a generalization of simple-threshold functions, are defined as follows.

A q-input **threshold function** has q Boolean inputs x_1, x_2, …, x_q with respective weights w_1, w_2, …, w_q, a Boolean output y and a threshold α. The value of y is 1 iff $\sum_{i=1}^{q} w_i x_i \geq \alpha$.

The class of SDS over the Boolean domain where each local transition function is a threshold function is denoted by (Bool, AWT)-SDS. In such SDS, the weights used in the local transition functions at the two nodes of an edge may not be equal. A useful subclass of (Bool, AWT)-SDSs are those in which the weights used in local transition functions are *symmetric*; that is, for each edge, the weights used by the local transition functions at the two nodes of the edge are equal. We denote this subclass of SDS by (Bool, SWT)-SDSs. As will be seen, permitting threshold functions that use weights in an asymmetric manner changes the complexity of reachability problems significantly.

In order to state Theorem 5.1 in the next section we introduce the following notation. Let G be a group and let Y be an undirected graph with automorphism group $\mathrm{Aut}(Y)$. Then G acts on Y if there exists a group homomorphism $u : G \longrightarrow \mathrm{Aut}(Y)$. If G acts on the graph Y, then its action induces (i) the graph $G \setminus Y$ where

$$\mathrm{v}[G \setminus Y] = \{G(i) \mid i \in \mathrm{v}[Y]\} \quad \text{and}$$

$$\mathrm{e}[G \setminus Y] = \{G(\{i,k\}) \mid \{i,k\} \in \mathrm{e}[Y]\},$$

and (ii) the surjective graph morphism π_G given by

$$\pi_G : Y \longrightarrow G \setminus Y, \quad i \mapsto G(i) \ .$$

Furthermore we will study SDS that are induced by the multi-sets $(\mathrm{nor}_{(k)})$ and $(\mathrm{nand}_{(k)})$, where

$$\mathrm{nor}_{(k)}(x_1, \ldots, x_k) = \overline{x_1} \wedge \overline{x_k} \cdots \wedge \overline{x_k} \qquad (2)$$

$$\mathrm{nand}_{(k)}(x_1, \ldots, x_k) = \overline{x_1} \vee \overline{x_k} \cdots \vee \overline{x_k} \qquad (3)$$

We will refer to these SDS as $[\mathrm{Nor}_Y, \pi]$ and $[\mathrm{Nand}_Y, \pi]$, respectively. We finally define basic computational problems that will be considered here.

1. Given an SDS $\mathcal{S}$ over a domain $\mathbb{D}$, two configurations $\mathcal{I}$, $\mathcal{B}$, and a positive integer t, the t-Reachability problem is to decide whether $\mathcal{S}$ starting in configuration $\mathcal{I}$ will reach configuration $\mathcal{B}$ in t or fewer time steps. We assume that t is specified in binary. (If t is specified in unary, it is easy to solve this problem in polynomial time since we can execute $\mathcal{S}$ for t steps and check whether configuration B is reached at some step.)

2. Given an SDS $\mathcal{S}$ over a domain $\mathbb{D}$ and two configurations $\mathcal{I}$, $\mathcal{B}$, the Reachability problem is to decide whether $\mathcal{S}$ starting in configuration $\mathcal{I}$ ever reaches the configuration $\mathcal{B}$. (Clearly, for $t \geq |\mathbb{D}|^n$, t-Reachability is equivalent to Reachability.)

3. Given an SDS $\mathcal{S}$ over a domain $\mathbb{D}$ and a configuration $\mathcal{I}$, the Fixed Point Reachability problem is to decide whether $\mathcal{S}$ starting in configuration $\mathcal{I}$ reaches a fixed point.

4. Given an SDS $\mathcal{S} = (Y(V,E), \mathcal{F}, \pi)$ and a configuration $\mathcal{C}$, the Predecessor existence problem (abbreviated as Pre) is to determine whether there is a configuration $\mathcal{C}'$ such that $F_{\mathcal{S}}(\mathcal{C}') = \mathcal{C}$.

5. Given a partially specified SDS $\mathcal{S}$ consisting of graph $Y(V,E)$, the set $\mathcal{F}$ of symmetric Boolean functions associated with the nodes of G, an initial configuration $\mathcal{C}'$ and a final configuration $\mathcal{C}$, the Permutation existence problem (abbreviated as Pme) is to determine whether there is a permutation π for $\mathcal{S}$ such that $F_{\mathcal{S}}(\mathcal{C}') = \mathcal{C}$.

6. Given an SDS $\mathcal{S} = (Y, \mathcal{F}, \pi)$, the Garden of Eden existence problem (abbreviated as Gee) is to determine whether $\mathcal{S}$ has a GE configuration. The corresponding *counting problem*, abbreviated #GE, is to determine the number of GE configurations of $\mathcal{S}$.

7. Given an SDS $\mathcal{S} = (Y, \mathcal{F}, \pi)$, the Fixed point existence problem (abbreviated as Fpe) is to determine whether $\mathcal{S}$ has a fixed point. The corresponding counting problem, #FP, asks for the number of fixed points in the phase space of $\mathcal{S}$.

Let a configuration $\mathcal{C}$ of an SDS be called a **Strong Garden-of-Eden** (SGE) configuration if it is a Garden-of-Eden configuration under *every* node permutation. Analogous to the Garden of Eden existence problem, the following problem (called the Strong Garden of Eden existence problem, abbreviated as follows: Given the underlying graph $Y(V,E)$ and the set $\mathcal{F}$ of local transition functions of a partially specified SDS $\mathcal{S}$, determine whether $\mathcal{S}$ has an SGE configuration.

5 Scientific Foundations

In this section, we outline our mathematical and computational results on sequential dynamical systems. Many of these results are quite general and as corollaries yield appropriate results for other classes of dynamical systems as well. The primary research objective is to characterize the phase space properties by inspecting the underlying representation (i.e. its intentional form) rather than looking exhaustively at the phase space itself (its extensional form). The results obtained include combinatorial characterizations of phase space, bounds on the various phase space features, e.g. fixed points, Garden-of-Eden states, etc. Section 3 has already outlined the results and discussed some of its implications.

5.1 Equivalence

A permutation $\pi = (i_1, \ldots, i_n)$ induces an orientation $\mathfrak{O}(Y)_\pi$ of Y by setting for $\{i_k, i_r\} \in \mathrm{e}[Y]$ and $k < r$, $o(\{i_k, i_r\}) = i_k$ and $t(\{i_k, i_r\}) = i_r$. By construction $\mathfrak{O}(Y)_\pi$ is acyclic and we have a mapping $w : S_n \to \mathrm{Acyc}(Y)$, $\pi \mapsto \mathfrak{O}(Y)_\pi$. w is surjective and for any $\pi, \sigma \in S_n$, $\mathfrak{O}_\pi = \mathfrak{O}_\sigma$ implies $[\mathfrak{F}_Y, \pi] = [\mathfrak{F}_Y, \sigma]$. Accordingly, we obtain that

$$\begin{aligned} h : \mathrm{Acyc}(Y) &\longrightarrow \{[\mathfrak{F}_Y, \pi] \mid \pi \in S_n\}, \\ \mathfrak{O}_\pi &\mapsto [\mathfrak{F}_Y, \pi] \end{aligned}$$

is well-defined.

One central question in SDS analysis is that of two SDS $[\mathfrak{F}_Y, \pi]$ and $[\mathfrak{F}_Y, \sigma]$ being *equivalent*. Here, equivalence of SDS is defined with respect to a category $\mathfrak{C}[Y, \mathfrak{F}_Y]$ whose objects are the digraphs which are the corresponding phase spaces of the SDS. Here, we will consider the category $\mathfrak{C}_{\mathrm{di}}[Y, \mathfrak{F}_Y]$ having all digraph-morphisms as morphisms and therefore considering two SDS $[\mathfrak{F}_Y, \pi]$ and $[\mathfrak{F}_Y, \pi']$ as to be equivalent if and only if their corresponding digraphs are isomorphic. In the following we will analyze the set of non-equivalent SDS for fixed Y and $\mathfrak{F}_Y$ which we denote by $\mathbf{E}[Y, \mathfrak{F}_Y]$. SDS with different Boolean functions can be equivalent, too: let $[\mathfrak{F}_Y, \pi]$ be an arbitrary SDS and $\mathrm{inv} : \mathbb{F}_2^n \to \mathbb{F}_2^n$, $\mathrm{inv}(x_i) = (\overline{x_i})$ and $\mathfrak{F}_Y^{\mathrm{inv}} = (\mathrm{inv} \circ F_{i,Y} \circ \mathrm{inv})$. Then $[\mathfrak{F}_Y, \pi]$ and $[\mathfrak{F}_Y^{\mathrm{inv}}, \pi]$ are equivalent SDS. In particular, $[\mathrm{Nor}_Y, \pi]$ and $[\mathrm{Nand}_Y, \pi]$ are equivalent.

In our first result we give a combinatorial upper bound on the number of non-equivalent SDS which is known to be sharp for certain classes of SDS. Let $\mathrm{Acyc}(Y)$ denote the set of acyclic orientations of Y and set $a(Y) = |\mathrm{Acyc}(Y)|$.

Theorem 5.1 *[87] Let Y be an arbitrary graph, $\pi \in S_n$ and let $[\mathfrak{F}_Y, \pi]$ be an* SDS *over Y. Setting*

$$\Delta(Y) = \frac{1}{|\mathrm{Aut}(Y)|} \sum_{\gamma \in \mathrm{Aut}(Y)} |a(\langle\gamma\rangle \setminus Y)| \qquad (4)$$

the following assertions hold:

$$\begin{aligned} |\mathbf{E}[Y, \mathfrak{F}_Y]| &\leq \Delta(Y) \\ |\mathbf{E}[\mathrm{Star}_n, \mathrm{Nor}_{\mathrm{Star}_n}]| &= \Delta(\mathrm{Star}_n) = n \ . \end{aligned}$$

5.2 Nor Systems, Preimages, Garden-of-Eden & Fixed Points

Theorem 5.1 rises the following question on SDS induced by nor functions:

For which graphs Y does

$$(\#) \qquad |\mathbf{E}[Y, \mathrm{Nor}_Y]| = \Delta(Y)$$

hold. In fact in [15] additional graph classes are shown to have property (#) and it could be speculated that (#) holds for arbitrary graphs Y. Clearly, in this context, the general structure of SDS of the form $[\mathrm{Nor}_Y, \pi]$ is of central importance. In the following theorem we will provide some insight into these systems.

Let $\mathfrak{I}(Y)$ be the set of Y-independence sets. We will next analyze the structure of SDS that are induced by a multi-set $(f_{(k)})_k$ such that they are fixed-point-free for any graph Y:

Theorem 5.2 *[87] Let $(f_{(m)})_m$ be a family of Boolean, symmetric functions inducing for an arbitrary graph Y the fixed point–free* SDS $[\mathfrak{F}_Y, \pi]$. *Then $[\mathfrak{F}_Y, \pi]$ is equivalent to $[\mathrm{Nor}_Y, \pi]$.*
Suppose $[\mathfrak{F}_Y, \pi]$ is equivalent to $[\mathrm{Nor}_Y, \pi]$, then we have:
(a) each periodic point of $[\mathfrak{F}_Y, \pi]$ corresponds uniquely to a Y-independence set, i.e. there exists a bijective mapping $\iota : \mathrm{Per}[\mathfrak{F}_Y, \pi] \longrightarrow \mathfrak{I}(Y)$.
(b) each vertex in the associated phase space is either periodic or it has in-degree 0. Furthermore, (0) has maximal in-degree in the associated phase space.
(c) Let $Y = \mathrm{Line}_n$ or $Y = \mathrm{Circ}_n$. Then equivalence of two SDS *implies $\lambda((0)_i) = (0)_i$. In particular, the corresponding orbits containing (0) are isomorphic.*
(d) Suppose $\mathrm{Aut}(Y)$ *is transitive and there exist $\rho, \sigma, \pi \in S_n$ such that $[\mathfrak{F}_{\rho(Y)}, \sigma] = [\mathfrak{F}_Y, \pi]$ holds. Then we have $\rho \in \mathrm{Aut}(Y)$ and $\mathfrak{O}(Y)_{\rho^{-1}\sigma} = \mathfrak{O}(Y)_\pi$.*

Using combinatorial result in Theorem 5.2 and additional combinatorial arguments, we can obtain additional results concerning $[\mathrm{Nor}_Y, \pi]$. These results can be viewed as providing a computational analogue of Theorem 5.2.

Theorem 5.3 *Given a $[\mathrm{Nor}_Y, \pi]$, n be the number of nodes of Y. Then*
(a) For each n, $\exists$ $[\mathrm{Nor}_Y, \pi]$, s.t. the phase space of $\Gamma[\mathrm{Nor}_Y, \pi]$ consists of limit cycles of length $\Omega(2^n)$.
(b) An SDS $[F_Y, \pi]$ for which each f_i is drawn from the set { Or, Nor, Nand, And *} has an SGE configuration.*

Next, we focus on the Pre problem and the Gen-pme problem for SDS.

Theorem 5.4 *The* Pre *problem is* **NP***-complete for any of the following restricted classes of SDS $[F_Y, \pi]$: (i) Each f_i (local transition function) is a k-simple-threshold function, for any $k \geq 2$, (ii) each f_i is the exactly-k function for any $k \geq 1$, (iii) each f_i is drawn from the set {* ***Or****,* And *} and (iv) SDS whose underlying graphs are planar.*

Theorem 5.5 *There exists polynomial algorithms for the* Pre *problem for SDS $[F_Y, \pi]$ for which each f_i is a non-empty subset of one of the following sets: {Or,* Nor *}, {*And, Nand *} and {*Xor, Xnor *}.*

There exists polynomial time algorithms for the Gen-pme *problem for SDS $[F_Y, \pi]$ for which each f_i is drawn from the set { Or,* Nor, Nand, And *}.*

These results show an interesting contrast between the complexity of Gen-pme and Pme problems for Nor-SDS and Nand-SDS.

5.3 Reachability

We first consider variants of the reachability problems as defined below.

Using a direct reduction from the acceptance problem for linear bounded automata (LBA), we have the following hardness result.

Theorem 5.6 *[20, 21, 52] The t-*Reachability *problem is* ***PSPACE****-complete for* (Bool, SIT)*-SDSs. The reachability problems remain* ***PSPACE****-complete for SDS $[F_Y, \pi]$ restricted to instances in which* ***either*** *(i) each f_i is either a simple threshold function or a simple threshold function* ***or*** *(ii) each f_i is a weighted threshold but the functions can be asymmetric. Moreover, the results hold even under all of the following restrictions: (a) The maximum node degree in the underlying graph is a constant, (b) The bandwidth (and hence the pathwidth and treewidth) of the underlying graph is a constant, (c) The number of distinct local transition functions used is a constant.*

The above hardness results can be viewed as indicating a trade-off between (i) asymmetry in information exchange between nodes (ii) the types of threshold functions that are sufficient to make the problems computationally intractable and the (iii) interconnectivity structure of the underlying graph. The results should be compared with the work of Buss, Papadimitriou and Tsitsiklis [27] on the complexity of t-Reachability problem for coupled automata.

The results show that, in contrast to the polynomial time solvability of the reachability problem for globally controlled systems of independent automata, a small amount of local interaction suffices to make the reachability problem computationally intractable. Our reduction leads to an interaction graph that is of constant degree, bandwidth bounded and regular. (The interaction graph is obtained from a simple path by replacing individual nodes in the path by groups of nodes that interact only with nodes in neighboring groups.)

In contrast to the **PSPACE**-hardness results for (Bool, AWT)-SDSs, we show the following polynomial time solvability result for reachability problems. These results use the characterization of the phase space in [17, 89] for such systems.

Theorem 5.7 *[20, 21, 52] The t-Reachability, Reachability and Fixed Point Reachability problems for $\mathcal{S}$ $[F_Y, \pi]$ can be solved in polynomial time for the following classes of SDS.*

1. *SDS such that each f_i is symmetric and in which all the node and edge weights are strictly positive. Let $w_{\max}$ denote the largest value among the node and edge weights, and let $w_{\min}$ denote the smallest node weight. If the ratio $w_{\max}/w_{\min}$ is bounded by a polynomial in the size of $\mathcal{S}$.*

2. *SDS in which f_i is a linear function over a unitary semi-ring and*

3. *SDS in which f_i is a* Nor *function and the total number of independent sets of the vertices is polynomial.*

Our main results provide a *dichotomy* between hard to predict and easy to predict classes of SDS. First, the results show that for SDS restricted to symmetric Boolean function are hard to predict while the class of (Bool, SymMon)-SDSs is easy to predict. Similarly, they show that SDS induced by Boolean functions that are either threshold or inverse-threshold functions are hard to predict while SDS restricted to Boolean functions that are just threshold are easy to predict.

Conjecture: [21] Reachability and t-Reachability problems for Nor-SDS are **PSPACE**-hard; i.e. Nor-SDS are "Universal".

The result would be interesting since Nor is a "universal" Boolean function, i.e. any Boolean function can be realized by a circuit made of Nor elements. If the above conjecture is true then Nor based systems would also by "dynamically universal", i.e. discrete dynamical system over finite domain could be simulated using only a combination of Nor local transition functions.

When the domain $\mathbb{D}$ is Boolean and the operators of the unitary semi-ring are Or (+) and And (*), each linear local transition function is either Xor (exclusive or) or Xnor (the complement of exclusive or). Denote this class of SDS by (USRing, Linear)-SDSs. Thus, Theorem 5.7 implies that the Fixed Point Reachability problem for such SDS can be solved efficiently.

The Reachability problem for a $\mathcal{S}$ when each f_i is a linear function over a finite unitary semi-ring can be stated as follows: Given an initial configuration $\mathcal{I}$ and a final configuration $\mathcal{C}$, is there an integer τ such that $\xi(\mathcal{S}, \mathcal{I}, \tau) = \mathcal{C}$? The t-Reachability problem can also be expressed in the same manner with the additional constraint that $\tau \leq t$.

Open Question: What is complexity of Reachability problems for SDS in which each f_i is a linear function over a finite unitary semi-ring?

The successive squaring technique does not seem applicable to these problems since the technique does not compute all the intermediate configurations. The Reachability and t-Reachability problems for (USRing, Linear)-SDSs appear to be closely related to the **discrete logarithm problem** [64].[6]

5.4 Factorization of SDS

The idea behind the factorization of a given SDS is to relate it to SDS considered over simpler graphs. Accordingly, the term "relation" has to be made precise which results in defining what a morphism between SDS is:

Definition 5.1 *Let $[F_Z, \sigma]$ and $[F_Y, \pi]$ be two* SDS. *An* SDS*-morphism between $[F_Z, \sigma]$ and $[F_Y, \pi]$ is a pair (ϕ, Φ) where $\phi : Y \to Z$ is a graph morphism and where $\Phi : \Gamma[F_Z, \sigma] \to \Gamma[F_Y, \pi]$ is digraph morphism.*

Given a graph morphism $\phi : Y \to Z$ we want to relate the dynamics of SDS over the two graphs Y and Z. The local functions will be the same for the two graphs unless otherwise stated. To begin, we relate update schedules for Y and Z via ϕ. Assume $|\mathrm{v}[Y]| = n$ and $|\mathrm{v}[Z]| = m$ and let $\phi^{-1}(i) = \{i_1, \ldots, i_{l_i}\}$ where

[6] The discrete logarithm problem is given positive integers x, y and n to find an e such that $x^e \equiv y (\bmod n)$. As noted in [64], the discrete logarithm problem appears to be computationally very difficult.

$i_1 < \ldots i_{l_i}$ for $1 \le i \le m$. Let $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$, we define the map $\eta_\phi : S_m \to S_n$ by

$$\eta_\phi(\pi) = (\pi_{11}, \ldots, \pi_{1l_{\pi_1}}, \ldots, \pi_{m1}, \ldots, \pi_{ml_{\pi_m}}). \quad (5)$$

For instance, in the example with $\phi : Q_2^3 \to K_4$, we have $\eta_\phi(4,3,2,1) = (3,4,2,5,1,6,0,7)$.

Similarly, we define the map $\tau : \mathbb{F}_2^m \to \mathbb{F}_2^n$ by

$$(\tau(x))_k = x_{\phi(k)}. \quad (6)$$

The dynamics of SDS over Y and Z can now be related in the following way [89]:

Theorem 5.8 *[89] Let Y and Z be loop-free connected graphs, let $\phi : Y \to Z$ be a locally bijective graph morphism, and let $(f_i)_i$ be a fixed sequence of Boolean quasi-symmetric functions. Then the map τ induces a natural embedding*

$$T : \Gamma[F_Z, \pi] \hookrightarrow \Gamma[F_Y, \eta_\phi(\pi)]. \quad (7)$$

Thus, when we have a covering map $\phi : Y \to Z$ we can obtain parts of the phase space of SDS over Y from simpler SDS over Z by the map τ. ***Example 5.*** Let $\sigma = (0,7,1,6,2,5,3,4)$. To illustrate Theorem 5.8 we show how to relate the phase space of the following two SDS $[\mathbf{Min}_{K_4}, \mathrm{id}_4]$ and $[\mathbf{Min}_{Q_2^3}, \sigma]$. We already discussed the covering $\phi : Q_2^3 \longrightarrow K_4$ and observe $\eta_\phi(\mathrm{id}_4) = \sigma$. In [12] we have shown that $[\mathbf{Min}_{K_4}, \mathrm{id}_4]$ has exactly two 5-cycles and no fixed points. The two 5-cycles are shown in the top row of figure 8. For convenience we use the map

$$\xi_i : \mathbb{F}_2^i \longrightarrow \mathbb{N}, \qquad \xi_i(x_1, \ldots, x_i) = \sum_{j=0}^{i} x_j \cdot 2^{j-1}$$

to encode states (binary tuples), and we have, e.g. $(1,1,0,1) \mapsto 1 + 2 + 8 = 11$. It is straightforward to see that the phase space of $[\mathbf{Min}_{K_4}, \mathrm{id}_4]$ is indeed embedded in the phase space of $[\mathbf{Min}_{Q_2^3}, \sigma]$.

We remark that $[\mathbf{Min}_{Q_2^3}, \eta_\phi(\mathrm{id}_4)]$ has two fixed points in addition to the two 5-cycles shown in the last row in Figure 8. These fixed points are related by the graph automorphism $\gamma = (07)(16)(25)(34)$, and consequently, so are their transients. Stated differently, the two components in $\Gamma[\mathbf{Min}_{Q_2^3}, \eta_\phi(\mathrm{id}_4)]$ containing the fixed points are isomorphic. Their structure is shown in figure 9.

It will usually be more feasible to analyze the SDS over the smaller graph. A particularly interesting instance of this situation is the following:

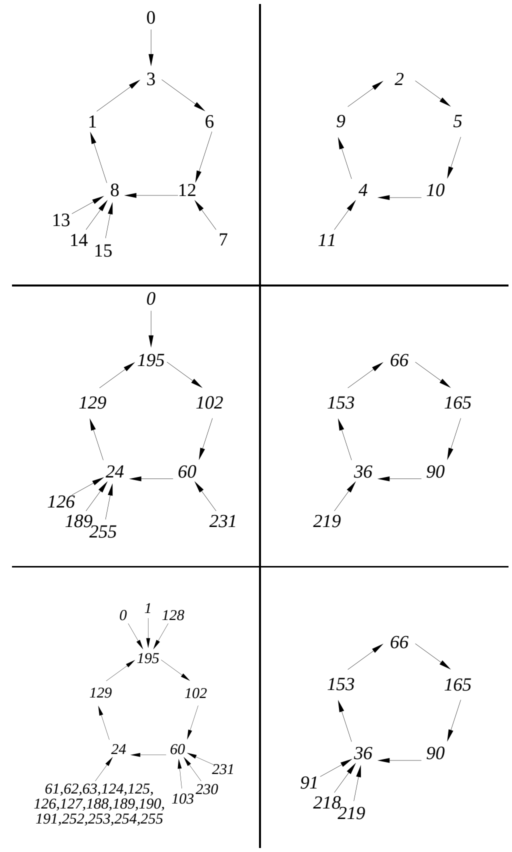

Figure 8: The top row shows the two five-cycles in $[\mathbf{Min}_{K_4}, \mathrm{id}]$. The second row shows the images of the top cycles under τ_ϕ, and the last row shows the corresponding periodic cycles in the digraph $\Gamma[\mathbf{Min}_{Q_2^3}, \eta_\phi(\mathrm{id}_4)]$.

Proposition 5.1 *Assume $2^n \equiv 0 \bmod n+1$, and let $\pi \in S_{n+1}$. Then the* SDS $[\mathbf{Par}_{Q_2^n}, \eta_\phi(\pi)]$ *has a periodic orbit of length $n+2$.*

This situation is very advantageous since SDS over the complete graph are usually a lot simpler to analyze than SDS over any other graphs.

6 Engineering: A Theory Based Approach

A primary and hard requirement in building these simulations was the size and scope of the simulations. Specifically, each simulation has over 10^6 entities and covers large geographical areas on the order of a medium sized metropolis. For instance, the network simulator is aimed at simulating up to 10^9 transceivers and 10^{12} packets per hour. Each simulation is devel-

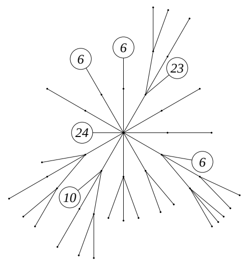

Figure 9: The structure of the components in $\Gamma[\mathbf{Min}_{Q_2^3}, \eta_\phi(\mathrm{id}_4)]$ containing a fixed point. A single filled circle depicts a single state, while a circled number i depicts that there are i direct predecessors that do not have any predecessors themselves.

oped and demonstrated (or is being demonstrated) on real large infrastructures rather than merely synthetic data. This involves interesting issues of data collection and validation. The size, scope and the intended purpose of the simulations impose a strong set of engineering requirements:

1. high performance parallel/distributed simulation design
2. parametric/normative yet approximate representations of individual agents
3. Algorithmic Semantics: Correctness of parametric representations as producing the required dynamics
4. lightweight agent abstraction and representation

Due to lack of space, the last requirement will be discussed in a companion paper.

Example 6: **Updating Driver States.** In this example, we investigate the relation between the structure of the dependency graph of a system and its influence on the corresponding dynamical system properties. We choose an example from **TRANSIMS**, wherein one updates the vehicle positions based on the vehicles in the neighborhood. For more references on traffic studies we refer to [7, 10, 14, 75, 76, 77]. The setup is the following. There is a circular one-lane road divided into n cells as in figure 10.

One vehicle occupies one cell and has a given velocity. A vehicle can travel at one of three velocities: 0, 1 and 2. There are m vehicles and their initial positions are chosen at random. They are labeled 1 through m by the order in which they initially appear on the road. There is a schedule π that determines

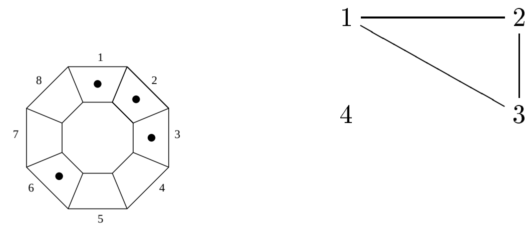

Figure 10: A circular one-lane road divided into cells. A dot indicates that the given cell is occupied by a vehicle. The dependency graph $Y(t = 0)$ associated to configuration to the left is shown to the right.

the update ordering. A vehicle at cell i with speed v is updated as shown in Table 1.

(Cell,Speed)	$i+1$ taken	$i+1$ free, $i+2$ taken	$i+1$, $i+2$ free
$(i,0)$	$(i,0)$	$(i,1)$	$(i,2)$
$(i,1)$	$(i,0)$	$(i+1,1)$	$(i+1,2)$
$(i,2)$	$(i,0)$	$(i+1,1)$	$(i+2,2)$

Table 1: The update rule for a single vehicle

Thus a vehicle at cell i with speed 1 that has two free cells ahead moves one cell ahead and gets the new speed of 2.

At each time step t we can derive the associated dependency graph $Y(t)$. The graph $Y(t)$ has vertices $1, 2, \ldots, m$ corresponding to the vehicles. Two vehicles k and l are connected by an edge if the distance between them at time t is less than or equal to $v_{\max} = 2$. If the distance is larger they are independent by construction. (A vehicle only depends on what is ahead on the road.) Thus for the configuration in Figure 10 we derive the dependency graph shown in Figure 10.

There are two ways to display the occupancy pattern of the cells. One way would be to give phase space diagrams. Here it is more convenient to display orbits, frequently referred to as space time diagrams. Each horizontal line shows the occupancy pattern for a given time. A black square means that the particular cell is occupied at the given time, while a white square means that it is unoccupied at the given time. The initial configuration is displayed on the bottom, its subsequent state next, and so on.

Clearly, the space time development will depend on the order that is used to update the car. This is illustrated in Figure 11 where the same initial occupancy pattern is studied under natural update order $(1, 2, \ldots, m)$ and even-odd update order

$(2, 4, \ldots, 1, 3, \ldots)$ respectively.

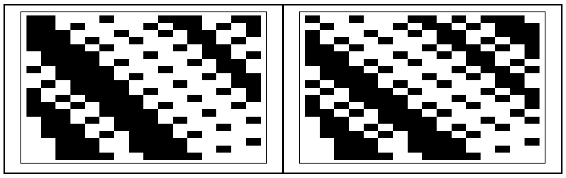

Figure 11: The left diagram shows the evolution for the natural update order and the left diagram shows the evolution for the even-odd update order.

The traffic pattern evolution as a function of the density $\rho = m/n$ exhibits a threshold value for congestion. In Figure 12 we show evolution patterns for densities below, at and above the threshold together with a snapshot of one of the corresponding dependency graphs $Y(t_0)$. In this example there are 16 cells. The threshold occurs between 4 and 5 vehicles.

The existence of a density threshold for the traffic evolution is natural. When the road is occupied with a small number of vehicles they will eventually spread out and travel independently of each other at maximum speed. The dependency graph approaches the empty graph $Y_\emptyset$ independently of initial distribution. As more vehicles are put into the system one cannot avoid overlapping dependency regions. This can for instance be observed using the average speed. For low densities the average speed approaches 2. For higher densities it fluctuates in the range $(0, 2)$. Thus there is an interesting interplay between the dependency graph evolution and the corresponding dynamical system properties.

6.1 Disaggregated Normative Agents: PARALEL Algorithms

A key component of our research paradigm that specifically addresses scaling issues is the notion of PARameterized Approximate Local and Efficient aLgorithms (**PARALEL**) discussed below. As discussed above, in simulating large systems with tens of thousands (or more) of interacting elements, i.e., interactors, it is computationally infeasible to explicitly represent each entity in virtual detail using, perhaps, naive one agent-one encapsulated software object representational ideas. A common method of simulating such systems is to use parameterized representations of entities. The goal is to capture different behaviors of the system using different sets of parameters. The concept corresponds to having a normative representation of each abstract agent. A parameterized representation allows efficient use of computational

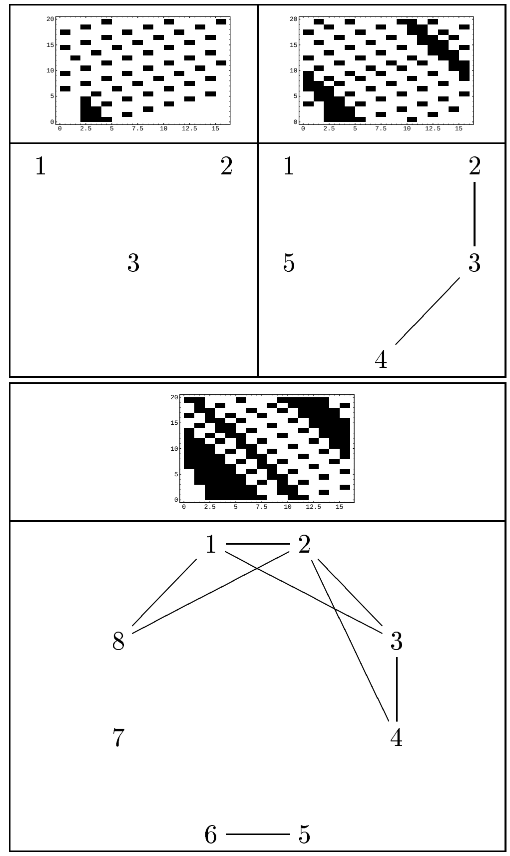

Figure 12: The diagram in the upper left shows the evolution below the density threshold while the upper right diagram and the lower diagram show the evolution at and above the threshold respectively. The graphs under the diagrams show snapshots of the associated dependency structure.

resources. Indeed, even in systems with only tens of thousands of entities, the set of potential interactions among the entities is so large that parameterized representations are desirable, if not absolutely necessary to simulate the interactions in an efficient manner. The basic ideas behind agent abstraction are found in our concept of **PARALEL** algorithms:

- **PAR**ameterized, in that a single basic algorithm with a correct set of input parameters is capable of representing a class of algorithms,
- **A**pproximate, in that their behavior closely approximates an exact algorithm achieving a given task,
- **L**ocal, in that the information required by such algorithms is local as opposed to global, and
- **E**fficient, in that are very fast and can be executed efficiently on both sequential and dis-

tributed shared memory multiprocessor architectures a-**L**-gorithms.

The concept of local algorithms is akin to the recently independently introduced concept of decentralized algorithms [57] and also to the classical concept of distributed algorithms. The approximate behavior is also pertinent at two levels. At the basic level an approximate algorithm closely models the behavior of each physical entity. At a global level, an approximate solution implies that the composed local algorithms representing each agent along with the update mechanism approximates the global system dynamics. The second notion of approximation is more important, although the first notion cannot be completely ignored. We discuss three examples of such classes of algorithms in two contexts: (i) Representation of a network protocol stack (ii) modeling the driving logic of a driver [7, 75, 77] and (iii) Finding minimum cost paths in a multi-modal transportation network [18, 14]. In addition similar generic methods for load balancing and actuated signal timings have been used in our parameterized representation of actuated signals in TRANSIMS [14].

Example 7: **Parametric Driver in TRANSIMS.** As another example of a parametric algorithm (or normative agent), we describe the simple logic used for representing a driver in TRANSIMS [7, 75, 77, 76]. As in Example 1, we will only consider one lane road[7]. Here, a road is divided into cells of 7.5 meters and the variable *gap* is used to measure the number of empty cells between a car and the car ahead of it. In the following, let v denote the speed of the vehicles in number of cells per unit time, $v_{\max}$ denote the maximum speed and *rand* as a random number between 0 and 1. Finally, p_{noise} denotes the probability with which a vehicle is slowed by 1 unit. Each iteration consists of the following 3 sequential rules that are applied in parallel to all the cars:

1. Acceleration of free vehicles: **If** $v < v_{max}$, **Then** $v = v + 1$.

2. Braking due to cars in front: **If** $v > gap$, **Then** $v = gap$.

3. Stochastic Jitter: **If** $(v > 0)$ **AND** $(rand < p_{noise})$, **Then** $v = v - 1$.

In spite of their simplicity, these rules produce fairly realistic traffic flow characteristics and can in limit approach the fluid dynamics models studied in traffic flow theory [7, 75, 77, 76]. Figure 13 shows illustration of traffic flow characteristics produced by the above set of rules for a one-lane road with periodic boundary conditions.

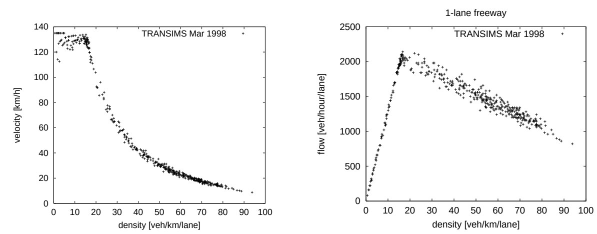

Figure 13: Figures representing various traffic flow characteristics.

Example 8: **Parametric Protocol Stack.** Here we describe our abstraction for modeling each transceiver. See [49, 50] Each transceiver of an ad-hoc packet switched network can be idealized as a computer along with the necessary hardware for network communication. We will use communication network terminology and abstractions as our starting point. Our abstraction will thus span the *transport layer, network layer, data link layer* and the *channel access layer.* One approximate representation for the entire protocol stack would then look like:

$$P = \exp\left[-\beta \, \frac{\Delta_1}{\Delta_2} \, f(t) \, g(h) \, \nu\right] \tag{8}$$

Here P denotes the probability of retransmission of a packet, Δ_1 and Δ_2 are the distance proxies to the destination and from the previous transceiver respectively, $f(t)$ is an arbitrary function of time since the packet was launched from the source, and $g(h)$ is a function of the number of hops the packet has taken from the source. ν is a function of the available resources; it will be 1 if resources for transmission remain, and infinite otherwise. Before considering the calibration of β, we must fix the functional forms of $f(t)$ and $g(h)$. As shown in [16], *simple* variations in the order of evaluations or changes in the can be used to mimic various protocols at different levels.

6.2 Algorithmic Semantics: Correctness of PARALEL

Given a specification of an individual agent, a fundamental question is to decide if this specification (algorithm) serves as an "approximation" of the original agent. Ideally, what one would like is to seek a "proof" that the simulation as a whole serves as an approximation for the simulated system. A detailed discussion of this issue is beyond the scope of this paper and is very much an *open research question.* We

[7]The models has been extended to multi-lane systems also

offer a few possible approaches to tackle the problem. These approaches include:

- Notion of Simulation, Bisimulation, Trace equivalence and other equivalence relations used in distributed computing [52, 44, 46, 71, 82, 84, 94].
- Notion of exact and approximate distributed algorithms and the associated proof techniques that a given distributed algorithm simulates a known sequential algorithm [65, 85].
- Uses of hierarchical abstractions and associated methods such as hiding to show that one system can approximately simulate the other at a certain level of granularity [2, 3, 52, 71, 82, 94, 97].
- Using game theoretic constructs and arguments to show that the dynamic process being studied is an Equilibrium of a certain game [55, 79, 81, 93][8].

7 Brief Overview of Projects

In this section, we will briefly describe our ongoing simulation effort for number of socio-technical systems. The systems described include:

- **TRANSIMS:** a transportation analysis tool that also provides detailed mobility patterns
- **adhopNET:** a simulation based analysis tool for simulation hybrid packet switched telecommunication networks,
- **EpiSIMS:** a simulation for studying the propagation and transmission of disease and
- **Market-Simulation:** a simulation to study deregulated power markets and commodity markets in general.

See [100, 6, 31] for other efforts on building simulations for these infrastructures. These socio-technical systems represent the basic national infrastructures. The design and implementation of the simulations draws on the SDS based theory of simulations; examples of this have already been given in the previous section. The systems are at various stages of development – **TRANSIMS** being most advanced (is in fact being commercialized) and Market Simulation being only a couple of years old. As expected mobility patterns and activities produced as a part of **TRANSIMS** form the basis of all the other simulations. Figure 14 shows the current software inter-dependence. As would be expected, these simulations can be used to perform a range of analysis. For example **TRANSIMS** can be used to design land use models, study the cost benefit analysis of building transportation infrastructure such as freeways, mass transit systems, etc. It can also be used to study emissions and their environmental impact. All these studies can be carried out at a regional scale.

[8] As an example, Kelly [55] shows a result in this direction by showing a game whose Nash equilibrium is the TCP/IP congestion control.

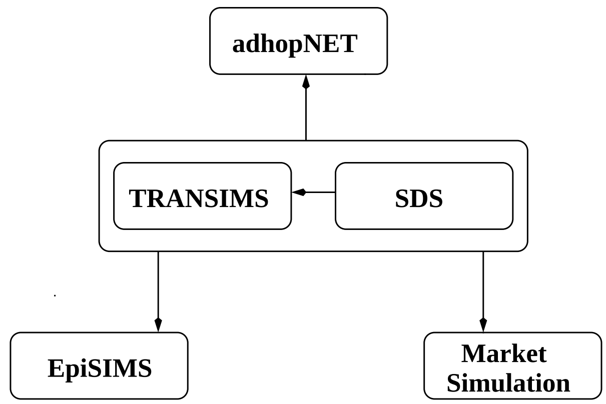

Figure 14: Various Infrastructure Simulations and their inter-dependence.

Building these infrastructures also allows us to study infrastructure inter-dependencies. Such studies can be carried out from the stand point of vulnerabilities, economic impact and cost benefit analysis across infrastructures. For example, one would like to study the impact of power loss to a particular community such as say the Wall Street to the commodity trading. Another example is to see how disruption in communication will strain the transportation infrastructure. The number of possible uses are indeed numerous; the aim was just to illustrate the range of possibilities when one couples the individual simulations.

7.1 TRANSIMS

Transportation Analysis and Simulation System (**TRANSIMS**) developed at the Los Alamos National Laboratory produces estimates of the social network in a large urban area based on the assumption that the transportation infrastructure constrains people's choices about what activities to perform and where to perform them. The purpose of **TRANSIMS** is to develop new models and methods for studying transportation planning questions. See Figure 15. A prototypical question considered in this context would be to study the economic and social impact of building a new freeway in a large metropolitan area. We refer the reader to [8] and the web-site `http://transims.tsasa.lanl.gov` to obtain extensive details about the **TRANSIMS** project. **TRANSIMS** conceptually decomposes the transportation planning task into three time scales. First, a large time-scale associated with land use and demographic distribution as a characterization of travelers. In this phase, demographic information is used to create *activities* for travelers. Activity information typically consists of requests that travelers be at a certain location at a specified time. They include information on travel modes available to the traveler. A synthetic population is endowed with demographics matching the joint distributions given in census data. Observations are made on the daily activity patterns of several thousand households (survey data). These patterns are used as templates and associated with synthetic households with similar demographics. The locations at which activities are carried out are estimated taking into account observed land use patterns, travel times, and dollar costs of transportation. Second, an intermediate time-scale consists of planning routes and trip-chains to satisfy the activity requests. This module find minimum cost paths through the transportation infrastructure consistent with constraints on mode choice. An example constraint might be: "walk to a transit stop, take transit to work using no more than 2 transfers and no more than 1 bus". Finally, a very short time-scale is associated with the actual execution of trip plans in the network. This is done by a simulation that moves cellular automata corresponding to the travelers through a very detailed representation of the urban transportation network. The simulation resolves traffic down to 7.5 meters and times down to 1 second. It provides an updated estimate of link costs, including the effects of congestion, to the Router and location estimation algorithms, which produce new plans. This feedback process continues iteratively until convergence to a steady state in which no one can find a better path in the context of everyone else's decisions. The resulting traffic patterns are matched to observed traffic.

The **TRANSIMS** system can be viewed as composition of SDS. Each SDS corresponds to one of the modules as described above. The traffic simulation is an obvious one, but the population disaggregation and activity generation module and the multi-modal router can also be seen as SDS. The system consists of about 200,000 lines of mostly C++ code and runs on a variety of UNIX platforms. A case study in Portland, Oregon is currently underway. The Portland population located on this network is about 650 000 households with approximately 1.8 million travelers who participate in 8.9 million activities during the course of a 24 hour period. The current case study is being performed on a Linux cluster using 48 CPUs for the traffic simulator and 64 for the router. The router takes roughly 20 hours to generate trips for the entire population; the traffic micro-simulator runs about 3 times faster than real-time, on average. At the current time, no information is available regarding scalability. The current number of processors was chosen to keep each process's memory requirement comfortably below 1 Gigabyte.

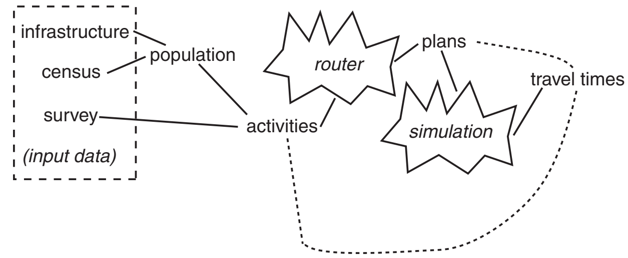

Figure 15: Data flow in the **TRANSIMS** simulation system, proceeding from left to right. Input data comes from the U.S. census and metropolitan planning organizations. We generate a synthetic population whose demographics match the census; give each household an appropriate set of activities; plan routes through the network; and estimate the resulting travel times. The dotted lines represent feedback pathways, along which data flows from right to left, in the system.

7.2 adhopNET

adhopNET is a simulation based analysis tool for an end-to-end simulation of packet switched communication systems. By end-to-end simulation system, we mean that the entire aspects of a mobile communication system is represented and studied. This ranges from detailed mobility calculations to packet level dynamic generation to computation of the performance measures characterizing the system performance. Initially development of the simulator has focused on adhoc networks (networks with no fixed infrastructure); but it will be clear that a successful implementation of this would directly yield solutions for hybrid packet switched networks. The prototype version has been tested on systems with millions of transceivers with billions of packets moving in the system. We view the underlying system as a large discrete dynamical system and use SDS to specify these systems mathematically. This view point allows a certain level of rigor in engineering these simulations. In the remainder of the section, we briefly outline the SDS specification of packet switched ad-hoc radio networks.

The function of *Mobility Data Generation Module*

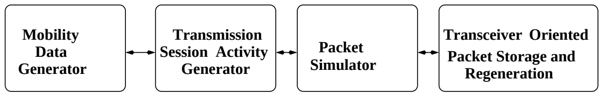

Figure 16: Overall design of **adhopNET**.

(MDG) is to generate the positions of transceivers at various times of the coarse simulation clock. This module also allows transceivers to become idle for some period of time and to rejoin the network at a later time. The module also provides for new transceivers to join the network and existing transceivers to leave the network permanently. The module allows us to use synthetic mobility models such as the ECRM model as well as TRANSIMS based realistic mobility patterns. *Transmission Session Activity Generation Module* generates sessions for a fraction of the transceivers that are active at a given coarse clock time. For each session, the module specifies the source, the destination, and the number of packets in the session, the size of each packet and the rate at which the source generates the packets. The third module is the *Packet Simulator*. This module simulates the packet flow in the system using the activity data and the graph structure generated by the mobility generation module and transmission session activity module. The module is responsible for producing the packet movements and the resulting dynamics. This is done using the **SORSRER** algorithm, a parameterized abstraction discussed in the previous Section. The *Transceiver-Level Packet Data Collection and regeneration* module is concerned with efficient methods for collecting summary information about packets and packet sequences as they arrive at a destination. Information is collected per activity at the destination node. The data collected relates to index and time distortion of a packet sequence. We focus on information about the network dynamics that is embedded in the distortion of the original sequential order of packets in a session during the rebroadcast routing process that is observed at the destination site. We then use Monte-Carlo Regenerative methods to efficiently regenerate synthetic packet sequences. These synthetic sequences are statistically indistinguishable from the packet sequences collected during the session generation phase, much more efficiently produced, but pertain only to the particular system configuration simulated. But it replaces enormous data storage requirements for detailed system analysis with much more modest storage and a certain amount of additional computation. The mapping of the generation-oriented simulation (SDS) into the re-generation method (an isomorphic SDS) is theoretically under-written by equivalence properties of SDS.

The telecommunication problem consists of modeling and analyzing a general setup of mobile transceivers. The transceivers are located and move in some appropriate two- or three-dimensional physical space. Stationary or fixed networks can be viewed as a special case.

We describe an SDS based specification of a generic ad-hoc communication network. It will be clear that the ideas can be extended to hybrid networks also. See [25] for additional details. Our modeling and mathematical framework for IP network must include quantities describing, e.g. i) mobility and position of stations or transceivers, ii) the packet structure used in broadcasting, iii) metrics or distance proxies for routing, and iv) measures for transmission quality.

The transceivers or stations pass messages from message origins to message destinations. A transceiver or station could be, e.g. a base station, a communication satellite, etc. Stations send and receive packets, and also act as intermediate communication links for other stations.

Transceivers have a certain broadcast power which is reflected in its broadcast radius r, or more generally, the extent of its domain of influence, $\mathrm{dom}_t(i)$. We will use Y_t for referring to the dependency structure induced by the broadcast radii, or more generally, domains of influence of the transceivers. The graph reflects which transceivers can communicate. Since, e.g. the transceivers are moving the dependency graph will change with time.

There are four layers for the process of packet and sequence transmission. There is the physical layer and the MAC layer which are the basic layers. Packet routing and end-to-end connection management are the other two layers. Each layer can be cast or represented as an SDS, and each of these SDS are defined over the same set of vertices. We will demonstrate how this can be done for the Physical layer and the MAC layer. We will denote the SDS obtained for the physical layer by $F_1(v, \pi_1)$. Similarly, we denote the SDS obtained for the MAC layer, packet routing and end-to-end connection management by $F_2(v, \pi_2)$, $G(v, \pi_3)$ and $H(v, \pi_4)$, respectively. Here v denotes the vertex set. It is included to emphasize that the vertex set is the same for all four SDS. The full update of the system will in general be of the form:

$$H(v, \pi_4) \circ G(v, \pi_3) \circ F_2(v, \pi_2) \circ F_1(v, \pi_1)$$

Physical layer. The domain of influence, or the broadcast range of a transceiver, is determined by its power level. The relation "who-can-reach-who" can be encoded in the dependency graph Y_t. We note that the graph Y_t is induced by the motion of the transceivers or stations.

The edges around each vertex or transceiver can be ordered or labeled 1 through $m-1$. We take the complete graph on m vertices as base graph, and we assign to each transceiver a state $x \in \mathbb{F}_2^{m-1}$.

The local update function for vertex i is defined by

$$f_i : [\mathbb{F}_2^{m-1}]^m \to \mathbb{F}_2^{m-1},$$
$$(f_i(x))_k = \begin{cases} 1, & \mathrm{pos}(\tau(e_k^i)) \in \mathrm{dom}_t(i) \\ 0, & \text{otherwise}, \end{cases}$$

where $\tau(e_k^i)$ denotes the transceiver adjoined to vertex i through edge e_k^i.
Clearly, the choice of update schedule does not matter.

The SDS over K_m induced by the local functions above is $F_1(v, \pi_1) : [\mathbb{F}_2^{m-1}]^m \to [\mathbb{F}_2^{m-1}]^m$, and the state x in

$$\begin{aligned} x &= ((x_1^1, x_2^1, \dots, x_{m-1}^1), \dots, (x_1^m, x_2^m, \dots, x_{m-1}^m)) \\ &= F_1(v, \pi_1)((0,0,\dots,0), \dots, (0,0,\dots,0)), \end{aligned}$$

is an encoding of the dependency graph Y_t. If $x_k^i = 1$ then the kth edge from vertex i in K_m will be an edge in Y_t, and conversely if $x_k^i = 0$.

Alternatively, but equivalently, we can formulate this in terms of the dependency graph itself and the transceiver displacements. That is, from the current dependency graph Y_t and the displacement $\Delta X_t = (\Delta X_{t,1}, \dots, \Delta X_{t,m})$ we obtain Y_{t+1}:

$$(Y_t, \Delta X_t) \overset{\phi}{\mapsto} Y_{t+1}$$

This may be advantageous in the situation where the transceiver motion is de-coupled from the dynamics of packet broadcast and also when mobility or displacement is generated according to some stochastic process.

MAC layer. The dependency graph provides information on what kind of transceiver-transceiver communication that in theory could occur. However, for packet broadcast to actually take place one needs to have allocated transmission frequencies or channels.

Clearly, two transceivers that are adjacent cannot operate on the same frequency since that may result in corrupted packet transmission. Similarly, two transceivers that are not adjacent but have a common neighbor transceiver cannot send on the same frequency either.

The frequency or channel assignment problem thus corresponds to the distance-2 vertex coloring problem of the graph Y_t [59]. Let d_Y denote the usual path length metric of a graph Y.

We claim that coloring by any local algorithm can be implemented as an SDS. As an example we take coloring by the greedy algorithm. In this particular case we go through the vertices in some order and assign to each vertex the smallest (by some ordering) color that is available for the neighborhood of that vertex. To be explicit, let be a graph on m vertices and let $\pi_2 \in S_m$. Take as the set of colors $C = \{0, c_1, \dots, c_n\}$ with ordering $0 < c_1 \dots < c_n$ and define

$$f_k : C^k \to C,$$
$$f_k(x_1, \dots, x_k) = \min_{c \in C}\{c \mid c > x_l \text{ for } l = 1, \dots, k\}.$$

We denote the SDS over Y induced by the local functions above by $F_2(v, \pi_2) : C^m \to C^m$. The coloring assigned by the greedy algorithm is

$$(x_1, \dots, x_m) = F_2(v, \pi_2).$$

A distance-2 coloring of a graph Y can be obtained as a coloring of the graph Y^2 defined by

$$\mathrm{v}[Y^2] = \mathrm{v}[Y]$$
$$\mathrm{e}[Y^2] = \{\{i,j\} \mid d_Y(i,j) \le 2\}.$$

In the actual implementation edges in Y_t may have to be deleted in case there is a shortage of available frequencies. This can be incorporated into $F_2(v, \pi_2)$ with no change in structure. It does, however, impose update ordering dependencies.

Clearly, for the generation of the covering graph it is advantageous for the transceivers to have a high power level for broadcasting. By this, longer packet jumps can be obtained. For the frequency allocation the opposite is true: Few neighbors, which equates to a smaller broadcast power level makes frequency allocation simpler. The power level must be attuned in accord with mobility and packet traffic.

7.3 EpiSIMS

As another example, we describe the design and implementation of a system for simulating the spread of disease among individuals in a large urban population over the course of several weeks. See [32] and `http:www.lanl.gov/orgs/d/d2` for a more detailed account of the project. In contrast to traditional approaches, we do not assume uniform mixing among large sub-populations or split the population into spatial or demographic subpopulations determined *a priori*. Instead, we rely on empirical estimates of the social network, or contact patterns, that are produced by **TRANSIMS**. The design we have chosen is a distributed discrete event simulation. Data flow in the simulation is sketched in the block diagram of Figure 17 and described in more detail below. Each individual is represented by an object that contains a subset of the available demographic information as well as his or her state of health. Each computational node is responsible for all the interactions at a subset of (geographical) locations. Individuals are passed among computational nodes via messages as they go about their daily activities.

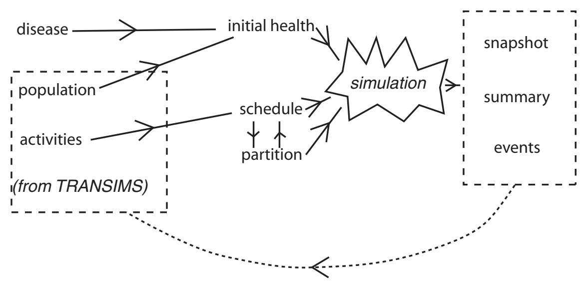

Figure 17: Data flow in the epidemiology simulation system. Input data comes from two sources: the user's disease model and information about the social network. Stand-alone tools operate on the disease model and the population's demographics to produce the initial state of health for everyone in the simulation. Another tool converts a list of activities and locations organized by person into a schedule of events (primarily arrivals and departures) organized by location. The final preparation step estimates an optimal partition of resources among computational nodes. The simulation itself executes events in strict time order and propagates disease in accordance with the user's disease model. It produces three kinds of output: snapshots at specified intervals for animation, statistical summaries of the simulation, and sets of disease-related events. In the near future, we will add the feedback pathway shown by the dotted line, allowing a person's state of health to affect his or her activities.

7.4 Deregulated Commodity Market Simulation

As a final example we briefly describe our ongoing effort to build a highly scalable, individual based, microscopic coupled simulation tool for analyzing a deregulated electricity market [24]. The aim is to build a system that can in principle simulate the entire North American power market and very specific design decisions have been taken into account to meet these long term requirements. The basic design is quite geneic and can be used for simulating other commodity markets as well. A particularly compelling situation is the bandwidth trading market for trading capacity on the Internet.

The overall design of our simulation based analysis tool is depicted schematically in Figure 18. The system is individual based and uses a bottom up method for generating power consumption usage patterns to drive the market and the physical grid. It consists of three main components that form a coupled system:

1. the electrical power grid, with associated elements including, generators, substations, transmission grid and their related electrical characteristics.
2. a market consisting of usual market entities: buyers, sellers, the power exchange (where electricity trades are carried out at various time/size scales), ISO; the market clearing rules and strategies.
3. a **TRANSIMS** based individual power demand creator that yields spatio-temporal distribution of the power consumed.

The simulation can be used as an analysis tool by the government, policy makers, regulators, generators and politicians, with which they could predict and foresee changes in the policies that they propose and any actions they potentially take. It will have the ability to produce results that follow from different regulatory changes, see the impact of changes in consumer behavior on the clearing price, impact of price caps on demand and supply, market efficiency, generators' bidding strategies etc. Also, how different market clearing rules result in different clearing prices, which clearing rules favor which class of generator, consumers and bring most efficiency to the system? Who benefits from the asymmetric information between consumers and generators, can market efficiency be improved by providing more instantaneous and complete information to the players or will it be misused to exploit market power? These are a few of the important questions that can be answered using this simulation.

The simulation uses a parametric representation for a buyer as well as a seller. Other unconventional design features include: (i) Drops of classical Cournot oligopolist's assumptions, (ii) Assumes bounded rationality, (iii) Individualistic and detailed modeling of the consumption function, (iv) Mobility driven consumption depends on activity, location, time and demographics, (v) Allows asymmetric information between consumers and generators, (vi) Generators aim to maximize not only profits but also market share.

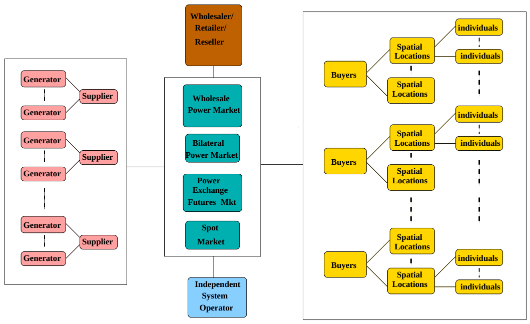

Figure 18: Activity and Location Based Power Consumption

8 Conclusions and Future Work

We have discussed mathematical foundations and engineering principles necessary for building large scale simulations of socio-technical systems. On the theoretical side, sequential dynamical systems are proposed as a mathematical model of large discrete simulations. Engineering principles are derived from such a theory. These engineering principles allow us to design simulations for extremely large systems and implement them on massively parallel architectures. Our ideas are illustrated by means of several on going socio-technical simulations being developed at Los Alamos National Laboratory. We conclude this paper by mentioning recent extensions to the SDS framework.

Time Evolving SDS: First, we want to extend the SDS framework where the underlying dependency graph, functions and order of update changes over time. The current simulation approach incorporates what we already know about evolving dependency structures and handles them explicitly. The need for time evolving SDS becomes clear when we note that the dependency structure of the underlying agent changes due to the inherent mobility present in the system under consideration. See [53, 67, 68] for recent work in this direction.

Coupled SDS: Second, we are extending the SDS framework to apply to so called *coupled/composed sequential dynamical systems*. The description of SDS based network specification in the previous section and in [25] are already a step in this direction. We have also a graph grammar based formalism that allows us to specify and investigate the properties of such systems [53, 67, 68].

Stochastic SDS. The third extension is to incorporate stochasticity in the SDS formalism. A moment of thought will reveal that this feature can be added to various attributes comprising SDS (the dependency graph, local transition function, states and ordering). We have begun preliminary investigations concerning these extensions in [54, 86, 88].

Algebraic Invariants of SDS. For the categorization of SDS (and their phase spaces in particular) a generic framework, like a homology theory for topological spaces, would be of great benefit. First steps towards such a framework have been made in [61, 89] where the fixed points of an SDS have been characterized as certain cohomology groups. This approach immediately allows for an efficient computation of fixed points of SDS on a local basis [62].

Acknowledgements: This research has also been funded in part by the LDRD-DR project *Foundations of Simulation Science* and by the LDRD-ER projects *Extremal Optimization* and *Advanced Markov Chain Monte Carlo Methods* at the Los Alamos National Laboratory. We thank Dr. Ernie Page and Professor David Nicol (Dartmouth College) for inviting us to submit the paper. We sincerely thank our co-authors, external collaborators and the team members of the various simulation projects discussed here.

References

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman. *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Co., Reading, MA, 1974.

[2] R. Alur and M. Yannakakis. Model Checking of Hierarchical State Machines. *6th ACM Symposium on the Foundations of Software Engineering,* pp. 175-188, 1998.

[3] R. Alur, S. Kannan, and M. Yannakakis. Communicating Hierarchical State Machines. *Proc. 26th International Colloquium on Automata, Languages, and Programming (ICALP),* Springer Verlag, pp. 169–178, 1999.

[4] E. Asarin and O. Maler. On Some Relations Between Dynamical Systems and Transition Systems. *Proc. 21st International Colloquium, on Automata, Languages and Programming (ICALP),* 820, LNCS, Springer-Verlag, pp. 59-72, Jerusalem, Israel, July 1994.

[5] S. Arora, Y. Rabani and U. Vazirani. Simulating Quadratic Dynamical Systems is **PSPACE**-Complete. *Proc 26th. Annual ACM Symposium on the Theory of Computing (STOC),* pp. 459-467, Montreal, Canada, May 1994.

[6] L. Bajaj, M. Takai, R. Ahuja and R. Bagrodia. Simulation of Large Scale Communication Systems. *Proc. MILCOM'99*, Nov. 1999.

[7] C. Barrett, M. Wolinsky and M. Olesen. Emergent Local Control Properties in Particle Hopping Traffic Simulations. *Proc. Traffic and Granular Flow*, Julich, Germany, 1995.

[8] C. L. Barrett, R. J. Beckman, K. P. Berkbigler, K. R. Bisset, B. W. Bush, K. Campbell, S. Eubank, K. M. Henson, J. M. Hurford, D. A. Kubicek, M. V. Marathe, P. R. Romero, J. P. Smith, L. L. Smith, P. L. Speckman, P. E. Stretz, G. L. Thayer, E. V. Eeckhout, and M. D. Williams. TRANSIMS: Transportation Analysis Simulation System. Technical Report LA-UR-00-1725, Los Alamos National Laboratory Unclassified Report, 2001.

[9] C. Barrett, R. Thord and C. Reidys, *Knowledge and Networks in a Dynamical Economy* Chapter titled *Simulations in Decision Making for Socio-technical Systems* M. Beckman, B. Johansson, F. Snickars and R. Thord, Ed. Springer Verlag, 1998,

[10] C. Barrett, B. Bush, S. Kopp, H. Mortveit and C. Reidys. Sequential Dynamical Systems and Applications to Simulations. Technical Report, Los Alamos National Laboratory, Sept. 1999.

[11] C. Barrett and C. Reidys. Elements of a Theory of simulation I: Sequential CA over random graphs. *Appl. Math. and Comput.*, 98:241–259, 1999.

[12] C. Barrett, H. Mortveit, and C. Reidys. Elements of a Theory of simulation II: Sequential dynamical systems. *Appl. Math. and Comput.*, 107(2-3):121–136, 2000.

[13] C. Barrett, H. Mortveit, and C. Reidys. Elements of a theory of simulation III: Equivalence of SDS. *Appl. Math. and Comput.*, 122:325–340, 2001.

[14] C.L. Barrett, R.J. Beckman, K.P. Berkbigler, and B.W. Bush. Actuated Signals in TRANSIMS. 7th Annual Meeting of Transportation research Board, Technical Report LA-UR 00-7, Los Alamos National Laboratory, 2000.

[15] C. Barrett, H. Mortveit, and C. Reidys. Elements of a Theory of Simulation IV: Fixed Points, Invertibility and Equivalence. *Appl. Math. and Comput.*, 2001. In press (2001).

[16] C. Barrett, M. Marathe, H. Mortveit, S. Ravi, C. Reidys, J. Smith, Ad-hopNET: Advances in Simulation-based Design and Analysis of Ad-Hoc Networks Technical Report LA-UR 00-1567, Los Alamos National Laboratory, 2000.

[17] C. Barrett, H. Hunt III, M. Marathe, S. Ravi, D. Rosenkrantz, R. Stearns and P. Tosic. Gardens of Eden and Fixed Points in Sequential Dynamical Systems. in *Proc. of the International Conference on Discrete Models - Combinatorics, Computation and Geometry* (DM-CCG), Paris, July 2001.

[18] C. Barrett, R. Jacob and M.V. Marathe. Formal Language Constrained Path Problems. *SIAM J. Computing*, 30(3), pp. 809-837, June 2001.

[19] C. Barrett, K. Bisset, R. Jacob, G. Konjevod and M.V. Marathe. Routing in Time Dependent Labeled Networks. Technical Report No. LA-UR-01-4698, Los Alamos National Laboratory, August 2001.

[20] C. Barrett, H. Hunt III, M. Marathe, S. Ravi, D. Rosenkrantz and R. Stearns. Analysis Problems for Sequential Dynamical Systems and Communicating State Machines. in *Proc. International Symposium on Mathematical Foundations of Computer Science* (MFCS'01), Czech Republic, Aug. 2001, LNCS, Vol. 2136, pp. 159–172.

[21] C. Barrett, H. Hunt III, M. Marathe, S. Ravi, D. Rosenkrantz and R. Stearns. On the Complexity of Analyzing Sequential Dynamical Systems, submitted for publication, Sept. 2001.

[22] C. Barrett, H. Hunt III, M. Marathe, S. Ravi, D. Rosenkrantz and R. Stearns. Efficient Local Inter-Simulability Between Models of Dynamical Systems: Towards an SDS-based Computer for Socio-Technical Systems. in preparation 2001.

[23] C.L. Barrett, H.B. Hunt III, M.V. Marathe, S.S. Ravi, D.J. Rosenkrantz and R.E. Stearns Sequential Dynamical Systems with Threshold Functions. Technical Report No. LA-UR-01-4696, Los Alamos National Laboratory.

[24] C. Barrett, A. Marathe and M. V. Marathe. Deregulated Power Markets: Models, Dynamics, Computational Complexity. Technical Report, Los Alamos National Laboratory, 2001.

[25] C. Barrett, M. V. Marathe, H. Mortveit and C. Reidys. SDS Based Specification of Large IP Networks Technical Report, Los Alamos National Laboratory, 2001.

[26] N. Biggs. *Algebraic Graph Theory*, Cambridge University press, 1993.

[27] S. Buss, C. Papadimitriou and J. Tsitsiklis. On the Predictability of Coupled Automata: An Allegory About Chaos. *Complex Systems*, 1(5), pp. 525–539, 1991. Preliminary version appeared in *Proc. 31st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, Oct 1990.

[28] D. Brand and P. Zafiropulo. On Communicating Finite State Machines. *J. ACM*, 30(2), Apr. 1983, pp. 323–342,

[29] A. Clouqueur and D. d'Humieres. A Cellular Automaton Machine for Fluid Dynamics. in *Lattice Gas Methods for Partial Differential Equations*, pp. 251-265, G. Doolen, Ed. Proc. of SantaFe Institute Studies in Complexity, Addison Wesley, MA 1990.

[30] T. Cormen, C. Leiserson, R. Rivest and C. Stein. *Introduction to Algorithms.* McGraw-Hill Book Co., 2000.

[31] J. Cowie, D. Nicol and A. Ogielski. Modeling 100 000 Nodes and Beyond: Self-Validating Design. *DARPA/NIST Workshop on Validation of Large Scale Network Simulation Models* May 1999.

[32] S. Eubank Scalable, Efficient Epidemiological Simulation. to appear in *6th ACM Symposium on Applied Computing (SAC).* Technical Report, LAUR-01-5513 Los Alamos National Laboratory, 2001

[33] P. Floréen, E. Goles, G. Weisbuch. Transient Length in Sequential Iterations of Threshold Functions. *Discrete Applied Mathematics*, No. 6, pp. 95–98, 1983.

[34] P. Floreén and P. Orponen. Complexity issues in Discrete Hopfield networks, in *Comp. and Learning Complexity of Neural Networks: Advanced Topics*, Edited by I. Parberry, 1999.

[35] M. Foster and M. Martin. Probability, Confirmation, and Simplicity. *Readings in the Philosophy of Inductive Logic*, The Odyssey Press, NY, 1966

[36] U. Frisch, B. Hasslacher and Y. Pomeau. Lattice-Gas Automata for Navier Stokes Equation. *Physics Review Letters*, (56), pp. 1505-1508, 1986.

[37] L. Garcia, A. S. Jarrah, and R. Laubenbacher. Classification of finite dynamical systems. In progress (2001)

[38] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman and Co., San Francisco, CA, 1979.

[39] P. Gacs. Deterministic computations whose history is independent of the order of asynchronous updating. Tech. Report, Computer Science Dept, Boston University, 1997.

[40] M. Gouda and C. Chang. Proving Liveness for Networks of Communicating Finite State Machines. *ACM Trans. Programming Languages and Systems* (TOPLAS), 8(1): 1986, pp. 154-182.

[41] F. Green. NP-Complete Problems in Cellular Automata. *Complex Systems,* 1(3), pp. 453-474, 1987.

[42] H. Gutowitz, Ed. *Cellular Automata: Theory and Experiment* North Holland, 1989.

[43] J. Hartmanis and H.B. Hunt III. The LBA Problem and Its Importance in The Theory of Computing. SIAM-AMS Proceedings, pp. 1-25, 1974.

[44] D. Harel, O. Kupferman, and M. Y. Vardi. On the complexity of verifying concurrent transition systems. *Proc. 8th Int. Conf. Concurrency Theory (CONCUR'97)*, Warsaw, Poland, July 1997, Vol. 1243 LNCS, pp. 258–272, 1997.

[45] H. W. Hethcote. The Mathematics of Infectious Diseases. *SIAM Review*, 42(4):599–653, 2000.

[46] C. Hoare. *Communicating Sequential Processes.* Prentice Hall International, 1984.

[47] G. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, 1991.

[48] J. Hopfield. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proc. of National Academy of Sciences of the USA*, No. 81, pp. 3088-3092, 1982.

[49] P. Huang and J. Heidemann. Minimizing Routing State for Light Weight Network Simulation Technical report, TIK-Nr. 106, ETH Zurich, 2000.

[50] P. Huang and J. Heidemann. Capturing TCP Burstiness in Light Weight Simulations Technical report, TIK-Nr. 92, ETH Zurich, 2000.

[51] B. Huberman and N. Glance. Evolutionary games and computer simulations. *Proc. National Academy of Sciences*, 1999.

[52] H.B. Hunt III, D.J. Rosenkrantz, C. Barrett, M.V. Marathe and S.S. Ravi. Complexity of Analysis and Verification Problems for Communicating Automata and Discrete Dynamical Systems. Technical Report No. LA-UR-01-1687, Los Alamos National Laboratory 2001.

[53] H.B. Hunt III, R.E. Stearns and M.V. Marathe, Towards a Predictive Computational Complexity Theory. Technical Report No. LA-UR-00-5985, Los Alamos National Laboratory, submitted, 2001.

[54] G. Istrate, M. Marathe and S. Ravi, Adverserial models in evolutionary game dynamics. *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA'2001)*, Washington, DC, pp. 719–720, Jan. 2001.

[55] F. Kelly. Mathematical modeling of the Internet. *mathematics Unlimited - 2001 and Beyond* B. Engquist and W. Schmid Ed. Springer Verlag 2001.

[56] Y. Kesten, Z. Manna and A. Pnueli. Verifying Clocked Transition Systems. *Hybrid Systems* pp. 13-40, 1995. Complete version in *Acta Informatica* 36(11), pp. 837-912, 2000.

[57] J. Kleinberg. Navigation in a Small World. *Nature* 406(2000). An extended version of this work is available as: The small-world phenomenon: An algorithmic perspective. Cornell Computer Science Technical Report 99-1776, October 1999.

[58] Z. Kohavi. *Switching and Finite Automata Theory*, McGraw-Hill Book Company, New York, 1970.

[59] S.O. Krumke, M.V. Marathe and S.S. Ravi. Approximation Algorithms for Broadcast Scheduling in Radio Networks. *2nd International Workshop on Discrete Algorithms and Methods for Mobile computing and Communications (DIALM),* Dallas, Texas, November 1998. Complete Invited paper to appear in *Wireless Journal*, 2001.

[60] R. Laubenbacher and B. Pareigis. Finite Dynamical Systems. Technical report, Department of Mathematical Sciences, New Mexico State University, Las Cruces.

[61] R. Laubenbacher and C. Reidys. Algebraic Aspects in SDS Theory. in preparation.

[62] R. Laubenbacher, H Mortveit and C. Reidys. Fixed Points of SDS, in preparation.

[63] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees and Hypercubes*, Morgan-Kaufmann Publishers, San Mateo, CA, 1992.

[64] A. K. Lenstra and H. W. Lenstra, Jr. Algorithms in Number Theory. in *Handbook of Theoretical Computer Science*, Vol. A, Edited by J. van Leeuwen, MIT Press, 1990.

[65] N. Lynch. Distributed Algorithms. Morgan Kaufmann, 1997.

[66] S. MacLane and G. Birkhoff. *Algebra*, Macmillan, NY 1967.

[67] M.V. Marathe, H.B. Hunt III, D.J. Rosenkrantz and R.E. Stearns, Theory of Periodically Specified Problems: Complexity and Approximability. *Proc. 13th IEEE Conference on Computational Complexity*, Buffalo, NY, June, 1998.

[68] M.V. Marathe, H.B. Hunt III, R.E. Stearns and V. Radhakrishnan. Approximation Algorithms for PSPACE-Hard Hierarchically and Periodically Specified Problems. *SIAM J. Computing*, 27(5), pp. 1237–1261, Oct. 1998.

[69] N. Margolus and T. Toffoli. *Cellular Automata Machines: A New Environment for Modeling*, Cambridge: MIT press, 1987. See also in *Lattice Gas Methods for Partial Differential Equations*, pp. 219–249, G. Doolen, Ed. *Proc. of SantaFe Institute Studies in Complexity,* Addison Wesley, MA 1990.

[70] B. Martin. A Geometrical hierarchy of graph via cellular automata. *Proc. Mathematical Foundations of Computer Science, (MFCS'98)* Satellite workshop on graph automata, Th. Worsch and R. Wolmar (Editors), Universitt Karlsruhe, 1998.

[71] R. Milner. *Communicating and Mobile systems: the π-calculus.* Cambridge University Press, 1999.

[72] C. Moore. Unpredictability and Undecidability in Dynamical Systems. *Physical Review Letters*, 64(20), pp 2354-2357, 1990.

[73] C. Moore. Generalized shifts: unpredictability and undecidability in Dynamical Systems. *Nonlinearity*, 4, pp. 199-230, 1991.

[74] H. S. Mortveit and C. M. Reidys. Discrete, Sequential Dynamical Systems. *Discrete Mathematics*, 226:281–295, 2001.

[75] K. Nagel and M. Schrekenberg. A Cellular Automata Model for Freeway Traffic. *J. de Physique I, France*, 2:2221, 1995.

[76] K. Nagel and C. Barrett. Using Microsimulation Feedback for Trip Adaptation for Realistic Traffic in Dallas. *International Journal of Modern Physics C*, Vol. 8, No. 3, 1997, pp. 505-525.

[77] K. Nagel, S. Rasmussen and C. Barrett. Network Traffic as a Self Organized Critical Phenomenon. *Self Organization of Complex Structures: From individual to Collective dynamics.* F. Schweiter Ed. pp. 579-592, Gordon Breach London 1997. Technical Report LA-UR 96-659, Los Alamos National Laboratory, 2001.

[78] C. Nicthiu and E. Remila. Simulations of graph automaton. *Proc. Mathematical Foundations of Computer Science (MFCS'98)* Satellite workshop on graph automata, Th Worsch and R. Wolmar Eds, Universitt Karlsruhe, pp. 69-78, 1998.

[79] G. Owen. *Game Theory.* Academic Press, 3rd Edition, 1995.

[80] C. Papadimitriou. *Complexity Theory.* Addison-Wesley, Reading, MA, 1994.

[81] C. Papadimitriou. Algorithms, games and the Internet. *Proc. 33rd Annual ACM Symposium on Theory of Computing.* pp. 749-753, 2001.

[82] W Peng. Deadlock Detection in Communicating Finite State Machines by Even Reachability Analysis. *Mobile Networks (MONET)*, 2(3), pp. 251-257, 1997.

[83] G. Pighizzini. Asynchronous automata versus asynchronous cellular automata. *Theoretical Computer Science,* 132(1-2), pp. 179-207, 1994.

[84] A. Rabinovich. Complexity of Equivalence Problems for Concurrent Systems of Finite Agents. *Information and Computation*, 127(2), pp. 164–185, 1997.

[85] Elements of Distributed Algorithms: Modeling and Analysis With Petri Nets W. Reisig

[86] C. Reidys. Acyclic Orientations of Random Graphs", *Adv. in Appl. Math.* 21, pp.181-192, 1998.

[87] C. M. Reidys. On Acyclic Orientations and SDS. *Adv. in Appl. Math.*, 2000. In press (2000).

[88] C. Reidys. Random Subgraphs of Cayley Graphs over p-Groups. *European J. Combinatorics*, 2000, pp. 1-10.

[89] C. M. Reidys. Phase space properties of SDS. *Adv. in Appl. Math.*, 2001. In progress (2001).

[90] C. Robinson. *Dynamical systems: stability, symbolic dynamics and chaos.* CRC Press, New York, 1999.

[91] Z. Roka. One-way cellular automata on Cayley graphs. *Theoretical Computer Science,* 132(1-2), pp. 259-290, 1994.

[92] C. Schittenkopf, G. Deco and W. Brauer. Finite Automata Models for the Investigation of Dynamical Systems. *Information Processing Letters,* 63(3), pp. 137-141, August 1997.

[93] Y. Sheffi. *Urban Transportation Networks: Equilibrium Analysis with Mathematical Programming methods.* Prentice Hall, 1985.

[94] S. K. Shukla, H. B. Hunt III, D. J. Rosenkrantz and R. E. Stearns. On the Complexity of Relational Problems for Finite State Processes. *International Colloquium on Automata Programming and Languages (ICALP)*, pp. 466-477, 1996.

[95] A. Smith. Simple computation-universal cellular spaces. *J. ACM,* 18(3), pp. 339-353, 1971.

[96] K. Sutner. On the computational complexity of finite cellular automata. *Journal of Computer and System Sciences (JCSS),* 50(1), pp. 87-97, February 1995.

[97] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. *Proc. First IEEE Symposium on Logic in Computer Science (LICS)*, 1986, pp. 332-344.

[98] G. Vichniac. Simulating Physics with Cellular Automata. *Physica*, 10D, pp. 96-115, 1984.

[99] D. West. *Introduction to Graph Theory*, Second Edition, Prentice Hall, 1999.

[100] Traffic Simulation Webpage: `http://cesimo.ing.ula.ve/linkstania.html`. The Network Simulator **ns-2** `http://www.isi.edu/nsnam/ns/`.

[101] S. Wolfram, Ed. *Theory and applications of cellular automata.* World Scientific, 1987.