# Modeling and Analysis of the Collective Dynamics of Large-Scale Multi-Agent Systems: *A Cellular and Network Automata based Approach*[*]

Predrag T. Tošić



[*]This technical report is identical in its content to the author's doctoral dissertation titled "Modeling and Analysis of the Collective Dynamics of Large-Scale Multi-Agent Systems" completed in the early autumn of 2006.

To my mother, sister and grandmother, with love.

And to the loving memory of my father.

**Abstract**

This technical report addresses a particular approach to modeling and analysis of the behavior of large-scale multi-agent systems. A broad variety of multi-agent systems are modeled as appropriate variants of cellular and network automata. Several fundamental properties of the collective dynamics of those cellular and network automata are then formally analyzed.

Various loosely coupled large-scale distributed information systems are of an increasing interest in a variety of areas of computer science and its applications – areas as diverse as team robotics, intelligent transportation systems, open distributed software environments, disaster response management, distributed databases and information retrieval, and computational theories of language evolution. A popular paradigm for abstracting such distributed infrastructures is that of multi-agent systems (MAS) made of typically a large number of autonomous agents that locally interact with each other. This report is an attempt at a cellular and network automata based mathematical and computational theory of such MAS.

The emphasis of this report is placed on the following three important aspects of large-scale multi-agent systems. First, the temporal and causal nature of inter-agent interactions is addressed, and some of its consequences discussed. In that context, a comparison and contrast of cellular automata with different communication models is undertaken. Second, the implications of homogeneity vs. heterogeneity of the individual agent behaviors in a large-scale MAS are analyzed. Third, in conjunction with the models of individual agent behaviors, the impact of the communication network topology on the collective behavior of large agent ensembles is studied.

In particular, it is formally established that a number of fundamental problems about the collective dynamics of large-scale MAS are demonstrably computationally intractable. Moreover, that intractability is shown to hold even when instances of the network automata models of multi-agent systems under scrutiny are severely constrained.

Research summarized in this report strengthens and/or generalizes much of the previous work on the global behavior of various models of discrete dynamical systems studied in the literature, such as the classical cellular automata and discrete Hopfield networks. Among several far-reaching implications of the results presented herein, perhaps the most important is the general conclusion that a highly complex and unpredictable collective dynamics in multi-agent systems can often arise from a synergy of very simple individual agent behaviors and their loosely coupled local interactions.

*Keywords:* multi-agent systems, distributed artificial intelligence, theoretical computer science, formal methods, computational complexity, cellular and network automata, discrete dynamical systems, agent-based modeling, distributed computing, complex networks

# Contents

## List of Tables

# List of Figures

## List of Abbreviations

**ACA** Asynchronous Cellular Automaton/Automata

**AI** Artificial Intelligence

**CA** Cellular Automaton/Automata

**CC** Cycle Configuration

**CFSMs** Communicating Finite State Machines

**coNP** Languages / decision problems whose *complements* are in the class **NP** (computational complexity classes)

**COP** Constraint Optimization Problem

**CSP** Constraint Satisfaction Problem

**DAI** Distributed Artificial Intelligence

**DCO(P)** Distributed Constraint Optimization (Problem)

**DCS(P)** Distributed Constraint Satisfaction (Problem)

**DHN(s)** Discrete Hopfield Network(s)

**DPS** Distributed Problem Solving

**(D)DTA** (Dynamic) Distributed Task Allocation

**DSC(P)** Distributed Set Covering (Problem)

**DSP(P)** Distributed Set Partitioning (Problem)

**FP** Fixed Point (type of configuration)

**FSM** Finite State Machine

**GA** Graph Automaton/Automata

**GE** Garden of Eden (type of configuration)

**LBA** Linear Bounded Automaton/Automata

**MAS** Multi-Agent System(s)

**MCDCF** Maximal Clique-based Distributed Coalition Formation

**MDP** Markov Decision Process

**MMAS** Massive Multi-Agent System(s)

**MSB** Monotone Symmetric Boolean (function or update rule)

**NICA** Nondeterministic Interleavings Cellular Automaton/Automata

**NP** Nondeterministic Polynomial time (computational complexity class)

**OCA** One-way Cellular Automaton/Automata

**OSL** Open Systems Laboratory

**POMDP** Partially Observable Markov Decision Process

**P** (Deterministic) Polynomial Time (computational complexity class)

**PH** Polynomial Hierarchy (concept from computational complexity theory)

**PRAM** Parallel Random Access Machine

**PS** Phase Space (also called *Configuration Space*)

**PSPACE** Polynomial Space (computational complexity class)

**P2P** Peer-to-Peer

**SCA** Sequential Cellular Automaton/Automata

**SDS** Sequential Dynamical System

**SyDS** Synchronous Dynamical System

**TC** Transient Configuration

**TM** Turing Machine

**TSP** Traveling Salesman Problem

**UAV** Unmanned Aerial Vehicle

**UIUC** University of Illinois at Urbana-Champaign

**#P** "Sharp-P" (computational complexity class)

# 1 Prelude: An Outline of Research Vision and Accomplishments

The main themes of this technical report are the formal modeling and the rigorous analysis of large-scale *multi-agent systems* (MAS). By *large scale* we mean those MAS with anywhere from thousands to possibly millions of autonomous agents [193, 202]. By *autonomous agents*, we mean physical, biological, software, robotic and/or other entities that, at the very least, possess the properties of (i) persistence, (ii) reactivity, and (iii) some degree of control of their internal *state*, as well as of their *execution* (that is, *behavior*) as observable by an outside observer, such as another agent [201]. In particular, insofar as our agents' *individual* properties and capabilities are concerned, we do not make any assumptions beyond what is captured by the notion of *weak (autonomous) agency* as defined in [201], and further elaborated upon in [195].

We particularly have in mind large ensembles of *reactive* agents that either are actually known to exhibit a rather simple behavior (for example, certain control devices would fit into this category), or else can be *approximated* as having simple deterministic individual behaviors for the purpose of studying their *ensemble behavior*, that is, the collective dynamics at the granularity level of large agent ensembles [192, 206].

Reactive autonomous agents, that are also often referred to as *situated agents* in the distributed AI literature (see, e.g., [155]), are characterized by the ability to *perceive* and be affected by the changes in their environment, and, in turn, to *act* and thus possibly affect the environment. Reactive or situated agents are usually conceptually envisioned – as well as, when applicable, practically designed[1] – so that they have very little internal structure. In particular, a common mathematical and computational abstraction for such reactive agents is that of a *finite state machine* that may be of a deterministic, nondeterministic or probabilistic variety. In this work, we will primarily focus on deterministic individual agent behaviors, and hence deterministic finite state machine models. Moreover, since we are interested in *large ensembles* of many such interacting agents, the formal models of our interest will be all based on the *communicating (deterministic) finite state machines* abstraction.

One main purpose of this, introductory Section is to provide some broader context for and outline motivations behind our research approach. The other purpose is to briefly summarize all our scientific work since the late fall of 2000 until the completion of our doctoral research; that includes both the particular results that will be subsequently presented in the remaining sections of this technical report, and our research efforts in other subfields within the general areas of autonomous agents and multi-agent systems. The main and, as of early 2006, most complete line of that, "other" research, addresses the problem of *coordination* in collaborative multi-agent systems, and is presented in much more detail in the author's M.S. thesis [193].

Our overall research on autonomous agents and MAS, while inherently interdisciplinary, has been primarily two-pronged.

On the one hand, we have been interested for many years in discrete dynamical systems such as the classical *cellular automata* (CA), as well as their various *graph automata* (GA)[2] extensions

[1] In the case of *engineered*, that is, usually either robotic or software agents.

[2] Also called *network automata* in the literature; see, e.g., [63, 68]). We warn the reader that, throughout most of this technical report, we shall use the two terms interchangeably. We will make exceptions when referring to the *specific* graph automata models proposed by other authors (see Section 3 and introductory subsections of Section 5). In those situations, whenever we have in mind more general models, we shall prefer the term *network automata*, whereas the more specific models found in the literature will be called *graph automata*.

and generalizations.

In particular, we have studied the behaviors, that is, the *configuration space properties*, of several restricted yet interesting classes of cellular and network automata. Our investigations have included both determining under what circumstances would a cellular automaton possess certain properties, and *how hard* it is, given the automaton's formal description, to determine whether or not it would possess the properties of interest. Those properties are typically related to the *global behavior* of these cellular and network automata models, that is, to the *collective dynamics* of an ensemble of (typically, loosely coupled) autonomous agents that the particular cellular or graph automaton model is abstracting. Furthermore, we often focus on the problem of what that global behavior is like *in the long run*. Thus, most of the fundamental problems about the possible cellular or network automata dynamics (or, equivalently, computations) that we have been interested in, essentially ask some variant of the fundamental question: given the *current state* of a cellular or network automaton, and given how each of its elements behaves *individually*, as well as how are these elements interacting with one another *locally*, under what circumstances, and at what computational cost, can it be predicted how is this system going to behave *globally*, in the near or distant future [189, 192, 206].

On the other hand, since joining professor Gul Agha's Open Systems Laboratory (OSL) at University of Illinois at Urbana-Champaign (UIUC) in the Fall of 2001, we have been also working on modeling and analysis of, as well as developing simulation scenarios for, certain *large-scale* multi-agent system (MAS) applications. Among the primary domains for the OSL's DARPA-funded TASK research project, completed in the fall of 2004, were parametric models and a scalable software simulation of large ($10^3 - 10^4$ agents) ensembles of small-sized, limited-resource *autonomous unmanned aerial vehicles* (also known as "micro-UAVs"). For more details and pointers to both research publications and software produced by the OSL group, we refer the reader to *http://osl.cs.uiuc.edu* and especially to the information there that is pertaining to the TASK research project.

One of the ultimate goals of the OSL team in the context of our research project on multi-agent systems (the TASK project) was to develop novel quantitative and parametric models for the large-scale MAS. The author's individual effort within the overall team work on that project chiefly focused on two specific issues. One was developing and analyzing some simple and scalable general-purpose models for an autonomous agent's local-knowledge based decision making (more specifically, task or action selection) in environments that are dynamic, multi-agent, multi-task, resource constrained and partially inaccessible to the agent [197]. The second generic problem was that of multi-agent coordination, and, more specifically, of *reaching distributed consensus* in a scalable, reliable and efficient manner. The two particular types of these distributed consensus problems we investigated are those of *leader election* and *group* or *coalition formation*. The accomplishments on one of these problems are outlined in subsection 1.3, and presented in detail in our MS thesis [193].

The rest of this Section that summarizes most of our doctoral research is organized as follows. Subsection 1.1 is dedicated to introducing cellular and network automata, and motivating the relevance and usefulness of these discrete dynamical system models for modeling and analysis of many important agent ensemble properties in large-scale MAS. The starting point are the classical, parallel CA. However, without some modifications, classical CA are an appropriate abstraction for only a very limited class of distributed information systems. In particular, the

main properties of the classical CA that require appropriate generalizations in order to make thus generalized models relevant in a broader MAS setting are discussed in some detail in subsection 1.1.

Subsection 1.2 then outlines our main results on various properties pertaining to the parallel and sequential threshold CA and their global configurations, and some possible long-term global behavior patterns of such cellular automata. That subsection also summarizes the main results on a particular class of the network automata extensions of the classical (both sequential and parallel) finite cellular automata; this class are Sequential and Synchronous Dynamical Systems. These discrete dynamical systems extend the CA model in two important respects: one, they allow for more general interaction patterns among the agents, and, two, they allow for some heterogeneity in the individual agents' behaviors.

Subsection 1.3 approaches MAS in a more conventional way. In particular, the individual agents are not viewed as simple fixed programs with a very minimal internal structure any longer, but are, instead, *autonomous decision makers* with a much richer internal state and a more complex interaction with their environments. The modeling and design challenges related to such autonomous agents in complex, dynamic and bounded-resource multi-agent environments are then briefly discussed, and our work on one of the aforementioned multi-agent coordination problems summarized.

Last but not least, subsection 1.4 motivates and discusses several promising directions for the possible future work, and outlines some interesting open problems.

## 1.1 A Coarse-Grained View of Large-Scale Multi-Agent Systems

*Multi-Agent Systems* (MAS) are commonly viewed as a research area where (distributed) artificial intelligence and distributed computing overlap. Hence, research in MAS heavily draws on the existing theories, tools and methodologies from both AI and distributed computing. What we would like to contribute to the more thorough understanding and better design of large-scale MAS are some ideas, paradigms and tools from another scientific discipline, namely, *complex dynamical systems* [189, 191, 206]. Among many abstract mathematical models of discrete dynamical systems, the one class that we find particularly appropriate and useful for addressing many fundamental issues in parallel and distributed computing in general, and in large-scale multi-agent systems in particular, are the classical cellular automata, as well as some of their graph or network automata extensions and variants [192, 206].

*Cellular automata* (CA) were originally introduced as an abstract mathematical model that can capture the behavior of biological systems capable of self-reproduction (see Section 4 and references therein). Subsequently, CA have been extensively studied in a great variety of application domains, but mostly in the context of simulation of complex physical, biological and/or socio-technical systems and their dynamics.

However, CA have also been viewed as an abstraction of massively parallel computers [63]. While most of the previous computer science research on CA and similar models have used these models as an abstraction for the parallel and/or distributed computer *hardware architectures* [187], our research agenda is to use these complex system models as a more general abstraction for a variety of distributed systems and infrastructures, including but not limited to (i) teams or coalitions of robots and/or humans and/or unmanned vehicles, (ii) socio-technical systems such

as, e.g., city traffic, and (iii) software agents for *open distributed environments*. More precisely, we would like to apply CA as an abstraction for *autonomously executing local processes* that are coupled to, and interact with, one another and possibly also with their outside environment. Even when these individual processes are rather simple, their mutual interaction and synergy may potentially yield a highly complex and difficult to predict *long-term global* behavior [192]. This property – that the behavior of the "whole" (i.e., the entire system) cannot be easily deduced from the simple and well-understood behaviors of the "pieces" (individual components) – is a hallmark property of both nonlinear complex dynamical systems in physics and open distributed systems in computer science. Thus, the well-known metaphor that "the whole is [sometimes] more than the sum of its parts" was what initially prompted our desire to establish some closer links between these two fascinating research areas.[3]

What are, then, the important properties of large-scale distributed computational and communication systems in general, and large-scale MAS in particular, that can be adequately captured by the classical CA and CA-like models? Let's consider a cellular automaton from a MAS perspective. Studying global dynamics of a CA then translates into an exploration of the global behavior of a multi-agent system when (i) the individual agent behaviors are fixed, (ii) the pattern of multi-agent interaction ("network topology") is fixed, and (iii) both the individual agent behaviors and the interaction patterns among the agents are highly regular and uniform (i.e., *homogeneous*) across the entire system [192]. In particular, CA and other related models capture the critically important MAS features of *locality* of interaction among the agents, and bounded speeds of information and impact propagation.

Several modifications of the basic CA model along different dimensions can be readily argued to be required in order for thus modified CA-based models to provide appropriate abstractions for the *large scale* multi-agent systems [193, 202]. We have identified the following four as the most relevant and important [192]:

- *heterogeneity* of the generalized cellular and network automata in terms of (i) the individual agent behaviors and (ii) the inter-agent interaction patterns, in contrast to the strict *homogeneity* of the classical CA in both these respects;
- *model of inter-agent communication* insofar as whether the agents locally compute synchronously or asynchronously, and whether they interact (communicate) with one another synchronously or asynchronously;
- *adaptability* of the individual agents, i.e., are these agents capable of *dynamically changing their behavior* via, e.g., reinforcement learning, or are their individual behaviors *fixed* once the conditions of the environment and the current state of an agent are specified;
- *dynamic changes* of the MAS network topology, that would be captured by allowing the underlying cellular space of a cellular or graph automaton to change as a function of time.

We will briefly elaborate on each of these extensions, and then focus on the two that have had the most prominent role in our research, and whose study constitutes the core of this technical report.

[3] ... Yet, until very recently, these two areas have been seldom if ever addressed within a single, unifying scientific framework.

Various models of *graph automata* (GA) have been proposed in the literature as straightforward generalizations of the *finite* classical CA. In these network or graph automata models, the agent-to-agent communication pattern(s) need not necessarily be regular, and the individual agents' behaviors need not be uniform. *Synchronous Dynamical Systems* (SyDSs) are a class of finite network automata models where the possible heterogeneity of individual agent behaviors is explicitly captured (Section 5). In an SyDS, while each agent is still a finite-state machine (FSM), different agents, in general, behave like *different* FSMs. This is in contrast to the classical CA, where each agent executes the same FSM program. Moreover, in SyDS the communication network topologies (that is, *cellular spaces* – see Section 4) can be *arbitrary* finite graphs, as opposed to the regular Cayley graphs as the only allowed cellular spaces in the classical CA [63].

Classical CA are also characterized by the *perfect synchrony* of the parallel node updates. This perfect synchrony implies, in effect, logical simultaneity, and is hard to justify on either physical or computer science grounds (e.g., [198]). By allowing the nodes to update one at a time, one arrives at a sequential version of CA, called *Sequential Cellular Automata* (SCA), and sequential versions of the corresponding more general network automata; for instance, *Sequential Dynamical Systems* (SDSs) are a sequential version of the aforementioned SyDS model.

Much of our work thus far has focused on comparison and contrast between some restricted yet nontrivial classes of parallel and sequential CA (see subsection 1.2 and references [191, 198]). However, these sequential cellular and network automata models, while more realistic than their *perfectly synchronous* parallel counterparts, still fall short of being a sufficiently general theoretical model for the large-scale distributed information systems. Namely, a global clock, and therefore communication synchrony, are still assumed in all of those cellular and network automata models [198]. That is, the local computations of agents may be asynchronous, but the inter-agent *interaction* is still implicitly assumed synchronized. Therefore, the natural next step is to study properties of what we call *genuinely asynchronous* cellular and network automata, where no synchrony is assumed when it comes to either local computations or agent-to-agent communication [198]. A thorough qualitative and quantitative comparison and contrast of these genuinely asynchronous models vs. the sequential models vs. the synchronous parallel models would enable us to identify some of those properties of the large-scale MAS that are primarily or solely due to the temporal and/or causal nature of the interaction among the agents, as captured by various models of (a)synchrony of that interaction [198, 205].

Other possible extensions of the CA-like models that would render those models particularly well-suited for the high-level mathematical modeling of MAS, and hence potentially appealing to the autonomous agent and MAS research community, also include allowing for some *adaptability* at the level of individual agents. (In contrast, the classical CA and other models of *interacting finite state machines* explicitly hold the behavior of each single agent fixed and only allow for the adaptation and self-organization phenomena at the *agent ensemble* level.) To that end, rather than requiring that a single node in a CA or GA be a *fixed* FSM (whether deterministic, probabilistic or nondeterministic), one would allow each node to dynamically change its own behavior over time. While the resulting model arguably would not deserve to be called a (cellular or graph or network) *automaton*, this modification would enable one to abstractly capture the *learning capability* of the *individual* autonomous agents.

Last but not least, another modification of the classical CA as well as the more general GA models under consideration pertains to those automata's *network topology*. What could be of

a particular interest for various MAS, as well as *ad hoc* and sensor networks, are the CA-like models whose underlying graphs are dynamically changing. Communicating finite automata on dynamically changing network topologies in general, and various cellular automata models defined on *percolations* in particular, are some possible approaches to abstractly capturing the agents' mobility, possible link failures, and a possibility of the new links being dynamically created in the system [192].

Of the four outlined dimensions along which the classical CA can be readily generalized in order to increase their relevance for and applicability to the domain of large-scale MAS made of many locally interacting autonomous agents, our work thus far has mostly focused on the first two [192]. A brief summary of our most prominent results in that context, accomplished over the period of five years (early 2001 – early 2006) follows in the next subsection.

## 1.2 Some Configuration Space Properties of Parallel and Sequential Cellular and Graph Automata

The first dimension along which it may be worthwhile considering how to generalize the classical CA in order to make them closer to "the real world" (of MAS), as discussed in subsection 1.1, is to simultaneously (i) consider more general graphs as the underlying cellular spaces, and (ii) allow different nodes to behave differently, i.e., to change their local states according to *different update rules*. In that context, we have obtained some results along the following two main lines. One, we have characterized certain types of configurations in the SDS and SyDS network automata models (see the introductory parts of Section 5 for definitions), and established some relationships between various properties of the SDS and SyDS global dynamics, and the existence of those configurations (or their lack thereof). Two, we have obtained several related results on computational complexity of determining various configuration space properties of certain restricted classes of SDSs and SyDSs (see Sections 5 and 6, as well as references [17, 188, 189, 190, 194, 196, 204, 206]).

The two types of configurations of our particular interest are *the fixed points* (FPs) as the *globally stable states*, and *the gardens of Eden* (GEs) as the *unreachable states* [206]. Properties such as the existence of and the speed of convergence to a FP configuration can be readily related to the stationarity and self-stabilizing properties of a multi-agent system's long-term global behavior. Similarly, issues pertaining to the garden of Eden states in network automata can be viewed as abstract formulations of certain safety properties in distributed environments.

While the focus of the research that is summarized in Section 5 has been on formally establishing the hardness of *counting* the FPs and GEs in various restricted classes of Boolean SDSs and SyDSs, we have also obtained, as corollaries, the appropriate hardness results on certain decision problems pertaining to the FP and GE configuration existence, to the existence of more than one FP or GE, as well as the problems about the (non-)invertibility properties of the global maps of appropriately restricted classes of these network automata.

The early stages of this work were accomplished while the author was visiting Los Alamos National Laboratory, Fall 2000 – Summer 2001, under the supervision of Dr. Madhav Marathe and in collaboration with Prof. Harry B. Hunt and Dr. Madhav Marathe. Some of these, early results can be found in reference [17]. The extensions and generalizations of the results presented in that paper can be found in [188, 189, 190, 194]. The results that have originally appeared in the

aforementioned publications, and whose focus is on certain configuration space properties of the general Boolean SDSs and SyDSs, as well as those S(y)DSs with either monotone or symmetric Boolean update rules, are summarized in Section 5 of this technical report.

The more recent research (2005 and early 2006) on the computational complexity of counting FPs and other structures in Boolean SDSs and SyDSs is summarized in Section 6. The common theme of that research is the persistent focus on *severely restricted instances* of Boolean SDSs and SyDSs under consideration with respect to all the model parameters, and especially the structure of the underlying graphs [194, 196, 206]. In particular, the **#P**-completeness of counting FPs as well as of most other fundamental counting problems is established for the SDSs and SyDSs with restricted update rules and defined over *uniformly sparse* graphs [196, 206]. The computational complexity of counting in the context of symmetric Boolean S(y)DSs over uniformly sparse graphs is originally addressed in [206], whereas similar results in the context of monotone Boolean S(y)DSs and discrete Hopfield networks are established in [196]. These two sets of results are given in the first two subsections of Section 6. Among the most difficult technical results of the entire technical report are those in the context of *simple threshold* update rules that are *both* symmetric and monotone. Those results are summarized in subsection 6.3. Finally, to contrast the homogeneous classical CA with those uniformly sparse SDSs and SyDSs that allow for only a minimal amount of heterogeneity insofar as the individual agents' behaviors are concerned, we show in subsection 6.4 that counting FPs is, in principle, easy for the simple threshold cellular automata, such as those whose nodes update according to the MAJORITY function.

Thus, all our computational complexity results about SDSs, SyDSs, CA and discrete Hopfield networks outlined above are summarized in Sections 5 and 6 of this technical report. Most of those results have been already published [17, 188, 189, 190, 192, 196, 204, 206], while a handful of them are as of yet unpublished[4], but will be submitted for a journal publication by the end of 2006.

The second major line of inquiry has been to investigate and analytically characterize some of the consequences of *the perfect synchrony assumption* in the classical CA (see Section 4). In order to make the analysis tractable yet interesting and relevant, we have focused on the simplest class of the *nonlinear totalistic* CA update rules – namely, the *simple threshold functions*. We have been investigating in some detail the configuration space properties pertaining to FPs, cycle states and transient states of such simple threshold CA in both sequential and parallel settings. The emphasis has been twofold. One, we have characterized the main differences in possible behaviors (equivalently, dynamics or computations) of such CA depending on whether the nodes update sequentially, or synchronously in parallel [191, 198]. Two, we have also shown some estimates on what fraction of the entire configuration space of a threshold CA or SCA are the recurrent configurations (cf. fixed points), and provided several parametric characterizations of the cycle, fixed point and transient configurations for the simple threshold (S)CA defined over one-dimensional cellular spaces [200, 205].

In that work, originally initiated in [198, 200] and summarized in [205] as well as in Section 4 of this technical report, we have also introduced the notions of *fair sequences* of node updates, as well as of a new type of *nondeterministic cellular automata*, that we have dubbed *nondeterministic interleavings cellular automata* (NICA). These two new concepts are directly inspired by the notion of *fairness* and the *interleaving semantics* metaphor in the theory of concurrent

[4] As of the early autumn of 2006.

computation, respectively [198].

The notion of *fair* sequential cellular automata makes it possible to meaningfully study the properties of both finite and infinite *sequential* CA beyond the mere (non)existence of particular types of configurations (or, in the infinite cases, of various types of finite subconfigurations). Some of those other behavioral properties of interest include, e.g., those pertaining to how fast is the *fixed point convergence* taking place in various sequential cellular automata scenarios.

On the other hand, capturing all fair (alternatively, arbitrary) infinite node update sequences for the fixed deterministic SCA by a single fair (resp., general) *nondeterministic sequential* CA provides a natural way to strengthen the results on comparison and contrast between the parallel and the *deterministic* sequential CA. In essence, this type of nondeterministic cellular automata introduces the interleaving semantics of concurrency metaphor to the world of cellular automata and, more generally, the world of discrete-time dynamical systems that are made of multiple communicating components. For more details, see [198, 200, 205].

Last but not least, as alluded to before, we have been attempting to relate, compare and contrast the results on CA / SCA / NICA with our previous work on SyDSs and SDSs. Some preliminary results in that framework are presented towards the end of Section 6, and several promising directions for the future work are then discussed in Section 7.

Most of the work on parallel vs. sequential threshold CA, computational complexity of counting fixed point configurations of SDSs, SyDSs and CA, and other problems outlined in this subsection was accomplished between early 2002 and late 2005; during that period, the author was a full-time PhD student with the Open Systems Laboratory (Department of Computer Science, University of Illinois at Urbana-Champaign), under the supervision of his research advisor, professor Gul Agha.

### 1.3 A Fine(r)-Grained View of Large-Scale Multi-Agent Systems

We have argued in subsection 1.1 that many important properties of distributed systems and infrastructures in general, and large-scale MAS in particular, can be adequately abstracted via *discrete dynamical systems* formalisms such as various cellular and network automata models. While these abstract models may suffice for studying many of the MAS properties at the level of ensembles of reactive agents, and their qualitative collective long-term behavior, clearly there are also many other MAS properties of interest that cannot be adequately captured at the granularity level of those cellular and network automata models. In particular, most interesting properties of *individual* autonomous agents that are embedded in an environment, and are interacting with that environment (which, in the MAS context, includes other agents), are entirely *abstracted away* in the CA-based models.

The properties of autonomous agents that require abstractions of finer granularity than that provided by the CA and their variants include, among many others, the following:

– any form of an autonomous agent's *deliberation*, such as reasoning, planning, and individual adaptation/learning, and that part of the agent's internal state that is capturing those aspects of the environment relevant to the agent's deliberation (e.g., *beliefs* about how the world is, *desires* or preferences about how the agent would like the world to be, and so on [146]);

– related to the above, some degree of distinction of an agent's *(internal) state* from its *behavior* (that is observable and may have measurable consequences externally to the agent itself) [201];

– *resources* (both those internal to the agent and especially those external to it, yet potentially available), and *constraints* on those resources [197];

– *bounded rationality* [166] of autonomous agents' knowledge about the world – more precisely, the aspects of bounded rationality going beyond the mere *locality* of an agent's interaction with some of the other, nearby agents;

– the nature of an agent's tasks, goals, utility function, etc., as long as these have something to do with the state of the world outside of the agent itself, and, in particular, with *the agent being driven to change some aspect(s) of the outside world* [201, 229].

The participants of the DARPA TASK project meeting in Santa Fe, New Mexico (October 2002) reached the conclusion that the three main challenges to be addressed, in order to enable the successful design and deployment of large-scale MAS applications in the foreseeable future, are those pertaining to understanding and adequate modeling and analysis of the following critical autonomous agent capabilities:

(i) *individual agent autonomy* – especially in the context of deliberative agents viewed as autonomous decision-makers;

(ii) *multi-agent coordination* and, in particular, the interaction between the coordinated behavior of groups of agents, and the autonomous acting and decision-making of individual agents; and

(iii) *adaptability* in MAS at both the individual agent level and the agent ensemble level, and its interaction with autonomy and coordination (as in, for instance, the *reinforcement learning* of an ensemble of collaborative agents on how to effectively coordinate).

Our research on MAS beyond the work on abstract CA-like models has primarily focused on the autonomous agents' capabilities (i) and (ii) above, and the interaction between those two [197, 199, 202]. The goal has been to develop a modeling and simulation framework for the large-scale MAS such as large teams or coalitions of autonomous unmanned vehicles or large-scale smart sensor networks. This framework should be sufficiently general and broadly applicable, yet based on the realistic real-world assumptions, as well as simple enough to be scalable, at least in principle, to anywhere from thousands up to millions of agents.

The starting assumptions in our models of autonomous agents' acting and decision-making, as well as in the proposed multi-agent coordination strategies, have included the *limited computational, communication* and other *agent resources*, and the implications of those constraints on feasibility and scalability of various models of agents' behavior [197]. Another ontological commitment has been the focus on *collaborative* yet appropriately *locally constrained* multi-agent environments [193, 202, 203]. Under these assumptions, we have proposed several simple, scalable and realistically implementable parametric models for how a bounded-resource autonomous agent can choose an appropriate action in a highly complex, dynamic and partially inaccessible multi-agent, multi-task environment. These action selection mechanisms are based on an agent's local knowledge about the world, and are applicable in hard real-time, severely bounded resource, very large scale system settings. We have also proposed some simple quantitative parametric models of an agent's environment, with an emphasis on the agent's tasks and resources, and how the agent's decision making is coupled to the properties of the environment [197].

The main goal of [197] is to address some of the major challenges involved in understanding and adequately parametrically modeling the decision-making of autonomous agents acting in dynamic, partially inaccessible, multi-task and bounded-resource MAS environments. The em-

phasis in that work is on the three-way coupling among an autonomous agent's decision-making mechanisms, the nature of the agent's environment (cf. in terms of the environments dynamics, (in)accessibility, and resources), and the nature of the agent's tasks, goals and rewards (i.e., payoffs). Two particular properties of the agent environments are identified both as pervasive and (nearly) universal across the agent domains and application areas, and critically important to the agent's decision making process in general, and action selection, in particular. These two properties are *bounded resources* and the physical and/or computational constraints that they impose on the agent, and *bounded rationality* [166], insofar as the agent's knowledge about the world is concerned [193, 197, 202].

Insofar as *multi-agent coordination* in the large-scale MAS is concerned, our main contribution thus far has been a novel, fully decentralized, local communication and local knowledge based algorithm for (distributed) coalition formation. This coalition formation algorithm is based on (i) a simple model of the agents' individual capabilities and resources, and (ii) the implications of the existing communication network topology among the agents. In particular, the algorithm explicitly takes into account both the *locality* of each agent's knowledge about the world, and the *resource limitations* of the agents [193, 202].

Coalition formation in MAS is a special case of the more general *distributed consensus* problem [116]. Multi-agent coalition formation is one of the most important and frequently encountered coordination problems in many MAS domains, and particularly so in the *massively* multi-agent, locally constrained, and collaborative such domains [202].

The proposed coalition formation strategy is based on what we have named *Maximal Clique based Distributed Coalition Formation* (MCDCF) algorithm [193, 199, 202, 203]. This algorithm is a resource-aware, fully decentralized graph algorithm. Each agent is a node in the graph. A pair of nodes is connected by an edge if and only if the corresponding agents can communicate to each other *directly* (as opposed to via multiple hops). The crucial assumption is that this underlying communication network ought to be sufficiently *sparse*. This, indeed, is a reasonable assumption in most very large-scale MAS application domains that we are familiar with; we refer the reader to [193] for several examples of such candidate application domains.

Under the appropriate graph sparseness conditions, we have shown how autonomous agents can efficiently solve variants of the (generally hard) set covering, set partitioning and maximal clique problems in a local and fully distributed fashion, in order to effectively – and without assistance from any sort of a *central authority* – self-organize into many relatively small, but tight coalitions: thus formed coalitions can be argued to be robust to the subsequent node and/or communication link failures. The generic version of the proposed algorithm is envisioned as a basic coordination subroutine that, if certain assumptions on the graph structure hold, is sufficiently efficient to be repeatedly invoked by the agents. The agents may need to repeatedly call this subroutine depending on the underling MAS dynamics, where that dynamics may include the changing communication topology of the agents and/or the changing distribution and resource requirements of the agents' tasks [193, 202].

Insofar as other lines of our work on autonomous agents and multi-agent systems are concerned, we attempt in [195, 201] to address what are the appropriate ontologies and epistemics of autonomous agency from the general systems science and cybernetics perspectives. The emphasis thus far has been placed on the problem of providing a hierarchical, broadly applicable *taxonomy* of various kinds of autonomous agents. The proposed taxonomy is based on the fundamental no-

tion that an agent should be defined mainly in terms of its *attributes* that may have consequences for, and may lead to the agent behaviors that are observable, measurable and testable by, an observer external to the agent itself. In [201], we propose a hierarchical taxonomy that, in our view, captures a broad spectrum of autonomous agents encountered, among other possible domains, in open distributed software environments, robotics and autonomous unmanned vehicles, agent-based computer simulations, and elsewhere. We further elaborate on autonomous agent ontologies and epistemics, as well as on what we consider to be a tendency in the agent ontology research community to *over-anthropomorphize* artificial agents, in [195].

All the work on autonomous agents and multi-agent systems has been accomplished during the time period Fall 2001 – Fall 2004, while the author was a research assistant with professor Gul Agha's Open Systems Laboratory.

### 1.4 A Brief Outline of Some Future Research Plans

In this subsection, we briefly summarize main directions for the envisioned future research. A much more elaborate discussion of those research directions, as well as a motivation behind the proposed problems, will follow in Section 7.

The near future plans, insofar as our research on the CA-like models is concerned, mostly include some extensions and generalizations of the already completed work on various classes of cellular and graph/network automata, and their configuration space properties. We also plan to explore some concrete ways of strengthening our claims about the usefulness of the CA-based and GA-based formal models for addressing many important ensemble-level properties in MAS (see subsection 1.1). In particular, we intend to rigorously compare and contrast some of the fundamental configuration space properties of the *asynchronous cellular automata* (ACA) with those of their sequential (that is, SCA/NICA) and parallel CA counterparts. The current plan is to focus on the nature of nontrivial *temporal cycles* in the threshold ACA. We refer the reader to Section 7 for more details.

Another potential future research direction is to apply the simple threshold (S)CA (and, down the road, possibly also ACA), such as those where the nodes update according to the MAJORITY rule (see Section 4), to the concrete distributed consensus problems – more specifically, to the problems of *distributed leader election* and *distributed coalition formation* in simple network topologies such as rings, wheels or star-like graphs.

Down the road, we also intend to develop at least some very basic ACA / SCA / NICA / CA based *verification formalisms* for the open distributed environments, thereby indicating another potential usefulness of these automata models for the MAS modeling, design and analysis.

Finally, we have some promising ideas on a comparative study of the stochastic vs. deterministic threshold (S)CA and/or S(y)DS. Certain *ergodicity properties* [47] in that context can be readily related to the existence and/or development of *collective memory* in MAS (or its lack thereof).

Insofar as research on the MAS models where the individual autonomous agents are considerably more refined than in the coarse-grained CA-based models, the future work in the early stages of our post-doctoral career is envisioned to proceed along the following two main lines.

One, we would like to further develop and mathematically formalize the individual agents' local-knowledge based action selection models in complex and resource-bounded environments.

In particular, we intend to cast all of our already proposed models [197, 199, 202] into a single *distributed constraint optimization* (DCO) framework. While several such DCO-based formalisms can be readily found in the MAS literature (see, e.g., [128, 231, 232]), it is our hope that some of the agent action selection models that we propose, simple and often sub-optimal as they may be, will be shown to be more scalable and readily implementable in the very large-scale multi-agent environments than the approaches found in the existing literature.

Two, our work on multi-agent coordination in the context of distributed coalition formation [193, 199, 202, 203] certainly can be further improved. One important issue about our MCDCF algorithm [193, 202] yet to be addressed is that of robustness and some degree of fault-tolerance not only with respect to the node and/or link failures, but also with respect to Byzantine (including possibly adversarial) behavior of a small subset of agents in a large agent ensemble. In particular, the current version of the algorithm is potentially highly sensitive even to a *single cheater* (that is, an agent acting as an adversary), assuming the rest of agents are not *a priori* aware of the existence of such a "bad apple" in their midst [193]. Increasing the robustness to myopic or other forms of Byzantine behavior of individual agents during the coalition formation process itself is an important challenge both in the context of our work on coalition formation, and in the area of coalition formation in *collaborative* MAS domains, in general.

Another future research plan, insofar the MCDCF-based coalition formation is concerned, is to cast our maximal clique based distributed coalition formation into the distributed constraint satisfaction/optimization (DCS/DCO) terms, as well. This would enable us to fully formalize our ideas about the interaction, synergy and/or conflict between the autonomous agents' individual decision making and acting on the one hand, and the multi-agent joint coordinated behavior on the other, into a single unifying mathematical modeling and algorithmic framework.

# 2 Introduction

*Multi-Agent Systems* (MAS) are a research area where (distributed) artificial intelligence, software design and distributed computing overlap. In particular, research in MAS heavily draws on the existing theories, tools and methodologies from both AI and distributed computing. We would like to contribute to the more thorough understanding and analysis, as well as (whenever applicable) better design of the large-scale MAS some ideas, paradigms and tools from another scientific discipline, namely, *complex dynamical systems* [188, 189, 190, 191, 206]. Among many abstract mathematical models of dynamical systems, the one class that we find particularly appropriate and useful for addressing many fundamental issues in parallel and distributed computing in general, and in large-scale multi-agent systems in particular, are the classical *cellular automata* and some of their graph or network automata extensions and variants [194, 192, 206]. Therefore, the focus of this work is a thorough analytical study of a number of *behavioral* properties of several cellular automata based models, and discussion of those properties' significance and implications from the viewpoint of large-scale distributed computing and multi-agent systems [191, 192].

## 2.1 Motivation and Our Research Approach

Most of the *large-scale* biological and physical systems are inherently decentralized and distributed. Fully decentralized systems are growing in their number as well as importance among various engineering, socio-technical and other man-designed infrastructures, as well. In particular, computational and communication systems and networks of various kinds are getting increasingly distributed both logically and physically. The behavioral complexity of most such systems does not primarily stem from the sophistication of the individual components, since functioning of those components is typically well-understood. Rather, the challenges of good design and effective analysis of and forecasting about the behavior of such systems are primarily due to the nontrivial interaction and synergy among the individual components at the system level.

Examples of decentralized information systems where the basic understanding of the behavior of individual components or agents is relatively easy, yet inferring or predicting nontrivial aspects of the long-term global behavior of the entire system very difficult or even virtually impossible, are abundant: the Internet, different multi-agent systems (MAS) made of software or robotic agents (e.g., [9, 219, 229]), teams of autonomous unmanned vehicles [197], traffic systems [24], as well as various types of social networks [215, 216], to name just a few widely known such examples.

In order to understand the global behavioral properties of these and many other computational, social, socio-economic, and socio-technical distributed information systems, and to be able to at least sometimes and at least approximately *predict* their long-term dynamic behavior patterns, it seems natural to apply the methodology, tools and paradigms from physics and applied mathematics employed in the study of *complex systems* and their (typically, *nonlinear*) *dynamics*. At the high level of abstraction, the standard questions posed in the distributed computing systems context, such as those related to various *liveness, fairness* and *safety* properties, the problems of *reaching distributed consensus* among the agents on matters of common interest, and the like, can be appropriately formally phrased in terms of the basic *configuration space properties* of the corresponding formal complex system. Some examples of the fundamental configuration

space or phase space properties that are commonly studied in the complex systems literature include:

- TYPES OF GLOBAL BEHAVIOR problems: starting from an arbitrary configuration, is the system guaranteed to converge to a stable or stationary configuration? Is oscillatory behavior possible? How long can the transition take before the system settles into some form of recurrent behavior? Under what circumstances, if any, can the system diverge altogether?

- REACHABILITY type problems: given two global configurations of the system, $A$ and $B$, and assuming the system starts its dynamical evolution from the state $A$, is the state $B$ ever going to be reached? Is $B$ going to be reached after at most $t$ discrete time steps?

- CLASSIFICATION type problems: given a deterministic discrete-time dynamical system, and given an arbitrary system state $A$, is that configuration a stable one? Is it recurrent or transient? If it is recurrent, is it fixed or cyclic with a period of 2 or greater? If it is cyclic, what is the period of re-visiting $A$?

- COUNTING problems: given a discrete state dynamical system, how many of its configurations are fixed points (or, equivalently, what fraction of all possible configurations are stable configurations)? How many are cyclic? How many are transient? How many configurations are unreachable? How big is the basin of attraction of a given stable configuration?

To be able to predict the long-term behaviors of various decentralized information-processing systems, one may want to, first, abstract those infrastructures and translate them into appropriate models of formal dynamical systems, and, second, answer a kind of questions similar to the ones listed above in the context of those complex dynamical systems. The computational hardness of these idealized configuration space problems would then provide *lower bounds* on analyzing the dynamics and emergent behavior of the actual distributed networks and architectures, and on how predictable their long-term behavior can be expected to be.

The formal *discrete-time, discrete-state* dynamical systems that we study are various types of (finite) *cellular* and *network automata* [192].

The first model or, indeed, class of closely related models that we informally introduce in this Section are the classical cellular automata. Cellular automata are the oldest and most studied discrete dynamical systems based on the idea of *communicating finite state machines*. In essence, a cellular automaton is a finite or infinite collection of identical copies of some finite state machine, where these copies are connected to each other in some regular and uniform manner [134]. We informally introduce cellular automata (CA) in the next subsection; the formal definitions of CA, their configurations, and configuration space properties will follow in Section 4.

We then generalize the classical *finite* cellular automata along several orthogonal dimensions. One of those dimensions is the *interaction* or *communication model*; in that context, we shall motivate the study of, first, *sequential* cellular and network automata, and, second, *genuinely asynchronous* cellular automata. Comparison and contrast between the classical, parallel CA whose nodes update their states perfectly synchronously in parallel on the one hand, and sequential and/or asynchronous CA where the nodes update one at a time, on the other, will be the central theme of Section 4.

Another dimension, orthogonal to the underlying communication model, is the *interaction pattern* among the reactive autonomous agents that we abstract as finite state machines. This motivates the study of various *network automata* models, that are also often referred to as *network automata* in the literature [192, 196].[5] In those graph or network automata models, the assumptions of *regularity* and *uniformity* of the interaction pattern among the agents is abandoned, and more general underlying graphs that capture the communication links (or possibly *interaction patterns* of a more general nature than direct peer-to-peer communication) among the agents are considered.

Last but not least, many network automata models found in the literature also allow for different behaviors among different agents in the system; that is, instead of all nodes in a network being identical copies of one and the same finite state machines, which is the case in the classical CA model, different nodes simulate different finite state machines. Much of our study focuses on two specific classes of the network automata models, closely related to each other. These models are *Sequential* and *Synchronous Dynamical Systems.* After informally introducing them in subsection 2.2, we shall study some of the qualitative and quantitative properties of Sequential and Synchronous Dynamical Systems and their dynamics in some detail in Section 5.

Section 6 is a continuation of Section 5, with an emphasis on Sequential and Synchronous Dynamical Systems (SDSs and SyDSs, respectively), as well as the *discrete Hopfield networks* (DHNs), that are *severely restricted* in two orthogonal respects: the kind of underlying graphs they are defined on (i.e., the underlying network topologies), and the nature of the node update rules (that is, each reactive agent's individual behavior). In a sense, we move from the more general graph or network automata such as the SDSs and SyDSs studied in Section 5 back toward the more restricted, CA-like models.

Nearly all results in Section 6 are *computational hardness* results; these results can be interpreted as *lower bounds* on the behavioral complexity of the studied models. Our main agenda in Section 6 is, therefore, two-fold. On the one hand, we show that complex and, in general, computationally infeasible to predict collective dynamics can be obtained even via (uniformly) sparse couplings of rather simple kinds of individual behaviors and local interactions. On the other hand, we try to bridge the gap as much as possible between those (abstractions of) large-scale MAS made of simple reactive agents that have demonstrably predictable collective behaviors (such as those studied through most of Section 4), and those, similar and descriptively only slightly more complex networked multi-agent systems whose behavior, under the usual assumptions in theoretical computer science, are provably infeasible to predict.

The properties we shall focus on in Section 6 are centered at a general question, *how many* different possible dynamics can a given networked system exhibit, assuming we have the full specification of both the system's individual components' behaviors, and their mutual interaction patterns, and assuming that the system behaves fully deterministically. In that context, we address a number of related problems on *enumeration* or *counting* of various dynamical structures of interest, including (but not limited to) the number of system's stable configurations or the number of its unreachable – that is, inaccessible – configurations. Due to severe restrictions on the underlying systems' *structure*, yet such that, as we will formally argue, they still allow for very complex and in general unpredictable *behavior*, of the three Sections with our original results,

[5] We warn the reader that we shall treat the two terms as synonymous, and shall continue using both terms interchangeably throughout the technical report.

Section 6 represents the technically most challenging part of this technical report, and of our overall contribution thus far[6] to the state-of-the-art in the areas of complex dynamical systems and theory of large-scale multi-agent systems.

## 2.2 Discrete Dynamical System Models of Interest

CA can be defined by first considering *(deterministic) Finite State Machines* (FSMs) that do not produce any output, but may change their *state* depending on their current state and input. Such *output-less* finite state machines are also known as *Deterministic Finite Automata* (DFA) in the literature. An FSM has finitely many states, and is capable of reading the input signals coming from the outside. The machine is initially in some starting state; upon reading each input signal, the machine changes its state according to a pre-defined and fixed rule. In particular, the entire memory of the system is contained in what *current state* the machine is in, and nothing else about the previously processed inputs is remembered. Hence, the probabilistic generalization of deterministic FSMs leads to (discrete) Markov chains. It is important to notice that there is no way for a FSM to overwrite, or in any other way affect, the input data stream. Thus *individual* FSMs are computational devices of a rather limited power.

Now let us consider many such FSMs, all identical to one another, that are lined up together in some regular fashion, e.g., on a straight line or a ring or a regular 2D Cartesian grid or torus, so that each single *node* in the grid is connected to its immediate neighbors. We also eliminate any external sources of input streams to the individual machines at the nodes, and let the current values of each node's neighbors be that node's only *input data*. If we then specify a finite set of the possible values held at each node, and we also identify this set of values with the set of a node's *internal states*, we arrive at an informal definition of a classical cellular automaton.

To summarize, a CA is a finite or infinite regular grid in one-, two- or higher-dimensional space, where each node in the grid is a FSM, and where each such node's input data at a given time step are the corresponding *current values* (that is, states) of that node's neighbors. Moreover, in the most important special case, namely, the Boolean case, that FSM is particularly simple, i.e., it has only two possible internal states, usually labeled 0 and 1.

All the nodes of a classical CA execute the FSM computation in unison, i.e., *logically simultaneously* [198]. We note that the infinite CA are capable of universal (Turing) computation. Moreover, the general class of infinite CA, once arbitrary starting configurations are allowed, are actually strictly more powerful than the classical Turing machines; for more details, see [63]. There are many variants of the basic CA definition as informally stated above, as well as many generalizations of the basic model; see the discussion in Section 3 or, for much more details, [63].

We will formally define all needed CA-related concepts in the early subsections of Section 4.

Next, we briefly summarize the motivation and history behind cellular automata. CA were originally introduced as an abstract mathematical model that can capture the behavior of biological systems capable of self-reproduction [134]. Subsequently, CA have been extensively studied in a great variety of application domains, mostly in the context of physics and, more specifically, complex physical or biological systems and their dynamics (e.g., [70, 224, 225, 226, 227]). However, CA can also be viewed as an abstraction of massively parallel computers (e.g, [63]). In Section 4 we study a particular simple yet nontrivial class of CA from the parallel and distributed

[6] As of the spring of 2006.

computing perspectives. In particular, we pose – and partially answer – some fundamental questions regarding the nature of the classical CA *perfect synchrony*, and that perfect synchrony's implications.

Classical CA, among other properties, are characterized by (i) the uniformity of the interaction topology among the computing agents, and (ii) the homogeneity of the behavior across the agents. As such, classical CA have been generalized in a few different ways and under many different names. For instance, in the theoretical biology literature, CA-like models defined over more general underlying graphs have been referred to as *Discrete Boolean Networks*, *Network Automata*, *Random Boolean Networks* or *(Random) Automata Networks* (see, e.g., [10, 68, 69, 103, 105, 106]). In theoretical computer science, various notions of *Graph Automata* have been proposed and studied (e.g., [119, 135]). One particular type of finite network automata, where the individual automata update *sequentially* rather than synchronously in parallel, was proposed in [12, 19, 21] as a generic mathematical model for computer simulations of various *decentralized* large-scale systems. These sequential network automata are called *Sequential Dynamical Systems* (SDSs). Much of our technical report work (see Sections 5 and 6) has focused on the study of some important qualitative and quantitative computational properties of SDSs.

An SDS $\mathcal{S} = (G, F, \Pi)$ consists of three components. $G(V, E)$ is an undirected graph with $n$ nodes with each node being characterized by its *state* – a value from some *finite* domain. In the sequel, we shall restrict our attention to the Boolean (that is, binary-valued) domains and update rules. $F$ is the *global map* of $\mathcal{S}$; it is obtained by appropriately composing together the local update rules, each of which affecting the state of a single node. Last but not least, $\Pi$ is a permutation of (or a total order on) the nodes in $V$. A *(global) configuration* of an SDS is an $n$-bit vector $(b_1, b_2, \ldots, b_n)$, where $b_i$ is the value of the state of node $v_i$ (where $1 \leq i \leq n$). A single SDS transition from one configuration to another is obtained by updating the state of each node using the corresponding Boolean function. These updates are carried out in the order specified by $\Pi$. If the permutation $\Pi$ is omitted, and all the nodes update synchronously in parallel (the way the nodes of classical cellular automata update), we arrive at the definition of *Synchronous Dynamical Systems* (SyDSs). All these concepts will be formally defined at the beginning of Section 5.

Thus, it can be seen that SDSs and SyDSs generalize the classical (sequential and parallel, respectively) cellular automata in two orthogonal, but equally important, respects. One, by allowing arbitrary finite graphs as the underlying *cellular spaces*, much more general *interaction patterns* among the communicating agents can be captured. Two, the property that different nodes of an SDS or SyDS, in general, are being allowed to behave like different finite state machines, is one way of capturing *heterogeneity* in the individual agents' behaviors.

One of the main themes of our dissertation work (see Sections 5 and 6) has been the computational complexity of questions concerning the stable configurations, also called *Fixed Point* (FP) configurations. In particular, we address in detail the problem of the FP existence and especially that of *counting* how many FPs a given sequential or synchronous dynamical system or cellular automaton may have. We will show that, in general, the question about the number of FPs is hard, even when the underlying structure of an SDS or SyDS, as well as the local update rules, are severely restricted.

In the most general Boolean SDSs, each node is an arbitrary Boolean-valued deterministic finite automaton; that is, each node updates its current state according to an arbitrary (Boolean-

valued) update rule. Several restricted versions of Boolean SDSs have been proposed. Among these, perhaps the most extensively studied are the SDSs with *symmetric Boolean functions* as the node update rules [16, 19, 20, 110, 204, 206]. Symmetric Boolean functions are characterized by the property that their output depends only on $\Sigma_i x_i$, and not on which individual $x_i$ are equal to 0 and which ones are 1. These functions as the update rules in dynamical systems capture the *mean field effects* in statistical physics and modeling of other large-scale systems [15, 16]. Symmetric Boolean functions have been extensively studied in computer science, as well. The areas where symmetric Boolean functions are of major interest range from the classical computational complexity (see, e.g., [175, 217]), to the complexity of parallel computation [46], to the combinatorial complexity of general circuit design [31, 234] and communication complexity [7].

Another important class of Boolean functions that we study in the context of SDS and CA update rules in this technical report are the *monotone functions* [217]. Much of the computational theory of monotone Boolean formulae has been developed in the 1980s (see, e.g., [1, 2, 147, 213]). Subsequently, the emphasis has been on the applications of these monotone theories to the areas such as the logical circuit design (e.g., [31]), machine learning [28, 108, 165, 230], and data classification [3]. Perhaps the best known application of monotone Boolean functions, however, has been as the local update rules in various types of *artificial neural networks* (ANNs) [76, 81, 142].

Among a variety of the candidate monotone Boolean update rules, *linear threshold functions* (with all weights $w_{ij} \geq 0$) are particularly prominent both in the literature on artificial neural networks (see, e.g., [76, 81, 82, 137, 138]) and in the work on sequential dynamical systems [16, 17, 194, 196]. Boolean-valued linear threshold functions can be used for the *linear separation* of the inputs: such a function evaluates to 1 *if and only if* a weighted sum of the inputs, $\Sigma_i\, w_i \cdot x_i$, is equal to or exceeds a pre-specified threshold, $\Theta$. SDSs and SyDSs with linear threshold rules will be one of the most prominent classes of discrete dynamical system models in this technical report, as well.

The most restrictive type of linear threshold rules are the *simple threshold rules* [16, 198, 200] in which all the weights are positive and equal to one another. Therefore, simple threshold functions are *both monotone and symmetric.* The important special cases of simple threshold functions include the AND, OR and MAJORITY Boolean functions. The relationship between simple threshold functions, formulae and logic gates such as MAJORITY on the one hand, and the more general threshold functions, formulae and gates on the other, has been extensively studied in the literature; see, e.g., [66, 67, 78, 163, 213]. Our sharpest results on the computational hardness of *counting* in SDSs and SyDSs will be in the context of S(y)DSs with simple threshold node update rules and that are defined over *uniformly sparse* underlying graphs.

Formal definitions of SDSs and SyDSs, their components, and types of their configurations will be given at the beginning of Section 5. *Table* 1 below summarizes the communicating finite state machine based models of discrete dynamical systems that are studied in the rest of this technical report.

| | cellular automata | network automata |
|---|---|---|
| synchronous models | parallel cellular automata | Synchronous Dynamical Systems, synchronously updating discrete Hopfield networks |
| asynchronous models | sequential cellular automata, genuinely asynchronous cellular automata | Sequential Dynamical Systems, sequentially updating discrete Hopfield networks |

Table 1: Discrete dynamical system models studied in this technical report: variants of *cellular automata* and *network automata*, classified with respect to the communication model

### 2.3 Report Outline

The rest of this technical report is organized as follows. First, we briefly survey some related work in Section 3. The first subsection of that section gives an overview of some of the well-known work on the computational aspects of classical cellular automata. The emphasis there is on various properties of *finite* cellular automata, and computational complexity of determining those properties. In the next subsection, some extensions and generalizations of the classical CA are outlined, and some pointers to the relevant literature provided; we mostly cite the research literature where the models studied are most similar to our network automata of interest, namely, SDSs and SyDSs. Relevant previous work on S(y)DSs and their configuration space properties is also surveyed in that subsection. Last but not least, the third subsection of Section 3 covers the classical as well as some of the most relevant recent work on the computational complexity of counting.

Section 4 addresses the comparison and contrast between perfectly synchronous parallel CA, and various sequential CA models. This comparison and contrast is done in the context of *(simple) threshold* update rules. Parallel and sequential CA whose nodes update according to the most interesting simple threshold rule, namely, the MAJORITY rule, are studied in some detail in that section, as well. In the discussion part of the section, we motivate and propose a definition of *genuinely asynchronous cellular automata,* as the ultimate CA-based abstract model for distributed computing and distributed AI. Genuinely asynchronous CA are, in our view, a reasonable compromise between the simplicity of CA-like models on the one hand, and the need for a realistic model of inter-agent interaction applicable to the large-scale ensembles of reactive autonomous agents, on the other.

Sections 5 and 6 represent the core of this technical report. In those two sections, we undertake a detailed study of *computational complexity* of determining a number of *configuration space* properties (that is, possible dynamic behaviors) in the context of Boolean SDSs and SyDSs. Among a variety of configuration space properties studied in the literature, we primarily focus on the complexity of counting various types of configurations. In particular, we prove a number of results on the hardness of counting various types of configurations in several restricted classes of Boolean SDSs and SyDSs. As mentioned in the previous subsection, the emphasis is given to the problems of *exact* and, to a lesser extent, *approximate counting* of *Fixed Points* (FPs)

and other dynamical structures found in these discrete dynamical systems. In particular, we show that counting FPs is **#P**-complete even when the underlying SDS or SyDS is sparse, uses only two different update rules, and these update rules are required to be symmetric, monotone, and/or linear threshold. We also establish similar intractability results for the discrete Hopfield networks. In contrast, we show that exactly counting FPs in one-dimensional simple threshold CA is computationally feasible.

Finally, Section 7 summarizes this technical report, as well as briefly identifies several open problems, most of which closely related to the results presented in Sections 4, 5 and 6. The final Section of this technical report also outlines some possible directions for the future research in the general area of cellular and network automata based abstract models of large-scale multi-agent systems and their collective dynamics.

# 3 Related Work

In this Section, we overview both classical and more recent research literature that is most relevant to the work presented in the rest of this technical report. First, we will survey some (mostly classical) literature on various computational and dynamical aspects of cellular automata. Second, we will briefly survey the work on those network automata generalizations of the classical cellular automata that are the closest to the models of our interest. Third, we will provide some pointers to research on the computational complexity of *counting* or enumerating various combinatorial structures, with an emphasis on those papers whose results are used in our *(weakly) parsimonious*[7] reductions in Sections 5 and 6.

## 3.1 Computational Problems about Cellular Automata: A Brief Overview

Computational aspects of the classical Cellular Automata have been studied in many contexts. Prior to the 1980s, most of the theoretical work dealt with infinite CA and the fundamental (un)decidability results about the global CA properties. Some examples of such properties of infinite CA are surjectivity, injectivity, and invertibility of a cellular automaton's *global map*; see, e.g., [4, 133, 150]. Systematic study of other computational aspects of CA, from topological to formal language theoretic to computational complexity theoretic, was prompted in the 1980s by the seminal work of S. Wolfram [223, 224, 225]. Among other issues, Wolfram addressed the fundamental characteristics of CA in terms of their computational expressiveness and universality. He also offered the first broadly accepted classification of all CA into four qualitatively distinct classes in terms of the structural complexity of the possible computations or, equivalently, dynamical evolutions. Another seminal work from that period, where the CA and one-way CA computational universality properties and several other fundamental computability theory results are proven, is the work by Culik *et al.* [39, 41]. The state of the art pertaining to a broad variety of computational properties of CA in both theoretical and experimental domains at the end of that decade can be found in [75].

Since most interesting global properties of sufficiently general infinite CA have been shown to be formally undecidable, the computational complexity proper (that is, as contrasted with the computability theory) has been mainly concerned with the computational aspects of *finite* CA, or those pertaining to *finite*[8] *subconfigurations* of infinite CA. Most work within that framework has focused on the fundamental *decision problems* about the possible CA computations; examples of such problems are the first three problem classes listed in subsection 2.1. We provide below a very short survey of some of the more important results in that context.

Insofar as the computational complexity of fundamental problems about *finite* CA and their configuration space properties are concerned, we single out the following. The first **NP**-complete problems for CA are shown by Green in [72]; these problems are of a general REACHABILITY flavor, i.e., they address the properties of the FORWARD DYNAMICS of CA. Sutner addresses the

[7] The notions of parsimonious and weakly parsimonious reductions, as well as other necessary concepts from the computational complexity theory, will be defined in the introductory subsections of Section 5.

[8] That is, finitely or compactly supported; see, e.g., [63].

BACKWARD DYNAMICS problems, such as the problem of an arbitrary configuration's PREDECESSOR EXISTENCE, and their computational complexity in [179]. In the same paper, Sutner establishes the efficient solvability of the predecessor existence problem for an arbitrary CA with a *fixed neighborhood radius.* In [48], Durand solves the injectivity problem for arbitrary 2-D CA but restricted to the *finite subconfigurations* only; that paper contains one of the first results on **coNP**-completeness of a natural and important problem about CA. Furthermore, Durand addresses the REVERSIBILITY PROBLEM in the same, two-dimensional CA setting in [49].

Expressiveness and computational power of various variants of cellular automata with respect to each other, as well as with respect to the classical models of (sequential or parallel) computation such as Turing machines, Linear Bounded Automata (LBA) or Parallel Random Access Machines (PRAMs) have also been subjects of a considerable research interest (e.g., [63, 88]). In particular, [88] provides a self-contained survey of most important results on the relative powers of several CA-based models such as *linear cellular arrays* (that are equivalent to the finite 1D cellular automata with *fixed* boundary conditions[9]), *mesh connected arrays* (2D cellular automata) and *one-way cellular arrays* (that is, one-directional CA in one or two dimensions), and on the power of those and similar CA-based models with respect to the classical computational models and complexity classes.

Among the restricted versions of the classical parallel and sequential CA, perhaps the most studied variant are the *One-way Cellular Automata* (OCA) [51, 87, 88, 182, 209]. The central theme in comparing OCA with the ordinary, "two-way" CA is to analyze and quantify the impact of strictly one-directional information propagation; in particular, the concrete problems include (i) determining which properties of two-way CA can be simulated by appropriate OCA, (ii) identifying those properties that provably cannot, and (iii) for those CA behaviors that can be simulated by OCA, establishing how (in)efficient are the most optimal one-way simulations of the two-way CA dynamics [87, 88, 209].

Another approach to studying the expressiveness and computational power of CA is to view cellular automata as *language recognizers* (e.g., [44, 45]). Language recognition capabilities have been studied both in the context of "two-way" one-dimensional and two-dimensional CA, and of their "one-way" (that is, OCA) analogues [45]. Examples of interesting languages whose recognizability and acceptance by (O)CA have been under scrutiny include, for example, regular and context-free languages (over binary or other finite alphabets), palindromes over finite alphabets, the language of strings whose lengths are prime number (over a unary alphabet), and many other.

Determining the language recognition and acceptance power of various variants of CA and OCA becomes particularly interesting once certain constraints are imposed onto *how quickly* can a (O)CA of a given general or restricted kind recognize (respectively, accept) a formal language from a particular class. The two most studied computational complexity classes of CA language recognition / acceptance problems with respect to *time bounds* are *linear time* languages and *real time* languages [45, 88]. A comprehensive survey of the 1980s and 1990s research on CA as

[9]We warn the reader of two rather different uses of the attribute *linear* in the CA literature. On the one hand, that term is used with respect to the CA update rules: a CA is linear if its update rule is linear (or affine), that is, if it is *additive*. We will use the term *linear*, when applied to CA, SDSs, Hopfield nets or any other model, exclusively in this sense. In contrast, the second usage of the term *linear*, altogether unrelated to the first, is that found in e.g. [88]. It pertains to the CA's cellular space, not update rule; i.e., a CA is called linear if its cellular space is *one-dimensional*. To avoid confusion, we shall strictly adhere to the first usage of attribute *linear*, and refer to those CA defined on "linear" cellular spaces as *one-dimensional*.

language recognizers can be found in [45].

One of the most comprehensive monographs on a broad variety of mathematical and computational aspects of CA in general, and the computational complexity theoretic aspects in particular, is [63]. For a short yet fairly comprehensive historical survey covering much of CA research during the period from 1960s through 1990s, we refer the reader to [160].

Viewing cellular automata as a formal model of computation is one standard approach to studying CA and their behavior; that approach is based on what is essentially a computer scientist's view of cellular automata. From a physicist's perspective, on the other hand, a cellular automaton is first and foremost a discrete dynamical system. Hence, another major body of research on cellular automata and their behavior is based on treating CA as formal discrete dynamical systems, and then studying those dynamical systems' behavior the same way one studies the dynamics of, say, *iterated maps* and other common mathematical models of discrete-time dynamical systems (e.g., [34]).

Among different kinds of dynamical systems' properties that have been investigated in the research literature, we identify those that are of interest within one (or both) of the two particularly prominent approaches to CA as formal dynamical systems. One of those approaches is based on studying *topological properties* of the CA configuration spaces: what kinds of topological sets of configurations result from the repeated forward and/or inverse iteration of an update rule, as applied to all or a restricted class of the possible starting configurations of a cellular automaton. Most of that work has focused on *one-dimensional* CA defined on a two-way infinite line (e.g., [26, 37, 117, 118]). Studying topological properties of a CA's dynamics, such as *transitivity* and *sensitivity to initial configuration*, are at the core of analytically identifying the possibility of chaotic behavior in dynamical systems; thus, chaos in cellular automata has been a popular research subject through much of the 1990s [34, 117, 121].

The other frequently encountered mathematical approach to rigorously studying CA dynamics that has caught our attention is based on the classical mathematical analysis and especially *measure theory* [27]. In particular, using measure theoretic concepts and providing appropriate characterizations of CA dynamics can provide some insight into which (among possibly great many) possible dynamical evolutions for a given CA update rule lead to *statistically robust behaviors* [8]. In this technical report, mostly for illustration and the purpose of completeness of our characterizations of the possible behaviors of a particular class of CA, we shall only establish one result about simple threshold CA dynamics that uses probability theory and elementary measure theory; see Section 4 for details.

Both topological and measure theoretic properties of CA dynamics are jointly addressed in [27]. In [109], the ergodic and topological properties of CA dynamics are related to the CA language recognition properties discussed earlier in this subsection. Good surveys of the most important results on CA as formal dynamical systems can be found in [34, 99].

We end the present subsection with summarizing our own completed research on CA and their computational and dynamical properties. We have begun the study of possible computations and configuration space structures of parallel and sequential simple threshold CA in [198, 200, 205]. The emphasis in [198] is on some computational implications of the perfect synchrony assumption in the classical, parallel CA and, in particular, on a comparison and contrast between the possible computations of parallel threshold CA and those of their sequential counterparts. In [200], the configuration space properties of simple threshold CA are studied further, with the focus on the

most interesting such node update rule, namely, the MAJORITY function. Among other results, it is shown in that paper that (i) an infinite 1-D MAJORITY CA has *uncountably many* FPs, (ii) almost all configurations of such CA are transient. Finally, [205] summarizes (most of) our findings on simple threshold parallel and sequential CA to date. Section 4 of this technical report is, for the most part, based on [205]; however, it also expands on that paper, chiefly in terms of characterizing the *speed of convergence* of (parallel) simple threshold CA (see subsection 4.4.2).

### 3.2 Some Related Graph and Network Automata Models

A considerable variety of generalizations of the classical cellular automata can be found in the literature. One direction along which the classical CA can be generalized is to allow more general underlying graphs or cellular spaces. Another frequently encountered generalization of CA is with respect to *homogeneity* vs. *heterogeneity* of the node update rules, i.e., whether all the nodes behave the same, or, instead, different nodes are allowed to update according to different update rules. The most common names for a variety of fairly similar graph or network automata models that generalize classical CA in those two respects that one encounters in the literature are *network automata*, *random (Boolean) networks* and *automata networks* [63].

The motivation behind generalizing the cellular spaces and/or allowing for the heterogeneous update rules in one particular manner or another typically depends on the particular research community. For instance, *Kauffman's networks* in theoretical biology [103] may elsewhere (say, among the complex dynamical systems research community) simply be referred to as *random (Boolean) networks* [63, 68, 70]. In our own work, the motivation primarily stems from our interest in the collective dynamics of large-scale multi-agent systems made of autonomous robotic, software and/or human agents. We provide a more elaborate motivation for the cellular and network automata based approach to studying behavior of large agent ensembles in [191, 192, 196, 206].

Among a variety of network automata models and their applications found in the literature, *Sequential* and *Synchronous Dynamical Systems* (SDSs and SyDSs, respectively) have been most prominent in the context of computer simulation of various large-scale socio-technical systems and decentralized infrastructures [12, 207]. SDSs and SyDSs investigated in this technical report are also closely related to the *Graph Automata* (GA) models studied in [119, 135], and the aforementioned *One-Way Cellular Automata* defined on Cayley graphs more general than just 1D and 2D Cartesian grids, rings and/or tori that are studied by Roka in [153]. In fact, the general *finite-domain* SyDSs exactly correspond to the Graph Automata of Nichitiu and Remila as defined in [135]; more on the relationship between SDSs and SyDSs on one hand, and other graph automata models found in the literature on the other, can be found in [16, 17].

The main difference between the *graph automata* referred to above and the SDSs is the sequential ordering aspect. SDSs generalize finite *sequential* CA in that they allow *arbitrary (finite) underlying graphs*, as opposed to restricting cellular spaces to the regular Cayley graphs only [63]. We observe that the *sequential cellular automata* have been studied since at least 1984 [89], but under the (potentially somewhat misleading [198]) name of *asynchronous cellular automata*.

In the mid and late 1990s, several other authors have also started considering this particular aspect of the sequential vs. perfectly synchronous parallel node updates [60, 83, 153]. In particular, Huberman and Glance [83] discuss how simulations of certain $n$-person games exhibit very different – but probably more realistic – dynamics when the cells are updated *sequentially* as

opposed to when they are updated *synchronously in parallel*. The issue of sequential ordering has been also discussed in [13, 19, 132] in the context of developing a theory of large scale simulations. We will address the issue of the communication model in CA in detail in Section 4.

Barrett, Mortveit and Reidys [19, 20, 132, 148] as well as Laubenbacher and Pareigis [110] initiate a rigorous mathematical investigation of various properties of Sequential Dynamical Systems. Barrett *et al.* study the computational complexity of several phase space properties of Boolean and other finite-domain SDSs. The problems of interest include REACHABILITY, PREDECESSOR EXISTENCE and PERMUTATION EXISTENCE problems [14, 16].

Problems related to the existence of *garden of Eden* (GE) and *fixed point* (FP) configurations in the context of Boolean and more general finite domain SDSs and SyDSs are originally addressed in [17]. In particular, the basic **NP**-completeness results for the problems of FP, GE and non-unique predecessor existence in various restricted classes of Boolean S(y)DSs are proven in that paper. Algorithms for efficiently finding an FP in certain other restricted classes of S(y)DSs can be also found in [17]. We remark that some of the most important results in Sections 5 and 6 of this technical report can be viewed as a natural extension of the work in [17]; namely, instead of the *decision problems* about the fixed points and gardens of Eden in SDSs and SyDSs, we focus in most of Section 5 and all of Section 6 on studying the related *counting problems*.

Among various restricted classes of Boolean SDSs and SyDSs, those with the local update rules restricted to *symmetric functions* have received a particular attention (e.g., [20, 110, 132]). We address symmetric Boolean S(y)DSs in the context of interesting counting problems in [189, 204, 206]. In particular, several of our results in Sections 5 and 6 are about the complexity of counting the fixed point configurations of symmetric Boolean SDSs and SyDSs.

Another important and broadly studied class of update rules are *monotone* Boolean functions. We study monotone Boolean SDSs and SyDSs in [190, 194, 196]. Most fundamental decision and search problems about monotone Boolean SDSs and SyDSs, including the problem of fixed point existence, can be readily seen to be easy to solve [194]; however, we will show in this technical report that most fundamental counting problems about monotone Boolean S(y)DSs, such as those of enumerating such an S(y)DS's fixed point or garden of Eden configurations, are in the worst-case provably intractable.

Computational complexity of the reachability-related problems in the context of, among other restricted types, symmetric, monotone and simple threshold Boolean SDSs and SyDSs is investigated in [16]. In particular, it is shown in that paper that different variants of the reachability problem are solvable in polynomial time for *symmetric* as well as for *monotone* Boolean SDSs. Thus, our results on the complexity of counting, originally proved in [189, 204, 206] and summarized in this technical report, provide rare examples of intractability for the symmetric Boolean SDSs and SyDSs (however, see also [17]). Furthermore, our hardness of counting results from [190, 194, 196] also provide, to the best of our knowledge, the only examples thus far of important configuration space properties that are provably intractable for Boolean SDSs and SyDSs (or any similar graph or network automata model) with the *monotone* node update rules.

### 3.3 A Brief Survey of Computational Complexity of Counting

As already mentioned, one of the main themes of this technical report is the computational complexity of *counting* fixed points, gardens of Eden and other types of configurations in discrete

dynamical systems such as symmetric SDSs and SyDSs, monotone S(y)DSs, discrete Hopfield networks and simple threshold CA and sequential CA.

The subarea of computational complexity that addresses counting or enumeration of various combinatorial structures dates back to the seminal work of L. Valiant in the late 1970s [211, 212]. Counting problems naturally arise in a great variety of contexts. For example, in artificial intelligence (AI) alone, counting problems and their computational complexity are of a considerable interest in subareas such as approximate reasoning and Bayesian belief networks (e.g., [43, 144, 157]), constraint satisfaction [23], and automated deduction [79]. Exact and approximate counting problems arise in many areas that are not considered to "belong" to theoretical computer science or AI, as well. Examples include network reliability and fault-tolerance (e.g., [100, 101, 210]), statistical physics [91, 93, 114] and theoretical biology [32, 159].

However, it has been observed that the progress in understanding the complexity of counting problems has been much slower than the progress related to our understanding of decision and search problems [73, 210]. Since the reductions used in proving counting problems hard have to preserve the number of solutions, rather than just whether a solution exists or not, they are in general more difficult to devise than the reductions used to establish, say, **NP**-completeness of the corresponding decision problems. For example, most standard reductions used to establish the computational hardness of certain decision or search problems on graphs tend to "blow up" the underlying graph, thereby destroying the local structures that impact the number of that problem's solutions [73].

One area where this understanding of the complexity of counting was, until fairly recently, particularly poor, is whether the general counting problems that are provably hard remain hard when various restrictions are placed on the problem instances [210]. Some of the relatively recent results in that area, such as the hardness of counting in *planar graphs* [85], and especially in *uniformly sparse graphs* [73, 210], have directly inspired our work as summarized in Section 5 and particularly Section 6.

Counting problems naturally arise in the context of dynamical systems, as well. Indeed, being able to efficiently solve certain counting problems is essential for the full understanding of the underlying dynamical system's qualitative behavior. The most obvious counting problem is to determine (or estimate) the number of *attractors* of the dynamical system [10]; we shall consistently refer to these attractors and other stable configurations as to the *fixed points* throughout the rest of this technical report.[10] Counting the non-FP temporal cycles and cycle configurations, or the garden of Eden configurations, or the sizes of basins of attraction for different attractors, may also be of interest. For example, in *deterministic* discrete dynamical systems with finite configuration spaces that are temporal cycle-free, examples of which include the *asynchronous* discrete Hopfield networks with linear threshold update rules [81, 82], as well as the sequential CA and SDSs with monotone threshold update rules [68, 69, 198, 200, 205], the number of fixed points equals the number of possible dynamical evolutions of the system. In the context of Hopfield networks, the interpretation of the FP count is that it tells us how many distinct *patterns* a given Hopfield net can *memorize* [10, 81, 82]. In more general deterministic systems, the total number of possible evolutions equals the sum of the number of fixed points and the number of non-FP temporal cycles. Similarly, the number of GE configurations indicates what fraction of

[10] All types of configurations, such as the fixed points, cycle configurations, and gardens of Eden briefly discussed in this subsection will be formally defined in Sections 4 and 5.

all possible configurations are *unreachable* [17, 190].

It has been observed that determining the computational complexity of counting problems related to various *static* combinatorial structures, such as the CNF formulae or different types of graphs, tends to be difficult when compared to the complexity of the corresponding decision and search problems [73, 210, 211, 212]. Computational complexity of counting in *dynamic* structures, in general, can be only expected to be even harder. This is particularly the case with the dynamical systems with certain memory, sparseness and symmetry properties, due to the constraints that those properties may impose on how different system components interact with one another, which, in turn, may have considerable implications on how complex the resulting *collective dynamics* can be [196, 206]. Consequently, it is not surprising that complexity–theoretic results and characterizations of the fundamental enumeration problems pertaining to discrete dynamical systems are few and far between.

Insofar as what is known about the problems of enumerating various structures in discrete dynamical systems, there are precious few such results in the literature. We attribute that fact to two main factors. The first is the relative difficulty of either proving that a particular counting problem is hard, or actually providing an efficient counting algorithm for the problem in question. The second reason is that, in general, solving combinatorial type problems about discrete dynamical systems tends to be harder than solving the corresponding problems in the context of the ordinary graphs, Boolean formulae or other static objects. Hence, not surprisingly, most known results on counting different types of configurations or other structures of interest in discrete dynamical systems are essentially loose numerical estimates based on statistical sampling and extensive computer simulations, rather than analytically proven exact or approximate enumerations; see, e.g., [10, 81, 103, 105, 208, 218].

To the best of our knowledge, the only theoretical results on **#P**-completeness of some interesting (exact) counting problems about discrete dynamical systems are those by P. Floreen and P. Orponen in the context of *discrete Hopfield networks*. Namely, Floreen and Orponen prove in [55, 56] that the problems of (a) counting stable configurations (that is, fixed points) and (b) determining the sizes of these stable configurations' *basins of attraction* are **#P**-complete. However, the discrete Hopfield nets in the work of Floreen and Orponen are (i) *dense*, (ii) with negative weights $w_{ij}$ allowed, and also (iii) either *memoryless* (when the weights along the main diagonal satisfy $w_{ii} = 0$), or else with some negative weights along the main diagonal. In contrast, insofar as counting the fixed points of Boolean SDSs, SyDSs and CA is concerned, we will require that our restricted versions of CA and S(y)DSs be *with memory*, and, furthermore, we will not allow *inhibitory effects* (that would correspond to the negative weights in Hopfield networks). Last but not least, our main results in Section 6 hold for *uniformly sparse* network topologies, including the 3-regular graphs for the linear threshold SDSs and SyDSs similar to, and actually more restricted than, those considered in the Hopfield networks literature (subsection 6.2), as well as the 3-regular graphs for the symmetric Boolean SDSs and SyDSs (subsection 6.1). For all these reasons, we find our hardness results on complexity of counting in linear threshold and monotone S(y)DSs and discrete Hopfield networks to be considerably stronger than the comparable results on discrete Hopfield Nets found in [55, 56].

# 4 Parallel vs. Sequential Threshold Cellular Automata

In subsection 1.1, we have outlined several possible dimensions along which the classical CA can be generalized, so that the resulting extensions become more appropriate as abstract models for distributed computing in general, and large-scale MAS, in particular. The focus of the present Section will be on studying the implications of the classical CA parallel node updates' *perfect synchrony.* In Section 5, on the other hand, the assumptions about the classical CA homogeneity when it comes to both the individual agent behaviors and the inter-agent interaction patterns will be dropped, and a particular class of the resulting network automata and some of their interesting properties will be studied.

We now turn to the nature of local computations and inter-agent communication in the classical CA when it comes to the assumptions about (a)synchrony of both the local computations ("node updates") and interactions (i.e., how the change of state of one node affects the nearby nodes). Namely, not only does the parallel CA model allow for physically unrealistic unbounded parallelism, but the nodes also update *logically simultaneously.* In particular, a single parallel "step" of a cellular automaton with infinitely many nodes includes an infinite amount of local computation. A question, then, arises: what are the possible behaviors of a given type of CA once the perfect synchrony assumption is dropped?

In order for the CA model to become a more realistic abstraction for distributed computing, the nodes of a cellular automaton would need to locally update *asynchronously.* Moreover, this asynchrony should pertain not only to the local computations, but also to the inter-node *interaction* (that is, communication). While one can find in the literature some examples of the CA models with the asynchronous local computations – indeed, physicists have been exploring such cellular automata models since the 1980s – we have not found a single model anywhere where the asynchrony applies to *both* local computations and communication. We find it appropriate to refer to the models where the communication among nodes is still synchronized as *sequential cellular automata* (SCA), while reserving the name of *(genuinely) asynchronous cellular automata* (ACA) for those cellular automata models where both local computations and inter-agent communication are asynchronous. It is the models of this, latter type, that are of the main interest in the large-scale distributed computing context. However, the sequential models are also interesting and useful, and, indeed, can be viewed as a first step towards the realistic, genuinely asynchronous models. The formal definition of both sequential and genuinely asynchronous CA will follow in the subsequent subsections of this Section. For the time being, we just make an observation that, as one would intuitively expect, the genuinely asynchronous CA models, where asynchrony pertains to *communication*, would be expected to subsume the sequential models where asynchrony pertains to local computation but not communication. Indeed, the SCA we shall define in subsection 4.2 can be formally shown to be a special case of the genuinely asynchronous model (ACA) that we will define later on in this Section, namely, in subsection 4.5.

The rest of this Section will be devoted to considering the sequential version of CA, called SCA, and comparing these SCA with the classical, *parallel* CA. However, if *arbitrary* node update rules (that is, local agent behaviors) are allowed, almost every interesting global long-term behavior of CA becomes intractable, in the finite cases, or outright formally undecidable, in the infinite cases. Therefore, we will compare and contrast parallel vs. sequential CA in a very restricted, yet nontrivial and interesting context. In particular, we will show that there are parallel 1-D

CA with very simple node state update rules that cannot be simulated by any comparable SCA, irrespective of the node update ordering. Consequently, one possible interpretation, in terms of the theory of concurrency, is that the fine granularity of the basic CA operations and, therefore, the *fine-grain parallelism* of the classical, perfectly synchronous CA, insofar as the "interleaving semantics" paradigm is concerned, is simply *not fine enough*.

While interesting in its own right, this comparative analysis has an important additional purpose: to motivate introducing the aforementioned *genuinely asynchronous cellular automata* and rigorously studying their properties, and to investigate those properties' implications for the large-scale distributed computing in general, and *massive multi-agent systems*, in particular.

### 4.1 Problem Motivation

*Cellular automata* (CA) were originally introduced as an abstract mathematical model that can capture the behavior of biological systems capable of self-reproduction [134]. Subsequently, CA have been extensively studied in a great variety of application domains, mostly in the context of complex physical or biological systems and their dynamics (e.g., [69, 70, 224, 225, 226, 227]). However, CA can also be viewed as an abstraction of massively parallel computers (e.g, [63]). We study in this Section a particular simple yet nontrivial class of CA from the parallel and distributed computing perspectives. In particular, we pose – and partially answer – some fundamental questions regarding the nature of the CA parallelism, i.e., the perfect synchrony of the classical CA computation.

Cellular Automata are considered an abstract architecture model of *fine-grain parallelism*, in that the elementary operations executed at each node are rather simple and hence comparable to the basic operations performed by the computer hardware. In a classical (parallel) CA, all the nodes execute their operations in parallel and *in perfect synchrony*, that is, *logically simultaneously*: in general, the state of a node $x_i$ at time step $t+1$ is some simple function of the states of the node $x_i$ and a set of its pre-specified neighbors at time $t$.

We consider herewith the sequential version of CA, that we shall abridge to SCA in the sequel, and compare these SCA with the perfectly synchronous *parallel* (or *concurrent*) CA. In particular, we will show that there are 1-D CA with very simple state update rules that cannot be simulated by any comparable SCA, irrespective of the node update ordering.

We also share some thoughts on how to extend the results to be presented in the sequel, and, in particular, we try to motivate the study of *genuinely asynchronous cellular automata* where the asynchrony applies not only to the local computations at individual nodes, but also to the *communication* among different nodes via the "shared variables" stored as the respective nodes' states.

An example of asynchrony in the local node updates (i.e., the asynchronous computation at different "processors") is when, for instance, the individual nodes update one at a time, according to some random order. This is a kind of asynchrony found in the literature, e.g., in [89, 104]. It is important to understand, however, that even in case of what is referred to as *asynchronous cellular automata* (ACA) in the literature, the term *asynchrony* there applies to local updates (i.e., computations) *only*, but not to communication, since a tacit assumption of the globally accessible global clock still holds. We prefer to refer to this kind of (weakly asynchronous) CA as *sequential cellular automata*, and, in this work, consistently keep the term *asynchronous cellular*

*automata* for those CA that do not have a global clock.

Before dwelling into the issue of concurrency vs. arbitrary sequential interleavings applied to the threshold cellular automata, we first clarify the terminology, and then introduce the relevant concepts through a simple programming exercise in subsubsection 4.1.1.

Throughout, we use the terms *parallel* and *concurrent* as synonyms. Many programming languages experts would strongly disagree with this convention. However, a complete agreement in the computer science community on what exactly *concurrency* means, and how it relates to *parallelism*, is lacking. According to *Section §12* of [164], "concurrency in the programming language and parallelism in the computer hardware are independent concepts. [...] We can have concurrency in a programming language without parallel hardware, and we can have parallel execution without concurrency in the language. In short, *concurrency refers to the potential for parallelism*" (italics ours). Clearly, our convention herein does not conform to the notions of concurrency and parallelism as defined in [164]. In contrast, [149] uses the term *concurrent* "to describe computations where the simultaneously executing processes can interact with one another", and *parallel* for "[...] computations where behavior of each process is unaffected by the behavior of the others". [149] also acknowledges that many authors do not discriminate between *parallel* and *concurrent*. We shall follow this latter convention throughout and, moreover, by a *parallel (concurrent) computation* we shall mean actions of several processing units that are carried out *logically* (if not necessarily *physically*) *simultaneously*.

### 4.1.1 Capturing Concurrency by Sequential Interleavings

While our own brains are massively parallel computing devices, we seem to (consciously) think and approach problem-solving rather sequentially. In particular, when designing a parallel algorithm or writing a computer program that is inherently parallel, we still prefer to be able to understand such an algorithm or program at the level of sequential operations or executions. It is not surprising, therefore, that a great deal of research effort has been devoted to interpreting parallel computation in the more familiar, sequential terms. One of the most important contributions in that respect is the (nondeterministic) sequential *interleaving semantics* of concurrency (see, e.g., [42, 64, 80, 123, 124]).

When interpreting concurrency via interleaving semantics, a natural question arises: *Given a parallel computing model, can its parallel execution always be captured by such sequential nondeterminism, so that any given parallel computation can be faithfully reproduced via an appropriate choice of a sequential interleaving of the operations involved?* For most theoreticians of parallel computing[11], the answer is apparently "YES" – provided that we simulate concurrent execution via sequential interleavings at a sufficiently high level of granularity of the basic computational operations. However, given a parallel computation in the form of a set of concurrently executing processes, how do we tell if the particular level of granularity is *fine enough*, i.e., whether the operations at that granularity level can truly be rendered *atomic* for the purposes of capturing concurrency via sequential interleavings?

We shall illustrate the concept of *sequential interleaving semantics* of concurrency with a simple example. Let's consider the following trivia question from a sophomore parallel programming

[11]That is, for all "believers" in the interleaving semantics of concurrency - as contrasted with, e.g., proponents of *true concurrency*, an alternative model not discussed herewith.

class: *Find a simple example of two instructions such that, when executed in parallel, they give a result not obtainable from any corresponding sequential execution sequence?*

*A possible answer:* Assume $x = 0$ initially and consider the following two programs

$$x \leftarrow x + 1;\ x \leftarrow x + 1$$

vs.

$$x \leftarrow x + 1 \ ||\ x \leftarrow x + 1$$

where "||" stands for the parallel, and ";" for the sequential composition of instructions or programs, respectively. Sequentially, one *always* gets the same answer: $x = 2$. In parallel (when the two assignment operations are executed synchronously), however, one gets $x = 1$. It appears, therefore, that no sequential ordering of operations can reproduce parallel computation – at least not at the granularity level of high-level instructions as above.

The whole "mystery" can be readily resolved if we look at the possible sequential executions of the corresponding machine instructions:

| | |
|---|---|
| LOAD $x, *m$ | LOAD $x, *m$ |
| ADD $x, \#1$ | ADD $x, \#1$ |
| STORE $x, *m$ | STORE $x, *m$ |

There certainly exist choices of *sequential interleavings* of the six machine instructions above that lead to *"parallel"* behavior (i.e., the one where, after the code is executed, $x = 1$). Thus, by refining granularity from the high-level language instructions down to the machine instructions, we can certainly preserve the interleaving "semantics" of concurrency.

As a side, we remark that it turns out that the example above does not require finer granularity quite yet, if we allow that some instructions be treated as *no-ops*. Indeed, if we define $\Phi(P)$ to be the *set of possible behaviors of program P*, then the example above only shows that, for $S_1 = S_2 = (x \leftarrow x + 1)$,

$$\Phi(S_1||S_2) \ \not\subseteq\ \Phi(S_1; S_2) \ \cup\ \Phi(S_2; S_1) \tag{1}$$

However, it turns out that, in this particular example, it indeed is the case that

$$\Phi(S_1||S_2) \ \subseteq\ \Phi(S_1; S_2) \ \cup\ \Phi(S_2; S_1) \ \cup\ \Phi(S_1) \ \cup\ \Phi(S_2) \tag{2}$$

and no finer granularity is necessary to model $\Phi(S_1||S_2)$, assuming that, in some of the sequential interleavings, we allow certain instructions not to be executed at all.

However, one can construct more elaborate examples where the property (2) above does not hold. The only way to capture the program behavior of parallel compositions of the form $\Phi(P_1||P_2)$ via sequential interleavings in such cases would then be to find a finer level of granularity, i.e., to reconsider at what level can operations be considered *atomic*, so that the union of all possible sequential interleavings of such basic operations, *including those interleavings that allow "no-ops" for some of the instructions*, is guaranteed to capture the concurrent behavior – that is, in order for the relationship (2) to hold. In other words, sometimes *refining the granularity of operations*, so that sequential interleavings can capture synchronous parallel behavior, becomes a *necessity.*

We address in this Section the (in)adequacy of the sequential interleavings semantics when applied to CA where the individual node updates[12] are considered to be elementary operations. In particular, we show that the perfect synchrony of the classical CA's node updates causes the interleaving semantics, as captured by the SCA and NICA sequential CA models (subsection 4.2), to fail rather dramatically even in the context of the simplest (nonlinear) CA node update rules.

## 4.2 Parallel and Sequential Cellular Automata and Their Configurations

We will introduce CA by first considering a kind of *(deterministic) Finite State Machines* (FSMs) that receive input but produce no output, and that are also known as *Deterministic Finite Automata* (DFA). An FSM has finitely many states, and is capable of reading the input signals coming from the outside and, based on its current state as well as the currently read input, of changing its state. Namely, the machine is initially in some starting state; upon reading each input signal, the machine changes its state according to a pre-defined and fixed rule. In particular, the entire memory of the system is contained in what *current state* the machine is in, and nothing else about the previously processed inputs is remembered. Hence, the probabilistic generalization of deterministic FSMs leads to (discrete) Markov chains. It is important to notice that there is no way for a FSM to overwrite, or in any other way affect, the input data stream. Thus *individual* FSMs are computational devices of rather limited power.

Now let us consider many such FSMs, all identical to one another, that are lined up together in some regular fashion, e.g., on a straight line or a regular 2-D grid, so that each single *node* in the grid is connected to its immediate neighbors. Let's also eliminate any external sources of input streams to the individual machines at the nodes, and let the current values of any given node's neighbors be that node's only *input data*. If we then specify a finite set of the possible values held in each node, and we also identify this set of values with the set of each node's *internal states*, we arrive at an informal definition of a classical cellular automaton. To summarize, a cellular automaton is a finite or infinite regular grid in one-, two- or higher-dimensional space, where each node in the grid is a FSM, and where each such node's input data at each time step are the corresponding internal states of the node's neighbors. Moreover, in the most important special case – the Boolean case, this FSM is particularly simple, i.e., it has only two possible internal states, labeled 0 and 1. All the nodes of a classical CA execute the FSM computation in unison, i.e., *(logically) simultaneously*. We note that infinite CA are capable of universal (Turing) computation. Moreover, the general class of infinite CA, once arbitrary starting configurations are allowed, are actually *strictly more powerful* than the classical Turing machines – in particular, infinite CA are capable of solving the so-called *dynetic problems* that may not be Turing-computable. The subject of computation beyond the Turing limit is beyond the scope of this technical report, but for more on dynetic problems and related matters an interested reader is referred to, e.g., *Section 5* of [63] .

We will follow [63] and formally define classical (that is, synchronous and concurrent) CA in two steps: we first define the notion of a *cellular space*, and, subsequently, we define a *cellular*

[12]It is tacitly assumed here that a complete node update operation, in addition to computing the local update function on appropriate inputs, also includes the necessary *reads* of the neighbors' values preceding the local rule computation, as well as the *writes* of one's new value following the local computation. These points will become clear once the necessary definitions and terminology are introduced in subsection 4.2.

*automaton* over an appropriate cellular space.

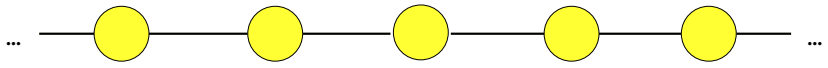
**Definition 4.1.** *A Cellular Space,* $\Gamma$*, is an ordered pair* $(G, Q)$*, where*

– *$G$ is a regular undirected Cayley graph that may be finite or infinite, with each node labeled with a distinct integer; and*

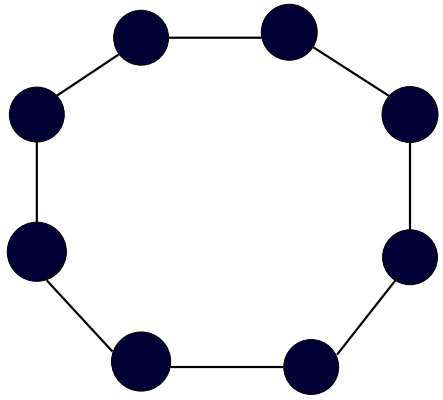
– *$Q$ is a finite set of states that has at least two elements, one of which being the special quiescent state, denoted by* 0*.*

We denote the set of integer labels of the nodes (vertices) in $\Gamma$ by $L$. That is, $L$ may be equal to, or be a proper subset of, the set of all integers.

Two kinds of *one-dimensional* (1D) cellular spaces most popular in the CA literature are (finite as well as infinite) *line graphs* and finite rings; see *Figure* 1.



**One–dimensional cellular space: infinite line**



**One–dimensional cellular space: finite ring (circular boundary conditions)**

Figure 1: One-dimensional cellular spaces: an infinite line graph (top) and a finite ring (bottom)

**Definition 4.2.** *A* Cellular Automaton (CA) ***A*** *is an ordered triple* $(\Gamma, N, M)$*, where*

– $\Gamma$ *is a* cellular space*;*

– $N$ *is a* fundamental neighborhood*; and*

– $M$ *is a* finite state machine *such that the input alphabet of* $M$ *is* $Q^{|N|}$*, and the local transition function (update rule) for each node is of the form* $\delta : Q^{|N|+1} \to Q$ *for the CA* with memory*, and* $\delta : Q^{|N|} \to Q$ *for the* memoryless CA*.*

The fundamental neighborhood $N$ specifies what nearby nodes provide inputs to the update rule of a given node. In the classical CA, $\Gamma$ is a regular graph that locally "looks the same everywhere"; in particular, the local neighborhood $N$ is the same for each node in $\Gamma$.

The local transition rule $\delta$ specifies how each node updates its state (that is, value), based on its current state (value), and the current states of its neighbors in $N$. By composing together the application of the local transition rule to each of the CA's nodes, we obtain *the global map* on the set of (global) configurations of a cellular automaton.

We observe that there is plenty of parallelism in the CA "hardware", assuming, of course, a sufficiently large number of nodes[13]. Actually, classical CA defined over infinite cellular spaces provide *unbounded parallelism* where, in particular, an infinite amount of information processing is carried out in a finite time – in fact, even in a single parallel step. In particular, the notion of independence between *parallelism* and *concurrency* as defined in [164] seems inappropriate to apply to CA: without the parallel *hardware*, that is, multiple interconnected nodes, a CA is not capable of *any* concurrent computation. Indeed, a single-node CA is just a *fixed* deterministic finite state machine – an entirely sequential computing model.

Insofar as the CA *computer architecture* is concerned, one important characteristic is that the memory and the processors are not truly distinguishable, in stark contrast to *Turing machines*, *(P)RAMs*, and other standard abstract models of digital computers. Namely, each node of a cellular automaton is both a processing unit and a memory storage unit; see, e.g., the detailed discussion in [187]. In particular, the only *memory content* of a cellular automaton is a tuple of the (current) states of all its nodes. Moreover, as a node can *read* (but not *write*) the states or "values" of its neighbors, we can view the architecture of classical CA as a very simplistic, special case of *distributed shared memory* parallel model, where every *processor* (that is, each node) "owns" one cell of its *local memory*, physically separated from other similar local memories – yet this local memory is *directly accessible* (for the *read* accesses) to some of the other *processors*. In particular, the *reads* to any *memory cell* (or equivalently a *shared variable* stored in such a memory cell) are restricted to an appropriate neighborhood of that shared value's "owner processor", while the *writes* are restricted to the owner processor *alone.*

Since our main results in this Section pertain to a comparison and contrast between the classical, concurrent threshold CA and their sequential counterparts, we formally introduce two types of sequential CA next. First, we define SCA with a *fixed* (but arbitrary) sequence specifying the order according to which the nodes are to update. We then introduce a kind of sequential automata whose purpose is to capture the *interleaving semantics*, that is, where *all* possible sequences of node updates are considered at once.

**Definition 4.3.** *A* Sequential Cellular Automaton (SCA) $\boldsymbol{S}$ *is an ordered quadruple* $(\Gamma, N, M, s)$, *where* $\Gamma$, $N$ *and* $M$ *are as in* Definition 4.2, *and* $s$ *is an arbitrary sequence, finite or infinite, all of whose elements are drawn from the set* $L$ *of integers used in labeling the vertices of* $\Gamma$. *The sequence* $s$ *is specifying the sequential ordering according to which the SCA's nodes update their states, one at a time.*

However, when comparing and contrasting the concurrent CA with their sequential counterparts, rather than making a comparison between a given parallel CA with a *particular* SCA (that is, a corresponding SCA with some particular choice of the update sequence $s$), we compare the parallel CA computations with the computations of the corresponding SCA for *all* possi-

[13] See the discussion in subsection 4.1, and, in particular, the definition of the relationship between concurrency and parallelism in reference [164].

ble sequences of node updates. For that purpose, the following class of sequential automata is introduced:

**Definition 4.4.** *A* Nondeterministic Interleavings Cellular Automaton (NICA) ***I*** *is defined to be the union of all sequential automata* ***S*** *whose first three components,* $\Gamma, N$ *and* $M$ *are fixed. That is,* $I = \cup_s (\Gamma, N, M, s)$*, where the meanings of* $\Gamma, N, M$*, and* $s$ *are the same as before, and the union is taken over* all *(finite and infinite) sequences* $s : \{1, 2, 3, ...\} \to L$*, where* $L$ *is the set of integer labels of the nodes in* $\Gamma$*.*

### 4.2.1 Types of Cellular Automata Configurations

We now change the pace and introduce some terminology from physics that we find useful for characterizing *all possible computations* of a parallel or a sequential cellular automaton. To this end, a *(discrete) dynamical systems* view of CA is helpful.

A *configuration space* or *phase space* of a dynamical system is a directed graph where the vertices are the *global configurations* (or *global states*) of the system, and directed edges correspond to the possible direct transitions from one global state to another.

As for any other kind of dynamical systems, we can define the fundamental, qualitatively distinct types of global configurations that a cellular automaton can find itself in. We first define these qualitatively distinct types of dynamical system configurations for the parallel CA and then briefly discuss how these definitions need to be modified in case of SCA and NICA.

The classification below is based on answering the following question: starting from a given global CA configuration, can the automaton return to that same configuration after a finite number of parallel computational steps?

**Definition 4.5.** *A* fixed point (FP) *is a configuration in the phase space of a CA such that, once the CA reaches this configuration, it stays there forever. A* cycle configuration (CC) *is a state that, once reached, will be revisited infinitely often with a fixed, finite temporal period of 2 or greater. A* transient configuration (TC) *is a state that, once reached, is never going to be revisited again.*

In particular, a fixed point is a special, degenerate case of a recurrent state with period 1. We remark that *recurrent configurations* are also sometimes simply referred to as *cyclic* in the literature; however, we shall consistently make the distinction between the fixed points on one hand, and the *proper* cycle configurations, namely, those recurrent configurations that are not FPs, on the other. Due to deterministic evolution, a configuration of a classical, parallel CA is either a FP, a *proper* CC, or a TC.

On the other hand, if one considers SCA so that *arbitrary* node update orderings are permitted, then, given the underlying cellular space and the local update rule, the resulting phase space configurations, due to nondeterminism that results from different choices of possible sequences of node updates ("sequential interleavings"), are more complicated. In a particular SCA, a cycle configuration is any configuration revisited infinitely often – but the period between different consecutive visits, assuming an arbitrary sequence $s$ of node updates, need not be fixed. We call a global configuration that is revisited only finitely many times (under a given ordering $s$)

*quasi-cyclic.* Similarly, a *quasi-fixed point* is an SCA configuration such that, once the SCA's dynamic evolution reaches this configuration, it stays there "for a while" (i.e., for some finite number of sequential node update steps), and then leaves. For example, a configuration of an SCA can simultaneously be both an FP and a quasi-CC, or both a quasi-FP and a CC.

For simplicity, heretofore we shall refer to a configuration $\mathcal{C}$ of a NICA as a *(weak) fixed point* if there exists some infinite sequence of node updates $s$ such that $\mathcal{C}$ is a FP in the usual sense when the automaton's nodes update according to the ordering $s$. A *strong fixed point* of a NICA automaton is a configuration that is fixed (stable) with respect to *all* possible sequences of node updates. Similarly, we consider a configuration $\mathcal{C}'$ to be a cycle state, if there exists an infinite sequence of node updates $s'$ such that, if the NICA's nodes update according to $s'$, then $\mathcal{C}'$ is a cycle state of period 2 or greater in the usual sense (see Definition 4.5). In particular, in case of the NICA automata, a single configuration can simultaneously be a weak FP, a CC and a TC; see subsubsection 4.3.1 for a simple example.

## 4.3 1-D Simple Threshold Parallel vs. Sequential CA: Comparison and Contrast

After the introduction, motivation and the necessary definitions, we now proceed with our main results and their meaning. Technical results (and some of their proofs) are given in this subsection. Discussion of the implications and relevance of these results, as well as some possible generalizations and extensions, will follow in subsection 4.5.

We shall compare and contrast the classical, concurrent CA with their sequential counterparts, SCA and NICA, in the context of the simplest nonlinear local update rules possible, namely, the CA in which the nodes locally update according to *linear threshold functions* [68, 69, 205]. Moreover, we will choose those threshold functions to be *symmetric*, so that the resulting (S)CA will be also *totalistic* (see, e.g., [63] or [226]). We will show the fundamental difference in the configuration spaces, and therefore possible computations, between the parallel threshold cellular automata and the sequential threshold automata: while the former can have temporal cycles (of length two), the computations of the latter always either converge to a fixed point, or otherwise they fail to finitely converge to any recurrent pattern whatsoever.

Second, we will also fully characterize all the possible configurations, as well as their relative frequencies, for the 1-D CA with short-range interactions that update according to perhaps the single most interesting totalistic threshold rule, namely, the MAJORITY function.

For simplicity, but also in order to indicate how dramatically the sequential interleavings of NICA fail to capture the concurrency of the classical CA based on perfect synchrony, we restrict the underlying cellular spaces to *one-dimensional* $\Gamma$. We formally define the class of 1-D (S)CA of a finite radius below:

**Definition 4.6.** *A 1-D (sequential) cellular automaton of radius $r$ ($r \geq 1$) is a (S)CA defined over a one-dimensional string of nodes, such that each node's next state depends on the current states of its neighbors to the left and right that are no more than $r$ nodes away. In case of the (S)CA with memory, the next state of a node also depends on the current state of that node itself.*

Thus, in case of a Boolean (S)CA *with memory* defined over a one-dimensional cellular space $\Gamma$, each node's next state depends on exactly $2r+1$ input bits, while in the *memoryless (S)CA case,* the local update rule is a function of $2r$ input bits. The underlying 1-D cellular space is a string of nodes that can be a finite line graph, a ring (corresponding to the "circular boundary conditions"), a one-way infinite string, or, in the most common case, $\Gamma$ is a two-way infinite string (or "line").

We establish the following conventions and terminology. Throughout this Section, only *Boolean* CA, SCA and NICA are considered; in particular, the set of possible states of any node is $\{0, 1\}$. Also, due to a *terminological diversity* found in the literature, and the fact that, originally, different pieces of our work as summarized in this technical report have appeared in different kinds of venues and with different audiences in mind, we warn the reader that several important concepts used in this Section and, indeed, throughout the technical report, are referred to by two or, in a couple of instances, even more different names. The most notable examples of this "polymorphism" are listed below. The phrases "monotone symmetric" and "simple (linear) threshold" functions/update rules/automata are used interchangeably. Similarly, "(global) dynamics" and "(global) computation", when applied to any kind of automata, are used synonymously. Unless stated otherwise, *CA* denotes a classical, concurrent cellular automaton, whereas a cellular automaton where the nodes update sequentially is always denoted by *SCA* (or *NICA*, when appropriate). Also, unless explicitly stated otherwise, *(S)CA with memory* are assumed. The default infinite cellular space $\Gamma$ is a two-way infinite line. The default finite cellular spaces are finite rings. The terms *phase space* and *configuration space* are used synonymously throughout, as well, and sometimes abridged to *PS* for brevity.

### 4.3.1 Synchronous CA vs. Sequential Interleavings CA: An Example

There are many simple, even trivial examples where not only are concrete computations of the *parallel* CA from particular initial configurations different from the corresponding computations of any of the *sequential* CA, but actually the entire configuration spaces of the parallel cellular automata on the one, and the corresponding SCA and NICA on the other hand, turn out to be rather different.

As one of the simplest examples conceivable, consider a trivial CA with more than one node (so that talking about *parallel computation* makes sense), namely, a two-node CA where each node computes the logical *XOR* of its two inputs. The two phase spaces are given in *Figure* 2.

In the example in *Figure* 2, each node computes the logical *XOR* function of its own current state, and that of the other node. In particular, each node's update rule is Boolean XOR function of two inputs. In (b), the integer labels next to the transition arrows indicate which node, $v_1$ or $v_2$, is updating and thus causing the indicated two-node joint state transition.

In the parallel case, the state 00 is the "sink", and the entire configuration space is as in *Figure* 2 *(a)*. So, regardless of the starting configuration, after at most two parallel steps, a fixed point "sink" state, that is, in physics terms, a stable global attractor, will be reached.

Insofar as the sequential node updates are concerned, the configuration 00 is still a FP but, this time, it is not reachable from any other configuration. Also, while all three states, 11, 10 and 01, are *transient states* in the parallel case, sequentially, each of them, for a *typical* (infinite) sequence of node updates, is going to be revisited *infinitely often*. In fact, for some sequences

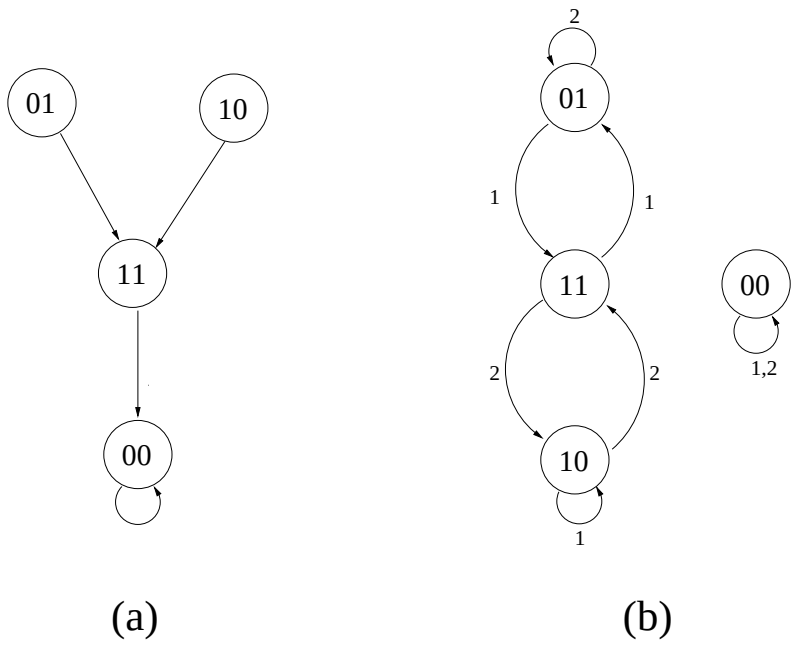



Figure 2: Configuration spaces for the two-node (a) parallel and (b) sequential cellular automata with $\delta = XOR$, respectively

of node updates such as, e.g., $(1, 1, 2, 2, 2, 1, 2, 2, 1, ...)$, configurations 01 and 10 are *both* quasi-fixed-point states and cycle states.

The phase space capturing all possible sequential computations of the two-node automaton with $\delta = XOR\{x_1, x_2\}$ for each node is given in *Figure* 2 *(b)*. This NICA has three configurations, 01, 10 and 11, each of which is simultaneously a weak FP, a CC and a TC; it is a trivial exercise to find particular update sequences for which each of these configurations is of a desired nature (weak FP, CC or TC). In contrast, configuration 00 is a FP for *every* sequence of node updates[14].

Some observations are in order. First, overall, the configuration space of the *XOR* NICA is richer than the configuration space of its parallel counterpart. In particular, due to determinism, any FP state of a parallel CA is necessarily a stable attractor. In contrast, in case of different possible sequential computations on the same cellular space, the (weak) fixed points clearly need not be stable. Also, whereas the phase space of a parallel CA is temporal cycle-free (recall that we do not count FPs among temporal cycles), the phase space of the corresponding NICA has nontrivial finite temporal cycles. On the other hand, the union of all possible sequential computations ("interleavings") cannot fully capture the concurrent computation, either: consider, for example, the *(un)reachability* of the state 00, which, in the sequential case, clearly depends on the *initial state*.

[14]In [200] we refer to such FPs of NICA as *proper* or *strong* fixed points, in order to contrast them with respect to those configurations that are fixed with respect to *some* but not *all* sequences of the node updates. We also remark that, in a given computation, if the starting configuration of this NICA, or any corresponding SCA, is different from 00, then this FP configuration is also an example of a *Garden of Eden* (GE) configuration, as it cannot ever be reached irrespective of the sequence $s$ of node updates. For more on GEs in discrete dynamical systems, the reader is referred to [16, 17, 194].

All these properties can be largely attributed to a relative complexity of the *XOR* function as the update rule, and, in particular, to *XOR*'s *non-monotonicity.* They can also be attributed to the idiosyncrasy of the example chosen. In particular, temporal cycles in the sequential case are not surprising. Also, if one considers CA on say four nodes with circular boundary conditions (that is, a CA ring on four nodes), these *XOR* CA do have nontrivial temporal cycles in the parallel case, as well. Hence, for the *XOR* CA with sufficiently many nodes, the types of computations that the parallel CA and the sequential SCA and NICA are capable of, are quite comparable. Moreover, in those cases where one class is of a richer behavior than the other, it seems reasonable that the NICA automata, overall, are capable of more diverse computations than the corresponding synchronous, parallel CA, given the nondeterminism of NICA arising from all different possibilities for the node update sequences.

This detailed discussion of a rather straightforward example of the CA and NICA phase spaces has the main purpose of motivating what is to follow: an entire class of CA and SCA/NICA, with the node update functions simpler than *XOR*, yet for which it is the concurrent CA that are *provably* capable of a kind of computations that no corresponding (or similar, in the sense to be discussed in subsubsection 4.3.3 and subsection 4.5) SCA and, consequently, NICA, are capable of.

### 4.3.2 Linear Threshold and Simple Threshold Cellular Automata

We shall now compare and contrast the classical, that is, parallel and perfectly synchronous CA, with their sequential counterparts, SCA and NICA. The comparison and contrast will be done in the context of the simplest nonlinear local update rules possible, namely, the CA in which the nodes locally update according to a restricted kind of *linear threshold functions.* This will be done by studying the configuration space properties, that is, the possible computations, of the *simple* threshold cellular automata in both parallel and sequential settings.

First, we define *(simple) linear threshold functions*, and the corresponding types of (S)CA.

**Definition 4.7.** *A Boolean-valued* linear threshold function *of $m$ inputs, $x_1, ..., x_m$, is any function of the form*

$$f(x_1, ..., x_m) \;=\; \begin{cases} 1, & \text{if} \quad \sum_i w_i \cdot x_i \;\geq\; \theta \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

*where $\theta$ is an appropriate* threshold constant*, and $w_1, ..., w_m$ are arbitrary (but fixed) real numbers*[15] *called* weights*.*

**Definition 4.8.** *A* threshold (sequential) cellular automaton (T(S)CA) *is a parallel (alternatively, sequential) cellular automaton whose local update rule $\delta$ is a* Boolean-valued linear threshold function*.*

[15] In general, $w_i$ can be both positive and negative. This is especially common in the neural networks literature, where negative weights $w_i$ indicate an *inhibitory effect* of, e.g., one neuron on the firings of another, nearby neuron. In most studies of discrete dynamical systems, however, the weights $w_i$ are required to be nonnegative – that is, only *excitatory effects* of a node on its neighbors are allowed; see, e.g., [16, 17, 224, 225]. One consequence of insisting that all weights of a linear threshold function be positive is that the resulting function is then also *monotone* in the usual theory of Boolean functions sense [217].

Therefore, given an integer $k$, a *k-threshold function*, in general, is a Boolean-valued function of the form as in Definition 4.7 with $\theta = k$ and an appropriate choice of weights $w_i$, where $i = 1, ..., m$. Heretofore we consider *monotonically nondecreasing* Boolean threshold functions only; this, in particular, implies that the weights $w_i$ are always nonnegative. We also additionally assume $\delta$ to be a *symmetric function* of all of its inputs. (In particular, if all the weights $w_i$ are positive and equal to one another, then, without loss of generality, we may set them all equal to 1; obviously, this normalization of the weights $w_j$ may also require an appropriate adjustment of the threshold value $\theta$.)

That is, the (S)CA we analyze have *symmetric, monotone Boolean functions* for their local update rules. We refer to such functions as to *simple threshold functions*, and to the (S)CA with simple threshold node update rules as to *simple threshold* (S)CA.

**Definition 4.9.** *A* simple *threshold (S)CA is a (sequential) cellular automaton whose local update rule $\delta$ is a monotone symmetric Boolean (threshold) function.*

**Examples:** We provide some examples of (i) simple threshold functions and (ii) linear threshold functions that *are not simple*. Both kinds of functions clearly belong to the class of (arbitrary) linear threshold functions. Examples of simple threshold functions are Boolean AND, Boolean OR, and the MAJORITY function on an appropriate number of inputs. For example, the Boolean AND on three inputs can be written in the "simple threshold form" as

$$And(x_1, x_2, x_3) \;=\; \begin{cases} 1, & \text{if } \; x_1 + x_2 + x_3 \geq 3 \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

An example of a function on three Boolean-valued inputs that is linear threshold, but not simple threshold, is given by

$$f(x_1, x_2, x_3) \;=\; \begin{cases} 1, & \text{if } \; 2x_1 + x_2 + x_3 \geq 3 \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

Clearly, the dependence of function $f$ on variable $x_1$ is different from the dependence on variables $x_2$ and $x_3$. We observe that the function $g$ below, written in a "non-simple threshold" way,

$$g(x_1, x_2, x_3) \;=\; \begin{cases} 1, & \text{if } \; 3x_1 + 2x_2 + x_3 \geq 1 \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

is actually simple threshold, as the actual dependence on all three variables is the same; in fact, $g(x_1, x_2, x_3) = OR$, and it can be written as

$$g(x_1, x_2, x_3) \;=\; \begin{cases} 1, & \text{if } \; x_1 + x_2 + x_3 \geq 1 \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

Thus, in particular, one should think of simple threshold functions as those Boolean-valued functions of an appropriate number of Boolean variables that *can* be written in the form given in Definition 4.7, with the additional requirement that all weights $w_i$ be equal to one another.

Needless to say, every mathematical function has an infinite number of different *representations*. However, it is the essence of the function itself (i.e., what is the actual dependence of the output(s) on the input(s)), and not of its one particular representation or another, that we are interested in, and with respect to which we want to classify the update rules of cellular and network automata of our interest.

Throughout, whenever we say a *threshold automaton* or a *threshold* (S)CA, we shall by default mean *a simple threshold automaton* (simple threshold (S)CA)), unless explicitly stated otherwise. That is, the 1-D threshold (S)CA studied in the rest of this Section will have the node update functions of the general form

$$\delta(x_{i-r}, x_{i-r+1}, ..., x_i, ..., x_{i+r-1}, x_{i+r}) \;=\; \begin{cases} 1, & \text{if} \quad \sum_{j=-r}^{r} \; x_{i+j} \quad \geq \quad k \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

where $k$ is a fixed integer from the range $\{0, 1, ..., 2r+1, 2r+2\}$. For example, if the automaton rule radius is $r = 2$, and if $k = 2$, then a *k-threshold* (S)CA with a specified number of nodes is a 1-D (S)CA with the node update rule $\delta =$ *"at least 2 out of 5"*, meaning that the update rule evaluates to 1 if and only if at least two out of five of its inputs are currently equal to 1.

For our purposes, the most important example of a simple threshold function as a cellular or network automaton update rule, beside the Boolean AND and OR functions, is the Boolean-valued MAJORITY function:

**Definition 4.10.** *The Boolean* MAJORITY *function, denoted $\delta = MAJ$, is a simple threshold function on an arbitrary odd number of inputs $2r + 1$ given by*

$$\delta(x_{i-r}, x_{i-r+1}, ..., x_i, ..., x_{i+r-1}, x_{i+r}) \;=\; \begin{cases} 1, & \textit{if} \quad \sum_{j=-r}^{r} \; x_{i+j} \quad \geq \quad r+1 \\ 0, & \textit{otherwise} \end{cases} \tag{9}$$

The definition of $\delta = MAJ$ can be extended to even arities if one specifies how to break the tie when the number of zeros and the number of ones among the input variables are equal. In the sequel, we shall refrain from applying the $MAJ$ update rule so that the number of inputs is even. That is, we will consider CA with $\delta = MAJ$ so that the neighborhood size is always odd. In the most frequent case, the underlying cellular space will be either a one-dimensional (possibly one-way or two-way infinite) line, or a finite ring, and the rule radius $r \geq 1$ will mean that the update rule at each node is a Boolean function of exactly $2r + 1$ inputs,[16] so that the problem of applying the $MAJ$ rule to an even number of nodes' values does not arise.

### 4.3.3 On the Existence of Temporal Cycles

Due to the nature of the node update rules, cyclic behavior intuitively should not be expected from the simple threshold cellular automata. This is, generally, (almost) the case, as will be shown below. We argue that the importance of the results in this subsubsection largely stems from the following three factors:

[16] We again remind the reader that, unless we specifically say otherwise, we consider CA *with memory*, so that a node's current state is one of the inputs to that node's update rule.

– the local update rules are the simplest nonlinear *totalistic* (that is, symmetric) rules one can think of;

– given the rules, the cycles are not to be expected – yet they exist, and in the case of parallel (i.e., synchronous) CA *only*; and, related to that observation,

– it is, for this class of cellular automata, the parallel CA that have the more diverse possible dynamics, and, in particular, while qualitatively there is nothing among the possible sequential computations that is not present in the parallel case, the classical parallel threshold CA do exhibit a particular qualitative behavior – they may have nontrivial temporal cycles – that cannot be reproduced by any threshold SCA.

The results below hold for the two-way infinite 1-D (S)CA, as well as for the finite (S)CA with the circular boundary conditions (i.e., for the (S)CA whose cellular spaces are finite rings). We remind the reader that, unless explicitly stated otherwise, all the CA/SCA/NICA in the sequel are *with memory*.

**Lemma 4.1.** *The following dichotomy holds for* (S)CA *with* $\delta = MAJ$ *and* $r = 1$*:*

*(i)* All *finite parallel 1-D CA with* $r = 1$*, the* Majority *update rule, and an even number of nodes, have finite temporal cycles in their phases spaces (**PS**); the same holds for the two-way* infinite *1-D* Majority CA.

*(ii) The 1-D* sequential *CA with* $r = 1$ *and the* $\delta = MAJ$ *update rule do not have any temporal cycles in their phase spaces,* irrespective of the sequential node update ordering $s$.

**Remark:** In case of the infinite sequential SCA as in the Lemma above, a *nontrivial* temporal cycle configuration – one that is not a fixed point – does not exist even in the limit. We also note that $s$ can be an arbitrary sequence of an SCA nodes' indices, not necessarily a (possibly infinitely repeated) permutation, or even a function that is necessarily *onto* $L$.

*Proof.* To show (i), we exhibit an actual two-cycle. We consider both an infinite 1-D CA and a finite one, with the circular boundary conditions and an even number of nodes, $2n$. Then the configurations $(10)^\omega$ and $(01)^\omega$ in the infinite case, where $\omega$ stands for the smallest infinite ordinal number, constitute a temporal two-cycle. Likewise, configurations $(10)^n$ and $(01)^n$ in the finite ring case also form a 2-cycle.

To prove (ii), we must show that no cycle is ever possible, irrespective of the starting configuration. We consider all possible 1-neighborhoods (there are eight of them: 000, 001, ..., 111), and show that, locally, none of them can be cyclic yet not fixed. The case analysis is simple: 000 and 111 are stable (fixed) sub-configurations. The sub-configuration 010, after a single node update, can either stay fixed, or else evolve into any of $\{000, 110, 011\}$; since we are only interested in non-FPs, in the latter case, one can readily show by induction that, after any number of steps, the only additional sub-configuration that can be reached is 111, i.e., assuming that 010 does not stay fixed, $010 \rightarrow^\star \{000, 110, 011, 111\}$. However, $010 \notin \{000, 110, 011, 111\}$. By symmetry, similar analysis holds for the sub-configuration 101. On the other hand, 110 and 011 either remain fixed, or else at some time step $t$ evolve to 111, which subsequently stays fixed. A similar analysis applies to 001 and 100. Hence, no local neighborhood $x_1x_2x_3$, once it changes, can ever "come back". Therefore, there are no proper cycles in Sequential 1-D CA with $r = 1$ and $\delta =$ Majority. □

An astute reader may have noticed that the above case analysis in the proof of (ii) can be somewhat simplified if one observes that, for $r = 1$, the sub-configurations 11 and 00 are always stable with respect to the MAJORITY node update function, irrespective of the left or right neighbors of the node performing its update, or the updating sequential order.

We remark that, in the finite ring case, if the number of nodes is odd instead of even, then the resulting CA is temporal cycle-free both when the boundary conditions are made circular and when they are held fixed. In case of the circular boundary conditions, since the number of nodes is odd, there must exist two adjacent[17] nodes that are in the same state. Therefore, that pair of nodes will never "flip"; those two nodes are therefore *stable*. Let's say, the two stable nodes are $x_i$ and $x_{i+1}$. Moreover, if another neighbor of the two stable adjacent nodes ever changes its state into the same state as that of $x_i$ and $x_{i+1}$, then that other node will also become a stable node, i.e., a part of an enlarged block of consecutive nodes in the same state (namely, either $x_{i-1}x_ix_{i+1}$ or $x_ix_{i+1}x_{i+2}$).

A formal proof based on this informal reasoning and an exhaustive case analysis can be readily constructed to establish the impossibility of temporal cycles in the scenario where $n$ is odd and the boundary conditions are circular. A similar argument can be provided for the *fixed* boundary condition case. The formal proofs are fairly elementary and will therefore be omitted.

Part (ii) of Lemma 4.1 above can be readily generalized: even if we consider local update rules $\delta$ other than the MAJORITY rule, yet restrict $\delta$ to *monotone symmetric (Boolean) functions* of the input bits, no such sequential CA can have any proper cycles.

**Theorem 4.1.** *For every* Monotone Symmetric Boolean 1-D Sequential CA *$A$ with $r = 1$, and every sequence $s$ of node updates, the phase space $PS(A)$ of the cellular automaton $A$ is cycle-free.*

*Proof.* Since $r = 1$ and $2r + 1 = 3$, there are only *five Monotone Symmetric Boolean* (MSB) functions (equivalently, simple threshold functions) on three inputs. Two of these MSB functions are utterly trivial (the constant functions 0 and 1). The "at-least-1-out-of-3" simple threshold function is the Boolean *OR* on three inputs; similarly, the "at-least-3-out-of-3" simple threshold function is the Boolean *AND.* It is straightforward to show that the CA (sequential or parallel, as long as they are *with memory*) with $\delta \in \{OR, AND\}$ cannot have temporal cycles. The only remaining MSB update rule on three inputs is $\delta = MAJ$, for which we have already argued that the corresponding parallel CA may have temporal two-cycles, whereas, in contrast, all the corresponding SCA – and therefore NICA – have cycle-free configuration spaces. □

Similar results to those in Lemma 4.1 and Theorem 4.1 also hold for 1-D CA with radius $r = 2$:

**Lemma 4.2.** *The following dichotomy holds for* (S)CA *with $\delta = MAJ$ and $r = 2$:*

*(i) There are 1-D parallel CA with $r = 2$ and $\delta =$ MAJ that have finite cycles in the phase space.*

*(ii) Every 1-D SCA with $r = 2$ and $\delta =$ MAJ, for any sequential order $s$ of the node updates whatsoever, has a cycle-free configuration space.*

[17] By this we mean, *adjacent in the extended sense* where $x_n$ is adjacent to $x_1$.

*Proof.* (i) For $r = 2$, consider configurations $(1100)^\omega$ and $(0011)^\omega$; it is easy to verify that these two configurations form a temporal two-cycle for the parallel CA defined over a two-way infinite line.

The argument in (ii) is similar to that of Lemma 4.1 part (ii), except that now there are $2^5 = 32$ fundamental neighborhoods of the form $x[1]...x[5]$ to consider. We notice that, for $r = 2$, the sub-configurations 000 and 111 are stable; this observation simplifies the case analysis.

□

Generalizing Lemmata 4.1 and 4.2, part (i), we have the following

**Corollary 4.1.** *For all $r \geq 1$, there exists a CA with a* monotone symmetric *Boolean update rule (that is, a* simple threshold cellular automaton*) $A$ such that $A$ has finite temporal cycles in the phase space.*

Namely, given an arbitrary $r \geq 1$, a (classical, concurrent) CA with $\delta = MAJ$ and $\Gamma =$ *infinite line* has at least one two-cycle in the configuration space: $\{(0^r1^r)^\omega, (1^r0^r)^\omega\}$. If $r \geq 3$ is odd, then such a threshold cellular automaton has at least two distinct two-cycles, since $\{(01)^\omega, (10)^\omega\}$ is also a two-cycle. Analogous results hold for the *simple threshold* CA defined on *finite* 1-D cellular spaces, provided that those CA have sufficiently many nodes, that the number of nodes is appropriate (see [200] for more details), and assuming circular boundary conditions (i.e., assuming that $\Gamma$ is a sufficiently big finite ring). Moreover, the result extends to many finite and infinite CA in the higher dimensions, as well; in particular, the simple threshold CA with $\delta = MAJ$ that are defined over *two-dimensional Cartesian grids* and *Hypercubes* have two-cycles in their respective phase spaces.

More generally, for any underlying cellular space $\Gamma$ that is a (finite or infinite) *bipartite graph*, the corresponding (nontrivial) parallel CA with $\delta = MAJ$ have temporal two-cycles. We remark that bipartiteness of $\Gamma$ is sufficient, but it is not necessary, for the existence of temporal two-cycles in this setting.

It turns out that the two-cycles in the configuration spaces of concurrent CA with $\delta = MAJ$ are actually the only type of (proper) temporal cycles such cellular automata can have. Indeed, for any *symmetric linear threshold update rule* $\delta$, and any *finite* regular Cayley graph as the underlying cellular space, the following general result holds [63, 68]:

**Proposition 4.1.** [68] *Let a classical, parallel simple threshold CA $A = (\Gamma, N, M)$ be given, where $\Gamma$ is an arbitrary* finite *cellular space, and let this cellular automaton's* global map *be denoted by $F$. Then for all configurations $\mathcal{C} \in PS(A)$, there exists a finite time step $t \geq 0$ such that $F^{t+2}(\mathcal{C}) = F^t(\mathcal{C})$.*

In particular, this result implies that, for every *finite* simple threshold cellular automaton, and for every starting configuration $\mathcal{C}_0$, there are *only two possible kinds of orbits:* upon repeated iteration, the computation either converges to a fixed point configuration after finitely many steps, or else it eventually arrives at a two-cycle.

It is almost immediate that, if we allow the underlying cellular space $\Gamma$ to be infinite, if computation from a given starting configuration converges after any finite number of steps at all, it will have to converge either to a fixed point or a two-cycle (but never to a cycle of, say, period

three – or any other finite period). The result also extends to finite and infinite *SCA*, provided that we reasonably define what is meant by a single computational step in a situation where the nodes update one at a time. The simplest notion of a single computational step of an SCA is that of a single node updating its state. Thus, a single parallel step of a classical CA defined on an infinite underlying cellular space $\Gamma$ includes an infinite amount of sequential computation and, in particular, infinitely many elementary sequential steps. Discussing the implications of this observation, however, is beyond the scope of this work.

Additionally, in order to ensure some sort of convergence of an arbitrary SCA (especially when the underlying $\Gamma$ is infinite), and, more generally, in order to ensure that *all the nodes* get a chance to update their states, an appropriate condition that guarantees *fairness* needs to be specified. That is, an appropriate restriction on the allowable sequences $s$ of node updates is required. As a first step towards that end, we shall allow only *infinite* sequences $s$ of node updates through the rest of this Section.

For the SCA defined on finite cellular spaces, one sufficient fairness condition is to impose a fixed upper bound on the number of sequential steps before any given node gets its turn to update (again). This is the simplest generalization of the fixed permutation assumption made in the work on sequential and synchronous dynamical systems; see, e.g., [16, 17, 19, 20]. In the infinite SCA case, on the other hand, the issue of fairness is nontrivial, and some form of *dove-tailing* of sequential individual node updates may need to be imposed. In the sequel, we shall require from the sequences $s$ of node updates of the SCA and NICA threshold automata to be fair in a simple sense to be defined shortly, without imposing any further restrictions or investigating how are such fair sequences of node updates to be generated in a physically realistic distributed setting. For our purposes herein, therefore, the following simple notion of fairness will suffice:

**Definition 4.11.** *An infinite sequence $s : N \to L$ is* fair *if (i) the domain $L$ is finite or countably infinite, and (ii) every element $x \in L$ appears infinitely often in the sequence of values $s(1) = s_1, s(2) = s_2, s(3) = s_3, ...$*

Let $s : N \to L$ be an arbitrary infinite sequence of elements from some domain $L$. Let $s^{[q]}$ denote the $q$-tail of $s$, i.e., $s^{[q]} = s_{q+1}, s_{q+2}, s_{q+3}, ...$

For the future reference, we state the following alternative characterizations of fair sequences:

**Observation 4.1.** *Let an infinite sequence $s : N \to L$ be given, where the set $L$ is countable. Then the following four properties are all equivalent to one another:*

*(i)* $s$ *is* fair;
*(ii)* $\forall n \in N$, $s^{[n]}$ *is* fair;
*(iii)* $(\forall x \in L)(\forall n \in N)(\exists n' \in N)(n' > n \ \wedge \ s(n') = x)$
*(iv)* $\forall n \in N$, $s^{[n]} : \{n+1, n+2, ...\} \to L$ *is* onto.

The proof that the four statements in Observation 4.1 are indeed equivalent is elementary, and is therefore omitted.

Now that we have defined what we mean by a *single step* of a sequential CA, as well as adopted some reasonable notion[18] of *fairness*, we can establish the generalizations of Proposition

[18]Our notion of fairness in Definition 4.11 need not be the most general, or most suitable in all situations, such a notion. However, it is appropriate for our purposes and, in particular, sufficient for the results on threshold SCA and NICA that are to follow; see Proposition 4.3 in the main text.

4.1 for both finite and infinite 1-D (parallel) CA and 1-D SCA. We start with the (S)CA defined on *finite* cellular spaces:

**Proposition 4.2.** *Let a parallel CA or a sequential SCA $A$ with a simple threshold update rule of radius $r \geq 1$ be defined over a finite* one-dimensional *cellular space. Let's also assume, in the sequential cases, that the fairness condition from Definition 4.11 holds.*

*Then for every starting configuration $\mathcal{C}_0 \in PS(A)$ there exists a time step $t \geq 0$ such that*

$$F^{t+2}(\mathcal{C}) = F^t(\mathcal{C}) \tag{10}$$

*where, in the case of* fair SCA*, the* Eqn. (10) *above can be replaced with*

$$F^{t+1}(\mathcal{C}) = F^t(\mathcal{C}) \tag{11}$$

A similar claim holds for the CA and *fair* SCA defined on *infinite* cellular spaces, provided that we confine our attention to *compactly supported* subconfigurations.

**Definition 4.12.** *A global configuration of a cellular automaton defined over an infinite cellular space $\Gamma$ is said to be* finitely supported *or* compactly supported [68] *if all except for at most finitely many of the nodes are* quiescent *(i.e., in the state 0) in that configuration.*

**Proposition 4.3.** *Let a parallel CA or a* fair *sequential SCA $A$ with a simple threshold update rule of radius $r \geq 1$ be defined over an infinite one-dimensional cellular space.*

*Then for every starting configuration $\mathcal{C}_0 \in PS(A)$ and every* finite subconfiguration[19] $\mathcal{C} \subseteq \mathcal{C}_0$*, there exists a time step $t \geq 0$ such that*

$$F^{t+2}(\mathcal{C}) = F^t(\mathcal{C}) \tag{12}$$

*where again, in the case of* fair SCA*, the* Equation (12) *can be replaced with*

$$F^{t+1}(\mathcal{C}) = F^t(\mathcal{C}) \tag{13}$$

In the special case of $\delta = MAJ$ (S)CA, a computation starting from any *finitely supported* initial configuration necessarily converges to either a FP or a two-cycle [68]:

**Proposition 4.4.** *Let the assumptions from* Proposition *4.3 hold, and let the underlying threshold rule be $\delta = MAJ$. Then for* all *configurations $\mathcal{C} \in PS(A)$ in the finite cases, and for all configurations $\mathcal{C} \in PS(A)$ that are finitely supported when $\Gamma(A)$ is infinite, there exists a finite time step $t \geq 0$ such that $F^{t+2}(\mathcal{C}) = F^t(\mathcal{C})$. Moreover, in the sequential cases with* fair *node update sequences, there exists a finite $t \geq 0$ such that $F^{t+1}(\mathcal{C}) = F^t(\mathcal{C})$.*

Furthermore, if *arbitrary* infinite initial configurations are allowed in Propositions *4.3–4.4*, and the dynamic evolution of the full such global states is monitored, then the only additional possibility is that the particular (S)CA computation fails to finitely converge altogether. In

[19] In this case, the subconfiguration need not necessarily be made of *consecutive* nodes.

that case, and under the fairness assumption in the case of SCA, the limiting configuration $lim_{t\to\infty}F^t(\mathcal{C}) = \mathcal{C}^{lim}$ can be shown to be a (stable) fixed point.

To summarize, if the computation of a SCA starting from some configuration $\mathcal{C}^0$ converges at all (that is, to *any* finite temporal cycle), it actually has to converge to a fixed point.

To convince oneself of the validity of Proposition 4.3, two basic facts have to be established. One, convergence to finite temporal cycles of length three or higher is not possible. Indeed, Propositions 4.1 and 4.2 establish that the only possible long-term behaviors of the finite threshold (S)CA are (i) the convergence to a fixed point and (ii) the convergence to a two-cycle. If infinite cellular spaces are considered, it is straightforward to see that the only new possibility is that the long-term dynamics of a (S)CA fails to (finitely) converge altogether. In some cases with infinite $\Gamma$ such divergence indeed takes place – even when the starting configuration is finitely (compactly) supported: consider, e.g., the *OR* automaton and the starting configuration ...00100... on the two-way infinite line. Two, in the sequential cases (that is, for the simple threshold SCA and NICA), temporal two-cycles are not possible. That is, a generalization of Lemmata 4.1, 4.2 and Theorem 4.1 to arbitrary finite $r \geq 1$, and arbitrary symmetric threshold update rules, holds. This generalization is provided by an appropriate specialization of a similar result in [16] for a class of sequential network automata called *Sequential Dynamical Systems* (SDSs), with possibly different simple $k$-threshold update rules at different nodes, and a node update ordering given by repeating *ad infinitum* a (fixed) permutation of the nodes. In particular, part (ii) in the Theorem 4.2 below and its proof are based on a similar result in [16]:

**Theorem 4.2.** *The following dichotomy holds:*

*(i) All* 1-D (parallel) CA *with an arbitrary odd $r \geq 1$, the local rule $\delta = MAJ$, and cellular space $\Gamma$ that is either a finite ring with an even number of nodes or a two-way infinite line, have finite cycles in their phase spaces. The same holds for* arbitrary *(even or odd) $r \geq 1$ provided that $\Gamma$ is either a finite ring with a number of nodes divisible by $2r$, or a two-way infinite line*[20]*.*

*(ii) An arbitrary* 1-D SCA *with a simple threshold Boolean update rule $\delta$, an arbitrary but fixed finite $r \geq 1$, defined over a finite or infinite 1-D cellular space, and for* an arbitrary *sequence $s$ (finite or infinite, fair or unfair) as the node update ordering, has a cycle-free phase space.*

*Proof.* Part (i): For the special case when $r = 2$, consider the configurations $(1100)^\omega$ and $(0011)^\omega$; it is easy to verify that these two configurations form a cycle for the corresponding parallel CA. Similar reasoning readily generalizes to arbitrary $r \geq 2$. The "canonical" temporal two-cycle for 1-D MAJORITY CA defined over an infinite line with $r \geq 1$ is $\{(1^r0^r)\}^\omega$, $(0^r1^r)^\omega$, with the obvious modification for the finite CA with $n$ nodes, where $n$ is even and sufficiently large (with respect to $r$), and assuming as before the circular boundary conditions.

Part (ii) (proof sketch): The proof of this interesting property is based on a slight modification of a similar result in [16] for a class of the sequential network automata called *Sequential Dynamical Systems* (SDSs) that we shall study in detail in Sections 5 and 6.

[20]There are also CA defined over finite rings and with even $r \geq 2$ such that the number of nodes in these rings is not divisible by $2r$ yet temporal two-cycles exist. However, a more detailed discussion on what properties the number of nodes in such CA has to satisfy is required; we leave this discussion out, for the sake of clarity and space constraints.

The central idea of the proof of part (ii) of the theorem is to assign nonnegative integer *potentials* to both nodes and edges in the *functional graph* of the given SCA. In this functional graph, for any two nodes $x_i$ and $x_j$, unordered pair $\{x_i, x_j\}$ is an edge if and only if those two nodes provide inputs to one another, i.e., in the 1-D SCA case, if and only if *distance*$(x_i, x_j)$ $\leq r$ (that is, assuming the canonical labeling of the nodes, so that consecutive nodes always get labeled by consecutive integers, iff $|i - j| \leq r$). The potentials are assigned in such a way that, each time a node changes its value (from 0 to 1 or *vice versa*), the overall potential of the resulting configuration is strictly less than the overall potential of the configuration before the node flip. Since all individual node and edge potentials are initially nonnegative, and since the total potential of any configuration (that is, the sum of all individual node and edge potentials in that configuration) is always nonnegative, the fact that *each flip* of any node's value strictly decreases the overall potential by an integer amount implies that, after a finite number of node flips (and, therefore, a finite number of sequential steps), an equilibrium where no nodes can further flip is reached; that equilibrium will be a fixed point configuration.

For a full formal proof of part (ii), see *Appendix* to this Section. □

To summarize, simple linear threshold CA, depending on the starting configuration, may converge to a fixed point or a temporal two-cycle; in particular, they may end up *looping* in finite yet nontrivial temporal cycles. In contrast, the corresponding classes of SCA (and therefore NICA) *can never cycle*. We also observe that, given an arbitrary sequence of node updates of a finite threshold SCA, if that sequence satisfies an appropriate *fairness condition*, then it can be shown that the computation of such a threshold SCA $A$ is guaranteed to converge to a stable fixed-point (sub)configuration on any finite subset of nodes in $\Gamma(A)$.

The temporal cycle-freeness of the threshold SCA and NICA holds irrespective of the choice of a sequential update ordering (and, extending to infinite SCA, temporal cycles cannot be obtained even *in the limit*[21]). Hence, we conclude that no choice of a *sequential interleaving* can adequately capture the perfectly synchronous parallel computation of the parallel threshold CA. Consequently, the *interleaving semantics* of NICA fails to capture the synchronous parallel behavior of the classical parallel CA even for this, simplest nonlinear and nonaffine class of totalistic update rules.

## 4.4 Configuration Spaces of (S)CA with $\delta$ = MAJORITY

Next, we specifically focus on $\delta$ = MAJORITY 1-D CA, and completely characterize the configuration spaces of such threshold cellular automata. In particular, in the $\Gamma$ = *infinite line* case, we will show that the cycle configurations are rather rare, that the fixed point configurations are quite numerous (there are uncountably many of them) yet still relatively rare in a precise mathematical sense to be discussed below, and that *almost all* configurations of these threshold cellular automata are transient. We remark that we use the term *'almost all'* in a mathematically precise, measure-theoretic sense.

[21] That is, via infinitely long computations, obtained by allowing arbitrary infinite sequences of individual node updates.

Throughout, unless explicitly stated otherwise, *one-dimensional* (S)CA will be assumed (either finite or infinite). In finite cases, *circular boundary conditions* will be assumed by default. That is, the cellular spaces in this subsection are either infinite lines or finite rings.

Also, in all the results to follow in the rest of this subsection, whenever we refer to a *subconfiguration*, unless explicitly stated otherwise, we will mean a subconfiguration that is made of some number (two or more) of *consecutive nodes*.

Insofar as *infinite* SCA are concerned, the *fairness condition* from Definition 4.11 is assumed. For the finite SCA, an appropriate fairness condition is assumed that ensures that each node gets its turn to update within a *fixed* finite number of discrete time steps. Also, when we refer to FPs in case of SCA (or NICA), we mean quasi-fixed points such that there exists an infinite sequence of individual node updates that satisfies the fairness condition and such that, with respect to that sequence, the particular configuration is a "proper" FP (but this configuration may be, for example, a CC or a TC with respect to other sequences of node updates).

We begin with some elementary observations about the nature of various configurations in the (S)CA with $\delta = MAJ$ and $r = 1$. We shall subsequently generalize several of those results to arbitrary $r \geq 1$.

We first recall that, for such (S)CA with $r = 1$, two adjacent nodes of the same value are stable. That is, 11 and 00 are stable subconfigurations. Consider now the starting subconfiguration $x_{i-1}x_ix_{i+1} = 101$. In the parallel case, at the next time step, $x_i \to 1$. Hence, no FP configuration of a parallel CA can contain 101 as a subconfiguration. In the sequential case, assuming fairness, $x_i$ will eventually have to update. If, at that time, it is still the case that $x_{i-1} = x_{i+1} = 1$, then $x_i \to 1$, and $x_{i-1}x_ix_{i+1} \to 111$, which is stable. Else, at least one of $x_{i-1}, x_{i+1}$ has already "flipped" into 0. Without loss of generality, let's assume $x_{i-1} = 0$. Then $x_{i-1}x_i = 00$, which is stable; so, in particular, $x_{i-1}x_ix_{i+1}$ will never go back to the original 101. By symmetry of $\delta = \text{MAJORITY}$ with respect to 0 and 1, the same analysis applies to the subconfiguration $x_{i-1}x_ix_{i+1} = 010$. In particular, the following properties hold:

**Lemma 4.3.** *A fixed point configuration of a* 1D-(S)CA *with* $\delta = \text{MAJ}$ *and* $r = 1$ *cannot contain subconfigurations* 101 *or* 010. *Similarly, a cycle configuration of such a* 1D-(S)CA *cannot contain subconfigurations* 00 *or* 11.

*Proof.* In any configuration that contains 101 as a subconfiguration, at the very next parallel update the 0 in between the two 1s will flip to 1, regardless of how many other nodes are present, and what are their current states. Analogous argument applies to configurations that contain 010. Hence, FPs of CA with $\delta = MAJ$ and $r = 1$ are solely made of consecutive blocks of two or more 1s and/or similar blocks of two or more 0s.

As for the claim about the cycle configurations, notice that 00 and 11 are stable (sub)configurations. Without loss of generality, assume a CC contains a block of two or more consecutive 0s. Consider, say, the node adjacent to the rightmost 0 in the block. This node is, by the assumption, in the state 1. We denote that node by $x_i$. There are two cases to consider. If its right neighbor $x_{i+1}$ is in the state 0, then, at the very next step, $x_i$ will flip to 0, and, since it will be adjacent to $x_{i-1} = 0$ to the left, it will remain at 0 thereafter. Hence, the starting configuration cannot be a CC. The other possibility is that the left neighbor of the first 1 to the right from the block of

zeros is also in state 1; that is, the configuration is of the form 00...0011.... Then the block of (at least) two 1s is stable, and so is the block of two or more 0s to the left from it.

That such a configuration cannot be a cycle state now follows by induction: proceeding moving along the line (or ring) of nodes from the assumed block of zeros to the right, either eventually a block of the form ...010 or ...101 is encountered, in which case this configuration is transient, or else no such a subconfiguration exists, in which case the entire configuration is made solely of the stable blocks of two or more 0s and similar stable blocks of 1s, and thus is a fixed point. Either way, the assumption that this configuration was actually a cycle configuration is violated.

□

Insofar as the *parallel* CA with $\delta = MAJ$ are concerned, by virtue of their determinism, a complete characterization of each of the three basic types of configurations (FPs, CCs, TCs) is now almost immediate:

**Lemma 4.4.** *The FPs of the* 1D-(S)CA *with* $\delta = \mathrm{MAJ}$ *and* $r = 1$ *are precisely of the form*[22] $(000^* + 111^*)^*$. *The CCs of such* 1D-CA *exist only in the parallel case, and the temporal cycles are precisely of the form* $\{(10)^*, (01)^*\}$. *The TCs of CA are all the rest, that is, precisely the configurations that contain both (i)* $000^*$ *or* $111^*$ *(or both), and (ii)* 101 *or* 010 *(or both) as their subconfigurations. In addition, the CCs in the parallel case become TCs in all corresponding sequential cases.*

*Proof.* The claim of the Lemma follows from the result of Lemma 4.3 and an elementary case analysis. □

We observe that, for $r \geq 2$, there exist cycle configurations that actually contain *stable subconfigurations*. Similarly, there exist FPs that are characterized by *spatial periodicity*, and are not made of the consecutive stable blocks of 0s and/or 1s. Such FPs will be discussed again in Sections 5 and 6, in the context of *enumeration* of fixed points in certain classes of SDSs and threshold CA. Likewise, giving a similar characterization for the higher dimensional cellular spaces is also not as straightforward as the results in the Lemmata above.

Therefore, the partition of the configurations into FPs, CCs and TCs obtained in this subsection is attributable to the peculiarity of the 1-D cellular spaces as well as the assumption that the rule radius $r = 1$.

However, some generalizations to arbitrary (finite) rule radii $r$ can be readily deducted. For instance, given any such $r \geq 1$, the finite subconfigurations $0^{r+1}$ and $1^{r+1}$ are stable with respect to $\delta = MAJ$ update rule applied either in parallel or sequentially; consequently, every configuration of the form $(0^{r+1}0^* + 1^{r+1}1^*)^*$, for a finite or infinite CA with an appropriate number of nodes, is a fixed point. This characterization, only with a considerably different notation, has been known for the case of configurations with *compact support* for a relatively long time; see, e.g., *Section 4* in [68]. On the other hand, fully characterizing CCs (and, consequently, also TCs) in case of finite or infinite parallel CA is more complicated than in the simplest case with $r = 1$. For example, for $r \geq 1$ odd, $\{(10)^*, (01)^*\}$ is a two-cycle, whereas for $r \geq 2$ even, each of $(10)^*$,

[22] We use the standard notation for *regular expressions*; in particular, $'*'$ stands for the *Kleene star* (e.g., [169]).

$(01)^*$ is a fixed point. However, for all $r \geq 1$, the corresponding infinite (parallel) CA are guaranteed to have some temporal cycles, namely, given $r \geq 1$, the set of states $\{(1^r0^r)^\omega, (0^r1^r)^\omega\}$ forms a two-cycle.

**Lemma 4.5.** *Given a (finite or infinite) simple threshold (S)CA with the rule radius $r = 1$, one of the following two properties must hold:*

*(i) this simple threshold cellular automaton does not have proper temporal cycles and cycle configurations at all; or else*

*(ii) if there are cycle configurations in the configuration space of this (S)CA, then none of those cycle configurations has any incoming transients.*

*Proof.* Let's assume a threshold CA with $r = 1$ has a cycle configuration. Then the update rule $\delta$ of this CA cannot be either Boolean $AND$ or Boolean $OR$ and, consequently, since $r = 1$, it follows that it must be the case that $\delta = MAJ$.

Now, if a cycle configuration of this CA actually had an incoming transient, then there would exist a predecessor configuration of this cycle configuration such that this predecessor configuration is transient. Let $C'$ denote that transient configuration, and let $C$ denote the cycle configuration in question (so that $F(C') = C$ where $F$ stands for this cellular automaton's global map). By Lemmata 4.3 and 4.4, configuration $C'$ must contain both stable and unstable subconfigurations. In particular, $C'$ contains either a stable block of the form 00... or of the form 11...; however, this implies that $F(C')$ also contains such a stable block and, consequently, by Lemma 4.3, $F(C')$ cannot be a cycle configuration. Therefore, it follows that any predecessor of a cycle configuration cannot be a TC; hence, such a predecessor itself also has to be a cycle configuration, and the claim of the Lemma follows. □

We strongly suspect that the property in Lemma 4.5 actually holds for *arbitrary* rule radii $r \geq 1$, but do not have a rigorous proof (or know of a counterexample) as of yet:

**Conjecture 4.1.** *Given a (finite or infinite) simple threshold parallel or sequential CA with an arbitrary rule radius $r \geq 1$, if there are cycle configurations in the configuration space of that cellular automaton, then none of those temporal cycles has any incoming transients.*

Next, we will show that the fixed points of simple threshold automata can be quite numerous when $\delta = MAJ$. Infinite sequential and parallel MAJORITY CA alike have infinitely many FPs, and this property holds for every rule radius $r \geq 1$. Moreover, the cardinality of the set of FPs, in case of $\delta =$ MAJ and the (countably) infinite cellular spaces, equals the cardinality of the entire configuration space:

**Theorem 4.3.** *An infinite* 1-D(S)CA *with $\delta =$ MAJORITY and any $r \geq 1$ has* uncountably many *fixed points.*

*Proof.* For the notational convenience, let us consider *one-way infinite* (S)CA. Similar proof can be constructed for the usual, two-way infinite (S)CA.

Let us consider the FPs of the form $1^{r+k_1}0^{r+k_2}1^{r+k_3}...$ with all $k_i$ being integers such that $k_i \geq 1$ $(i = 1, 2, 3, ...)$. We shall identify a string of $r+k_i$ *consecutive* 1s or 0s as above with decimal

digit $k_i - 1$ $(mod\ 10)$. We now construct a mapping from a subset of the set of all FPs of such an automaton to the real numbers in the unit interval $[0, 1] \subset \mathbf{R}$. Let the length of the $m$-th block of consecutive 0s or 1s be denoted by $L_m$. Then $L_m \geq r + 1$ is going to be mapped into $L_m - 2$ $(mod\ 10)$. To give some examples, if $r = 1$, then $11100111110000111111... = 1^3 0^2 1^5 0^4 11...$ gets mapped to $0.1032...$, and $1^{15} 0^{28} 1^7...$ gets mapped to $0.365...$, etc.

It is immediate that this mapping is constructed so that it is *onto* the real line unit interval $[0, 1]$, which has *uncountably many* points (i.e., real numbers), that is, this interval is of cardinality $2^{\aleph_0}$. Since the set of infinite 1D (S)CA configurations that includes all configurations made of stable blocks only (and no other configurations) is, in general (for arbitrary $r \geq 1$) a subset of the set of all fixed points of such an infinite (S)CA, it follows that every such (S)CA with $\delta = MAJ$ has at least as many FPs as there are real numbers in the unit interval on the real line.

Therefore, an infinite 1D-(S)CA with $\delta =$ MAJORITY and any $r \geq 1$ has *uncountably many* fixed points. □

The above result is another evidence that "not all threshold (S)CA are born equal". It suffices to consider only 1D, infinite CA to see a rather dramatic difference. Namely, in contrast to the $\delta =$ MAJORITY CA, the CA with memory and with $\delta \in \{OR, AND\}$ (i) do not have any temporal cycles, and (ii) have *exactly two* FPs, namely, $0^\omega$ and $1^\omega$. Other threshold CA may have temporal cycles, as discussed in the previous subsubsection, but they still have only a finite number of FPs.

### 4.4.1 Some Statistical Properties

We continue the analysis of the configuration spaces of the one-dimensional infinite simple threshold (S)CA. We shall now focus on their *statistical properties*. The two results in this subsubsection hold for *all* simple threshold update rules. Presently, we will sketch proofs for $\delta = MAJ$ only; that the results hold for arbitrary $k$-threshold rules will follow from the discussion in subsection 6.4 which focuses the structural properties of the fixed point configurations of simple threshold one-dimensional (S)CA, in the context of *counting* those FPs in various simple threshold (S)CA.

We have just shown in the previous subsubsection that there are *uncountably* many FPs of the MAJORITY sequential and parallel cellular automata. However, the FPs are, when compared to the transient states, still but *a few and far between*. To convince oneself of that fact, the basics of probability and measure theory are needed. In particular, let's assume that a *random* global configuration is obtained by "picking" each bit (or site's value) to be either 0 or 1 at random, with equal probability, and so that assigning bit-value to one site is independent of the bit assignment to any of the other sites. Then the following result holds:

**Lemma 4.6.** *If a global configuration of an infinite simple threshold cellular automaton is selected* at random*, that is, by assigning each node's value independently and according to a toss of a fair coin, then, with probability 1, this randomly picked configuration will be a transient state.*

When $\delta = MAJ$ and given the rule radius $r \geq 1$, the claim of the Lemma follows from the observation that a random infinite configuration $\mathcal{C}$ will contain with probability $p_{unstable} = 1$ an *unstable block* of consecutive nodes, such as $r$ or fewer consecutive nodes all in the state 0. Therefore, with probability 1, such random configuration cannot be a fixed point. Likewise,

the probability of random $\mathcal{C}$ being a cycle configuration is also zero, and therefore this random configuration must be transient with probability 1.

Moreover, the *unbiased randomness*, while sufficient, is certainly not necessary. In particular, assigning bit values according to outcomes of tossing a coin with a fixed bias also yields transient states being of probability $Pr(TC) = 1$.

**Proposition 4.5.** *Let $p$ be any real number such that $0 < p < 1$, and let the probability of a site in a global configuration of an infinite 1D simple threshold cellular automaton being in state 1 be equal to $p$ (so that the probability of this site's state being 0 is equal to $q = 1 - p$).*

*If a global configuration of this threshold cellular automaton is selected at random according to probability $p$, and so that each site's state is chosen independently of the other sites, then, with probability 1, this configuration will be a transient state.*

*Proof.* We formally establish the claim of the Proposition for the most interesting special case, namely, when $\delta = MAJ$. That the claim holds for all simple threshold rules will follow from the results and discussion in subsection 6.4.

Since the cellular space is assumed infinite, and since the probability $p$ of a randomly selected node being in state 1 is fixed (and hence bounded away from both 0 and 1), the following properties hold for a randomly selected configuration, where this configuration is a random (according to $p$) infinite sequence of 0s and 1s:

– any finite subsequence of 0s and 1s appears *somewhere* in this infinite sequence with probability 1;

– in particular, stable blocks $1^{r+k}$ and $0^{r+l}$ (for some $k, l \geq 1$) appear with probability 1 somewhere in the infinite configuration;

– the same holds of any unstable finite subconfiguration.

Examples of unstable subconfigurations of *consecutive* nodes include, e.g., 1010101 or 01010101 when the radius $r$ is odd. Again, a canonical example of an unstable subconfiguration, for an arbitrary $r \geq 1$, is a block of at most $r$ 0s (alternatively, 1s) *squeezed* in between some 1s (alternatively, 0s) – for instance, the subconfiguration $\mathcal{C}[r+1] = x_i x_{i+1} x_{i+2} ... x_{i+r-1} x_{i+r} = 100...01$ with exactly $r - 1$ consecutive 0s, where $r \geq 2$, is an example of such an unstable subconfiguration.

Consequently, since such a randomly selected (infinite) configuration contains unstable subconfigurations with probability 1, it follows that, with probability 1, it cannot be a fixed point.

That it likewise cannot be a cycle configuration follows from the argument we outline below. Namely, for any fixed integer $m \geq 1$, each of the binary strings from $\{0+1\}^m$ of length $m$ appears infinitely often[23] in this infinite configuration. In particular, with probability 1, somewhere in the configuration, a finite unstable subconfiguration $C_u$ is squeezed in between two finite stable subconfigurations, $C_{s1}$ and $C_{s2}$.

Therefore, regardless of whether the nodes update synchronously in parallel, or sequentially according to an arbitrary fair sequence $s$, the nodes in $C_u$ will eventually get their turn to update, and at least one of them is going to flip so that it joins the (expanded) stable block, either $C_{s1}$ or $C_{s2}$. It is now immediate, by the argument along the lines of proofs of Lemmata 4.3 and 4.4, that the overall CA configuration that contains the concatenation $C_{s1} \cdot C_u \cdot C_{s2}$ as its finite subconfiguration, cannot be a cycle configuration. □

[23] But certainly not equally often, which is immaterial for our argument here.

We remark that, since each node's state is an *independent, identically distributed* (i.i.d) random variable, this selection of a *random configuration* of an infinite (S)CA cannot be accomplished via finite means (unless one allows for a physically unjustifiable infinite parallelism where infinitely many nodes are assigned a value according to $p$ at once, i.e., instantly in parallel).

Insofar as the simple threshold (S)CA defined on finite cellular spaces are concerned, the following property holds: as the number of nodes, $n$, grows, the fraction of all $2^n$ global configurations that are *transient* will grow, as well. In particular, under the same assumptions as in Lemma 4.6 and Proposition 4.5 above, in the limit, as $n \to \infty$, the probability that a randomly chosen configuration, $\mathcal{C}$, is a transient state approaches 1:

$$lim_{n\to\infty}\ Pr(\mathcal{C}\ is\ transient)\ =\ 1 \tag{14}$$

Thus, a fairly complete characterization of dynamics of simple threshold (S)CA over infinite cellular spaces can be given. In terms of theory of probability and measure theory on infinite spaces, it can be said that, in the infinite threshold (S)CA case, *almost every* configuration is a transient configuration. However, the striking contrast between $\delta =$ MAJORITY on the one, and all other simple threshold rules, on the other hand, remains: the former has uncountably many FPs (besides uncountably many TCs, of course), whereas all other simple threshold update rules only yield finitely many FPs [205].

### 4.4.2 On the Rates of Convergence

In order to make the study of simple threshold CA and SCA defined on one-dimensional finite or infinite cellular spaces that has been undertaken in this Section complete, the issue of convergence, that is, the REACHABILITY of FP configurations, needs to be addressed. There are two closely related but distinct questions that remain to be answered. One, starting from an arbitrary configuration, can one predict which fixed point (or, perhaps, which temporal two-cycle) will be eventually reached? And two, how long can this convergence to the appropriate limit take, be that limit an FP or a 2-cycle? The first question has been addressed in the literature in several different contexts, including the classical finite cellular automata [179, 181], as well as the sequential dynamical systems [16] whose properties will be studied in detail in much of the rest of this technical report, starting with Section 5. We address the second of the two questions above in the context of simple threshold CA next.

Before we proceed to the results, we introduce some additional notational conventions, as well as remind the reader of some of the already established conventions. We will use superscripts to indicate the time step (equivalently, the iteration) of a parallel CA's evolution. Thus, given a configuration $\mathcal{C}$, we use $\mathcal{C}^t$ to denote the $t$-th iteration of a parallel CA $A$ starting from $\mathcal{C}$ as the initial configuration; that is, we denote by $\mathcal{C}^t$ what we have been also denoting as $F^t(\mathcal{C})$ whenever we wanted to emphasize the (iterated) global map view of a CA's evolution in time. However, we need to avoid the confusion between the time step pertaining to the evolution of a configuration (or even of a single node's state), and the number of nodes in a (sub)configuration. To that end, '[...]' associated with a CA configuration will be used to denote the number of nodes in a (sub)configuration; for instance, $\mathcal{C}[m]$ will denote a subconfiguration of $\mathcal{C}$ that is made of a block of $m$ consecutive nodes: $\mathcal{C}[m] = x[i+1]x[i+2]...x[i+m] \subset \mathcal{C}$, where it is tacitly understood that the CA in question has at least $m$ nodes. We use the notation $x[i]$ to denote the $i$-th node of a

cellular automaton, as well as the state of the $i$-th node in a particular configuration. Throughout the rest of this technical report, both the node itself and its state are normally denoted as $x_i$. (That the same notation is used for a node of a cellular or graph automaton, and for its state, with the intended meaning being provided by a given context, is a practice we use throughout this work.) In summary, if used to index (a part of) a configuration, '[q]' stands for the number of consecutive nodes of a subconfiguration, whereas the same notation, when used to index a *single* node of a CA (or, alternatively, a node's state), refers to the $q$-th node according to some ordering of the nodes. As we continue working with *one-dimensional* cellular automata, a linear ordering and appropriate indexing of its nodes will always be assumed, whether that ordering is explicitly stated or not.

We now address the speed of convergence of an arbitrary computation of a parallel 1-D MAJORITY CA: how long can the *transient chain* of configurations be, before the dynamics of such a cellular automaton converges to a fixed point (or, when applicable, a temporal two-cycle)? By *the length of a transient chain* we simply mean the number of (parallel) iterations, or, equivalently, the number of intermediate TCs, that a CA takes starting from a given transient configuration, to reach a fixed point or a two-cycle.

In general, the speed of convergence depends on the node update rule radius, $r$: the greater the radius, the faster a typical MAJORITY CA computation "settles" to one of its (possibly exponentially many – see Section 6) FPs.

**Lemma 4.7.** *Given an arbitrary configuration $\mathcal{C}^0$, a* 1-D MAJORITY *Boolean cellular automaton on $N$ nodes with an arbitrary rule radius $r \geq 1$ and with circular boundary conditions, if started from $\mathcal{C}^0$ as the initial state, will converge (typically, to a fixed point) in at most $\frac{N}{2}$ parallel steps.*

*Proof.* If $\mathcal{C}^0$ is actually a cycle configuration for the *MAJ* rule and a particular value of the rule radius $r \geq 1$, then, by Proposition 4.2, $\mathcal{C}^0$ belongs to a 2-cycle; this property can be verified in only two parallel transition steps. Otherwise, $\mathcal{C}^0$ is either already a FP, or else a transient state. The former can be verified in a single parallel step. So, let's assume $\mathcal{C}^0$ is transient. Since the underlying cellular space is one-dimensional, $\mathcal{C}^0$ can be viewed as a concatenation of alternating stable and unstable "blocks" of consecutive nodes (see subsection 4.3.3). Let $m \leq N-2$ be the maximal length of any unstable block of consecutive nodes in $\mathcal{C}^0$, and let's call this unstable subconfiguration $C^0[m]$, where $C^0[m] \subset \mathcal{C}^0$. Since this unstable block is in between two stable blocks[24], as the computation progresses, the size of the unstable block $C^0[m]$ will decrease by at least two after each parallel iteration. Namely, it can be readily seen that, at the very least, the leftmost and the rightmost nodes of $C^0[m]$ (that are bordering stable blocks to the left and to the right, respectively), will at the next iteration (parallel node update) "flip" to the value of their respective stable one-sided neighborhoods. Since now they will have the same value as their (at least) $r$ neighbors – namely, $r$ out of at least $r+1$ nodes belonging to the stable one-sided neighborhood – they themselves will each join the stable block adjacent to them. Thus, $\mathcal{C}^1 = F(\mathcal{C}^0)$ will have at most $m-2$ nodes in any unstable subconfiguration, and it is therefore

[24]We assume herein circular boundary conditions, so that, even if there is only a single stable block in $\mathcal{C}^0$, due to "wrap-around effect', it actually can be viewed as two, i.e., it neighbors the remaining unstable part of the configuration both from the left and from the right.

immediate (by induction) that a stable full configuration, that is, a fixed point, will be reached in no more than $\frac{m}{2}+1$ parallel steps. Since $m \leq N-2$, the claim of the Lemma follows. $\square$

**Example:** Let us consider the case where there are $n = 17$ nodes in total, and $r = 2$, so that subconfigurations made of three or more consecutive 0s or three or more consecutive 1s are stable. Let $\mathcal{C}^{t=0} = 11110100110101000$. The four leftmost 1s and the three rightmost 0s are stable. The subconfiguration in between, $\mathcal{C}_u^{t=0} = 0100110101$, is unstable. The evolution of this configuration, at parallel discrete time steps $t = 1, 2, 3$ is as follows:

$$\mathcal{C}^{t=0} = \overbrace{1111}\underbrace{0100110101}\overbrace{000}$$

$$\mathcal{C}^{t=1} = 11111001011100000 = \overbrace{11111}\underbrace{0010}\overbrace{111}\overbrace{00000}$$

$$\mathcal{C}^{t=2} = \overbrace{1111}\underbrace{1001}\overbrace{111}\overbrace{00000}$$

$$\mathcal{C}^{t=3} = \overbrace{1111}\underbrace{1111}\overbrace{111}\overbrace{00000} = \overbrace{111111111111}\overbrace{00000}$$

Since $\mathcal{C}^{t=3} = 1111111111100000$ is a FP, the convergence from $\mathcal{C}^0 = \mathcal{C}^{t=0}$ was achieved in three steps, while the size of the largest (and, in this example, the only) *maximal* unstable block in $\mathcal{C}^0$ was of size ten. An example where the convergence to a FP takes the maximal number of steps, $\frac{m}{2}$, is provided by $r = 1$ and, e.g., $\mathcal{C}^0 = 11101010101...01000$.

We remark that there is a considerable amount of literature dedicated to characterizing the possible computations of the MAJORITY cellular and network automata. In particular, characterizations of the speed or rate of convergence, that is, the appropriate upper bounds on possible lengths of transient chains for 1-D CA with $\delta = MAJ$, can be found in [68]. We emphasize that our results were obtained independently of those in [68] and its references, and that the proofs we provide are different from the ones found in the literature.[25]

Let's consider a 1-D finite or infinite CA with $\delta = MAJ$, and let's assume that the CA's nodes are labeled with integers from an appropriate finite subset $S$ of $\mathbf{Z} = \{..., -3, -2, -1, 0, 1, 2, 3, ...\}$ in case of a finite cellular space, or that the set of labels is $S = \mathbf{Z}$ if the underlying cellular space is infinite. Let's assume that the initial configuration $C^{t=0} = (x^0[i])_{i \in S}$ is not made of all 0s; otherwise, we know that the all-zeros configuration is already a fixed point. If the cellular space is infinite, let's also assume that only *finitely supported* starting configurations are allowed.

We introduce the following additional short-hand notation: given a configuration $C$, define $min(C) = min\{i : i \in S \text{ and } x[i] = 1\}$, and $max(C) = max\{i : i \in S \text{ and } x[i] = 1\}$. Given a global configuration $C$ we define $n_C = max(C) - min(C) + 1$. Then, for a given (fixed) rule radius $r \geq 1$, and an arbitrary nonzero starting configuration $\mathcal{C}^0$, we have the following bound on the transient chain length $\tau$ starting from $\mathcal{C}^0$ (see *Section 4* of [68]):

$$\tau(\mathcal{C}^0) \leq r \cdot (r + n_{\mathcal{C}^0} + 1) \tag{15}$$

[25] At the time we originally established the results summarized in this subsubsection, we were not aware of some of the already existing literature, and, in particular, of the results found in [68] that we state in a re-phrased form in the sequel.

In particular, for the MAJORITY 1-D CA that has a finite number of nodes $N$, since $r$ is assumed constant, and since for every configuration $A$ certainly $n_A = O(N)$, it follows immediately that $\tau(A) = O(N)$.

Insofar as the $\frac{N}{2}$ bound on $\tau$ for $\delta = MAJ$ that we have established in Lemma 4.7 above, it turns out that both that result of ours, and the bound provided by the inequality (15), are but special cases of what has been known since the early 1980s [68].[26] Namely, the upper bound on the number of parallel iterations until convergence to a FP for $\delta = MAJ$ *as a function of the starting configuration* given in Proposition 4.2 in [68] is always at least as good as, and can be strictly sharper for appropriately chosen starting configurations than, either of the two bounds discussed above. For details and a rigorous proof, we refer the reader to *Section 4* of [68] and the relevant references found in the bibliography of that book.

The alluded to result establishes an upper bound on $\tau(\mathcal{C})$ as a function of *the number of blocks* of one or more consecutive 1s that appear in configuration $\mathcal{C}$, as opposed to our proof that is based on *the size of the largest such block*. In particular, it can be easily shown that our uniform bound $\tau \leq \frac{N}{2}$ (which holds for *all* possible starting configurations, irrespective of how many blocks of consecutive 1s they have) is a special case of the bound provided by the aforementioned Proposition 4.2 in [68], when that Proposition is applied either to finite 1-D cellular spaces with $N$ nodes, or to the infinite line when the only starting configurations $\mathcal{C}$ allowed are the finitely supported ones, whose size of finite support satisfies $max(\mathcal{C}) - min(\mathcal{C}) + 1 \leq N$.

## 4.5 Discussion and Future Directions: Towards *Genuinely Asynchronous* CA

Among other things, the results in subsubsection 4.3.3 show that, even for the very simplest (nonlinear and nonaffine) *totalistic* cellular automata [222, 223], that is, the CA whose update rules are *symmetric* in the precise sense that will be discussed in detail in Section 5, *nondeterministic interleavings* dramatically fail to capture the perfectly synchronous parallelism that characterizes the classical CA. It is not particularly surprising that one can find a parallel CA such that no sequential CA with the same underlying cellular space and the same node update rule can reproduce the identical, or even an *isomorphic* computation. However, we find it rather interesting that very profound differences (a possibility of looping vs. a guaranteed convergence to a fixed point configuration when $\Gamma$ is finite) can be observed in the simplest nonlinear, nonaffine 1-D parallel and sequential CA – namely, those with *simple threshold functions* as the node update rules, and that this profound difference does not apply merely to the individual (S)CA, but to all possible computations of the entire class of (simple) threshold CA.

Moreover, the differences in parallel and sequential computations in the case of the Boolean *XOR* update rule, for example, can be largely ascribed to the properties of the *XOR* function (see subsubsection 4.3.1). For instance, given that *XOR* is not *monotone*, the existence of temporal cycles is not at all surprising. In contrast, monotone functions such as MAJORITY are intuitively expected not to have cycles, i.e., for all converging computations, to always converge to a fixed point. This intuition about the monotone symmetric *sequential* CA is shown correct. It is actually, in a sense, "almost correct" for the parallel CA as well, in that the actual non-FP

[26] Needless to say, we were not aware of this fact at the time we proved our upper bound on $\tau$; we do point out that our proof is different from the one found in the quoted reference.

cycles can be shown to be very few *and without any incoming transients* [200, 205]. Thus, in this case, the very existence of the (rare) nontrivial temporal cycles can be ascribed directly to the assumption of *perfect synchrony* of the parallel node updates.

In the actual engineering, physical or biological systems that can be modeled by cellular automata, however, such perfect synchrony is usually hard to justify. In particular, when CA are applied to modeling various complex physical, biological or social phenomena (be those the crystal growth, the forest fire propagation, the information or gossip diffusion in a population, or the signal propagation in an organism's neural system), one ought to primarily focus on the underlying CA behaviors that are, in some sense, *dynamically robust*. This robustness may require, for instance, a *low sensitivity to small perturbations* in the initial configuration. From this standpoint, temporal cycles in the parallel threshold CA are, indeed, an idiosyncrasy of the perfect synchrony, that is, a peculiarity that is anything but robust. Likewise, it makes sense to focus one's qualitative study of the dynamical systems modeled by the threshold CA to those properties that are *statistically robust* (see, e.g., [8]). It can be readily argued in a rigorous, probabilistic sense that, again, the typical, statistically robust behavior of a simple threshold CA computation is a relatively short transient chain, followed by the convergence to a stable state (i.e., a fixed point). In particular, the non-fixed-point temporal cycles of the simple threshold CA are statistically negligible for all sufficiently large finite, as well as for all infinite CA.

After these digressions on the meaning and implications of our results on the 1-D threshold parallel and sequential *threshold* (S)CA, we now discuss some possible extensions of the results presented thus far. In particular, we have considered extending our study to *non-homogeneous* simple threshold CA, where not all the nodes necessarily update according to *one and the same* threshold update rule. The goal of that study would be to identify those properties of large-scale distributed information systems that are solely due to *heterogeneity* of the individual agent behaviors. When both the heterogeneity of the local behaviors and the nonuniformity of the inter-agent interaction patterns are allowed, as discussed in the introductory Sections of this technical report, one arrives at various graph or network automata models. The REACHABILITY problems for a class of such network automata when the update rules are restricted to simple threshold functions is addressed in [16]. We will address the problem of *counting* the fixed point configurations of simple threshold network automata in Section 6 of this technical report.

Another direction for future inquiry is to consider *linear threshold* (S)CA defined over 2-D and other higher-dimensional regular grids, as well as the (S)CA defined over regular Cayley graphs that are not simple Cartesian grids.

One of the more challenging future directions, that have already been explored in other contexts, is to consider CA-like finite automata defined over *arbitrary* (rather than only regular) graphs. Some results on phase space properties of such finite automata with threshold update rules can be found, e.g., in [16, 17].

Another possible extension is to consider classes of the node update rules beyond the simple threshold functions. One obvious candidate are the monotone functions that are not necessarily symmetric (that is, such that the corresponding CA need not be totalistic or semi-totalistic). A possible additional twist, as mentioned above, is to allow for different nodes to update according to different monotone (symmetric or otherwise) local update rules. At what point of the increasing automata complexity, if any, do the possible sequential computations "catch up" with the concurrent ones, is an interesting problem to consider. Some partial answers to this general prob-

lem can be found in the two Sections of this technical report immediately following the present Section.

Yet another direction for further investigation is to consider other models of (a)synchrony in cellular automata. We argue that the classical concurrent CA can be viewed, if one is interested in node-to-node interactions among the nodes that are not close to one another, as a class of computational models of *bounded asynchrony.* Namely, if nodes $x$ and $y$ are at distance $k$ (i.e., $k$ nodes apart from each other), and the radius of the CA update rule $\delta$ is $r$, then any change in the state of $y$ can affect the state of $x$ *no sooner*, but also *no later* than after about $\frac{k}{r}$ (parallel node update) computational steps.

We remark that the two particular classes of network automata defined over arbitrary (not necessarily regular, or Cayley) *finite* graphs, namely, the sequential and synchronous dynamical systems (SDSs and SyDSs, respectively), and their various phase space properties, have been extensively studied; see, e.g., [16, 17, 20, 148, 189, 190, 204, 196, 204, 206] and references therein. It would be interesting, therefore, to consider *asynchronous cellular and network automata*, where the nodes are no longer assumed to update in unison and, moreover, where no global clock is assumed. We again emphasize that such automata would entail what can be viewed as *communication asynchrony*, thus going beyond the kind of mere asynchrony in computation at different nodes that has been studied since at least 1984 (e.g., [89, 104, 198]).

What are, then, such *genuinely asynchronous* cellular automata like? How do we specify the local update rules, that is, the computations at different nodes, given the possible *communication delays* in what was originally a multiprocessor-like, rather than distributed system-like, parallel model? In the classical, parallel case where a perfect communication synchrony is assumed, any given node $x_i$ of a 1-D CA of radius $r \geq 1$ updates according to

$$x_i^{t+1} = f(x_i^t, x_{i_1}^t, ..., x_{i_{2r}}^t) \tag{16}$$

for an appropriate local update rule $\delta = f(x_i, x_{i_1}, ..., x_{i_{2r}})$, whereas, in the asynchronous case, the individual nodes update according to

$$x_i^{t+1} = f(x_i^t, x_{i_1}^{t_1}, ..., x_{i_{2r}}^{t_{2r}}) \tag{17}$$

We observe that $t$ in *Eqn.* (16) pertains to *the global time,* which of course in this case also coincides with the node $x_i$'s (and everyone else's) *local time.* However, in case of equation (17), each $t_j$ pertains to an appropriate *local time*, in the sense that each $x_{i_j}^{t_j}$ denotes the node $x_{i_j}$'s value that was most recently received by the node $x_i$. That is, $x_{i_j}^{t_j}$ is a local view of the node $x_{i_j}$'s state, as seen by the node $x_i$. Thus, the nonexistence of the global clock has considerable implications. How to meaningfully relate these different local times, so that one can still mathematically analyze such ACA – yet without making the ACA description too complicated[27]? Yet, if we want to study *genuinely* asynchronous CA models (rather than the arbitrary sequential models where the existence of global clocks is still tacitly assumed), these changes in the definition seem unavoidable.

We point out that this genuine, that is, *communication* asynchrony in cellular automata (see equation (17)) can also be readily interpreted in the nondeterministic terms: at each time step,

[27]That is, while staying away from introducing explicit message *sends* and *receives,* (un)bounded buffers, and the like.

a particular node updates by using its own current value, and also nondeterministically choosing the current or one of the past values of its neighbors. Such a *past value* of a node $x_{i_j}$ used by the node $x_i$ would be only required not to be any older than that value of $x_{i_j}$ that $x_i$ had used as its input on its most recent local computation, i.e., on the node $x_i$'s most recent previous turn to update. That is, from the viewpoint of what are the current inputs to any given node's update function $\delta$, there is a natural nondeterministic interpretation of the fact that the nodes have different clocks.

Many interesting questions arise in this context. One is, what kinds of the phase space properties remain *invariant* under this kind of nondeterminism? Given a triple $(\Gamma, N, M)$, it can be readily shown that the fixed points are invariant with respect to the *fair* node update orderings in the (synchronized) sequential CA, and, moreover, the FPs are the same for the corresponding parallel CA. On the other hand, as our results in subsection 4.3.3 indicate, neither cycle configurations nor transient configurations are invariant with respect to whether the nodes are updated sequentially or concurrently (and, in case of the former, in what order).

It can also be readily argued that, indeed, the (proper, stable) FPs are also invariant for the asynchronous CA and network automata, as well – provided that all the nodes have reached their respective states corresponding to the same fixed point global configuration, and that they all *locally agree* what (sub)configuration they are in, even if their individual local clocks possibly disagree with one another. Therefore, earlier results in [17] on the FP invariance for the sequential and synchronous network automata are just special cases of this, more general result.

**Theorem 4.4.** *Given an arbitrary asynchronous cellular or graph automaton, every fixed point configuration of such an automaton is invariant with respect to the choice of a node update ordering, provided that each node $x_i$ has an up-to-date knowledge of the current state of its neighborhood, $N_i$.*

By *up-to-date knowledge* of one's neighborhood, we mean that, while there may be communication delays, those delays are *bounded* so that, by the time a node's turn comes to update its state, that node will have obtained the *current* values of all the neighbors its update depends on. This assumption is highly nontrivial and it may or may not be possible to guarantee that it holds in a particular distributed computing context; however, it is certainly less restrictive than assuming the existence of a global clock. Further elaboration on the issues such as network delays and (bounded) asynchrony of communication, and how to adequately capture them in an ACA-based abstract model, however, are left for the future work.

In addition to studying various invariants under different assumptions on asynchrony and concurrency, we also consider qualitative comparison-and-contrast of the asynchronous CA that we propose, and the classical CA, SCA and NICA. Such a study would shed more light on those behaviors that are solely due to (our abstracted version of) network delays.

More generally, the communication asynchronous CA, i.e., the various nondeterministic choices for a given cellular automaton that are due to asynchrony, can be shown to subsume all possible behaviors of the classical and sequential (S)CA with the same corresponding $(\Gamma, N, M)$. In particular, the nondeterminism that arises from (unbounded) asynchrony subsumes the nondeterminism of a kind studied in subsubsection 4.3.3; but the question arises, exactly how much more expressive the former model really is than the latter.

### 4.6 Section Summary

We present herein some early steps in studying cellular automata when the unrealistic assumptions of *perfect synchrony* and *instantaneous unbounded parallelism* are dropped. Motivated by the well-known notion of the sequential *interleaving semantics* of concurrency, we try to apply this metaphor to parallel CA and thus motivate the study of sequential cellular automata, SCA, and the sequential interleavings automata, NICA. In particular, we undertake a comparison and contrast between the SCA/NICA and the classical, parallel CA models when the node update rules are restricted to *simple threshold functions.* Concretely, we show that, even in this, very simple setting, the sequential *interleaving semantics* of NICA fails to capture concurrency of the classical, parallel CA.

One immediate lesson is that, simple as they may be, the basic local operations (i.e., node updates) in the classical CA cannot always be considered atomic. That is, the fine-grain parallelism of CA turns out not to be quite fine enough for our purposes. It then appears reasonable – indeed, necessary – to consider a single local node update to be made of an ordered sequence of the finer elementary operations:

(i) Fetching all the neighbors' values ("Receiving"? "Reading shared variables"?)

(ii) Updating one's own state according to the update rule $\delta$ (that is, performing the local computation)

(iii) Informing the neighbors of the update, i.e., making available one's new state/value to the neighbors ("Sending"? "Writing a shared variable"?)

Motivated by the early results on the sequential and parallel simple threshold CA, and some of the implications of those results, we next consider various extensions. The central idea is to introduce a class of *genuinely asynchronous* CA, and to formally study their properties. This would hopefully, down the road, lead to some significant insights into the fundamental issues related to bounded vs. unbounded asynchrony, formal sequential semantics for parallel and distributed computation, and, on the cellular automata side, to the identification of many of those classical parallel CA phase space properties that are solely or primarily due to the (physically unrealistic) assumption of perfectly synchronous parallel node updates.

We also find various extensions of the basic CA model to provide a simple, elegant and useful framework for a high-level study of various global qualitative properties of distributed, parallel and real-time systems at an abstract and rigorous, yet comprehensive level. The usefulness of this formal framework for modeling and analyzing *large-scale multi-agent systems* is discussed in much more details in [191, 206]. The related discussion that specifically pertains to the network or graph automata extensions of the *finite* classical (both parallel and sequential) CA will follow in the introductory parts of Section 5.

### Appendix to Section 4: Formal Proof of Theorem 4.2, Part (ii)

As outlined in the main text, the idea behind the proof of this result is well-known: assigning an appropriate *potential function* (also referred to in the literature as *energy function*) to each global configuration according to a pre-specified rule, and then showing that, as the nodes update sequentially, this potential can either decrease, or else stay the same, with each such update – but can never increase. In particular, whenever a node changes its state, the configuration's potential

strictly decreases by a bounded-from-below amount. This *monotonicity property* will ensure that cyclic behavior is, indeed, impossible in such sequential cellular automata. Variants of this basic idea have been successfully applied to various *(discrete) Hopfield Network* and *(sequential) Network Automata* models; see, e.g., [16, 68, 81, 137, 139]. Thus, while the proof idea utilized below is not original, the potential function we shall use nicely utilizes the idiosyncracy of *simple threshold* update rules, and, in particular, is considerably simpler than the similar potential functions that can be found, for instance, in the Hopfield networks literature (e.g., [68]).

We start with two easily verifiable observations. First, it suffices to consider integer values of threshold $k$ only. Second, if $k \leq 0$, then the resulting simple threshold rule is the constant function $f(x_{i-r}, ..., x_i, ..., x_{i+r}) = 1$, for all choices of inputs $x_{i-r}, ..., x_i, ..., x_{i+r}$. A fair 1-D SCA with $n$ nodes and this degenerate update rule will converge to the fixed point $1^n$ regardless of the starting configuration; the speed of convergence depends only on after how many sequential update sub-steps has each node got its turn to update. (The fairness condition ensures that each node will get a chance to update after finitely many sub-steps.) Similar argument applies to the situation where the threshold $k \geq 2r+2$, except that this time the FP to which all computations are guaranteed to converge is $0^n$. Hence, it suffices to consider $k \in \{1, ..., 2r+1\}$. We will assume that the threshold $k$ is an integer from $\{1, 2, ..., 2r, 2r+1\}$.

We shall first establish the claim of the theorem for *finite* SCA, and then extend the result to infinite one-dimensional cellular spaces. To prove the claim for the finite 1-D cellular spaces, we will consider the circular boundary conditions; we remark that other types of boundary conditions can be treated very similarly.

We define a potential function of a configuration of an SCA defined on a ring $\Gamma$ made of $n$ nodes as follows. Each node has a potential assigned to it that measures with how many of its neighboring nodes this node (dis)agrees in terms of having different current states. Each edge has a potential assigned to it that equals 1, if the two nodes incident to this edge are in different states, and 0, otherwise. The total potential of a global configuration is then defined to be the sum of all individual node and edge potentials.

That is, for each node $x_i$,

$$P(x_i) \;=\; \begin{cases} T_1, & \text{if} \quad x_i = 1 \\ T_0, & \text{if} \quad x_i = 0 \end{cases} \tag{18}$$

where $T_1 = k - 1$ is the minimal number of neighbors $x_j \in N(x_i)$ that need to be in state 1, in order for $x_i = 1$ to evaluate to 1 at the next time step. Similarly, $T_0 = 2r + 1 - k$ is the minimal number of neighbors that currently need to be in state 0, so that it is ensured that, if $x_i^t = 0$, then also $x_i^{t+1} = 0$.

For each pair of nodes $\{x_i, x_j\}$, the *edge potential* is given by

$$P(\{x_i, x_j\}) \;=\; \begin{cases} 1, & \text{if} \quad |i-j| \;\leq r \;\text{ and }\; x_i^t \neq x_j^t \\ 0, & \text{otherwise} \end{cases} \tag{19}$$

Let us assume that, at the time step $t+1$, node $x_i$ flips: $x_i^t = 0 \;\rightarrow\; x_i^{t+1} = 1$. In order for this to happen, it must be the case at time $t$ that $\Sigma_{j:|i-j|\leq r}\, P(\{x_i, x_j\}) \geq k$. Since $x_i^t = 0$, also $P(x_i^t) = T_0 = 2r + 1 - k$. By the assumption on sequentiality of the node updates, no node

except for $x_i$ has changed its state at this time step. Hence, the total potential of the entire configuration at time step $t$ satisfies

$$\begin{aligned} P^t &= \Sigma_{\{j:j\neq i\}}\ P^t(x_j)\ +\ P^t(x_i)\ +\ \Sigma_{\{e:\ not\ x_i\}}\ P^t(e)\ +\ \Sigma_{\{e:x_i\}}P^t(e) \\ &\geq \Sigma_{\{j:j\neq i\}}\ P^t(x_j)\ + 2r + 1 - k\ + \Sigma_{\{e:\ not\ \ x_i\}}\ P^t(e)\ +\ k \\ &\geq \Sigma_{\{j:j\neq i\}}\ P^t(x_j)\ +\ \Sigma_{\{e:\ not\ \ x_i\}}\ P^t(e)\ +\ 2r + 1 \end{aligned}$$

where the shorthand $\{e : not\ x_i\}$ means that the edge $e$ is not incident to the node $x_i$, whereas $\{e : x_i\}$ stands for the edge $e$ being incident to the node $x_i$. Hence, $P^t\ \geq\ \Sigma_{\{j:j\neq i\}}\ P^t(x_j)\ +\ \Sigma_{\{e:\ not\ \ x_i\}}\ P^t(e)\ +\ 2r + 1$.

On the other hand, at time $t+1$, $P^{t+1}(x_i) = T_1 = k - 1$ and $\Sigma_{\{e:\ x_i\}}\ P^{t+1}(e)\ \leq 2r - k$ and, consequently,

$$\begin{aligned} P^{t+1} &= \Sigma_{\{j:j\neq i\}}\ P^{t+1}(x_j)\ +\ P^{t+1}(x_i)\ +\ \Sigma_{\{e:\ not\ x_i\}}\ P^{t+1}(e)\ +\ \Sigma_{\{e:\ x_i\}}\ P^{t+1}(e) \\ &\leq \Sigma_{\{j:j\neq i\}}\ P^t(x_j)\ + k - 1 +\ \Sigma_{\{e:\ not\ x_i\}}\ P^t(e)\ +\ 2r - k \\ &\leq \Sigma_{\{j:j\neq i\}}\ P^t(x_j)\ +\ \Sigma_{\{e:\ not\ x_i\}}\ P^t(e)\ +\ 2r - 1 \end{aligned}$$

That is, $\Delta P = P^{t+1} - P^t \leq -2$, i.e., each sequential node update that changes the state of the updating node from 0 to 1 *decreases the overall potential* of the entire configuration by at least 2.

Hence, starting from an arbitrary global configuration, once any node flips from 0 to 1, there is no going back, and cycling behavior is impossible.

Similar case analysis shows that no cycling is possible if some node flips from 1 to 0. Therefore, no temporal cycles are possible in (finite) Simple Threshold SCA, irrespective of the choice of a node update sequence. Hence, no temporal cycles are possible in Simple Threshold NICA, either.

While the shown construction applies to the *finite* one-dimensional cellular spaces only, it is immediate that, by assigning potentials to appropriate *finite subconfigurations*, it can be proven that the temporal cycles are also impossible in case of the *infinite* Simple Threshold SCA and, therefore, infinite Simple Threshold NICA, as well.

# 5 Some Configuration Space Properties of Sequential and Synchronous Dynamical Systems

In this Section, we continue with addressing the general problem of determining and predicting the long-term and emerging system behavior in complex, large-scale and distributed computational and communication infrastructures from a general, abstract complex systems perspective. Certain classes of *network* or *graph automata* will be defined that can be used as an idealization of the classical networked distributed systems, as well as of various software or robotic multi-agent systems, social networks and *ad hoc* communication networks, and as a theoretical model for the computer simulation of a variety of engineering, social, and socio-technical distributed infrastructures. These network automata will be defined so that they appropriately generalize the parallel and sequential *finite* cellular automata in two important respects. One, the network/graph automata discussed in the sequel will be defined over more general interconnection topologies than the highly regular, grid-like underlying graphs of the classical CA. Two, the individual agents in the models studied in the rest of this technical report will be allowed to behave differently from one another. Thus, the graph automata in this Section and the next will be characterized by *heterogeneity* both in terms of the individual agents' behaviors and the interaction patterns among different agents [192, 196].

We will study in detail several fundamental *configuration space properties* of such graph or network automata: what are the possible *global behavior patterns* of the entire system, given the simple local behaviors of its components, and the interaction (that is, coupling) among those components. We shall specifically focus on the problem of determining *how many* global stable configurations such network automata have, and *how hard* is the computational problem of *counting* those stable configurations. We shall also address some other counting problems, such as the problem of enumerating all unreachable global configurations or all predecessor configurations of a given configuration of a network automaton. As corollaries to our results on the computational complexity of counting, we will establish several complexity theoretic results about some of the closely related decision problems, as well.

In the present Section, we shall argue that all fundamental counting problems about Boolean Sequential and Synchronous Dynamical Systems are computationally intractable [194, 196, 204, 206]. Moreover, we will show that this intractability of counting holds when the local update rules of the nodes in these network automata models are restricted to either symmetric or monotone functions. As corollaries to the (weakly) parsimonious reductions used to establish these results on the computational intractability of counting, we will also show that several important decision problems originally addressed in [17], and some of their variants, are also computationally intractable; see also [190, 194] for more details. In the next Section, we shall make the complexity of counting results even sharper, by further restricting the instances of Sequential and Synchronous Dynamical Systems we consider, chiefly in terms of the *uniform sparseness* of their underlying graphs. These restricted instances will appear rather close to the ordinary (finite) cellular automata, yet, as we will show, these structural similarities turn out to be misleading insofar as the resulting dynamics, that is, the behavioral properties, are concerned.

The rest of this Section is organized as follows. We first motivate the network automata based approach to abstracting large-scale MAS and other distributed infrastructures, and briefly

discuss what kind of insights about the MAS *collective dynamics* behavior one can hope to acquire via a formal study of the corresponding network automata's configuration space properties [206]. We then introduce the two classes of such network automata that will be the central subject of the rest of this Section, as well as the next Section; these two models are called *Sequential* and *Synchronous Dynamical Systems* (SDSs and SyDSs, respectively). We then summarize our main results, and survey the most relevant literature on both the classical cellular automata, and their various network or graph automata generalizations.

The original technical results about the Boolean SDSs and SyDSs with *arbitrary* local update rules are stated and proved in subsection 5.4. While the central theme is the computational complexity of counting special types of configurations such as the fixed points, we also discuss in that subsection some other consequences of our constructions used to establish the hardness of counting; those consequences include several corollaries about the related decision problems.

We then pursue studying Boolean SDSs and SyDSs that are appropriately restricted in terms of their (i) underlying graphs and/or (ii) update rules. We start with establishing that counting fixed point and garden of Eden configurations remains **#P**-complete even if an S(y)DS's underlying graph is planar, bipartite and very sparse on average; these results can be found in subsection 5.6. Moreover, these hardness results are shown to hold even if, simultaneously with the stated restrictions on the structure of the underlying graphs, the local update rules are restricted to *monotone* Boolean functions given as negation-free formulae, and where the encoding of these formulae is considered to be a part of the overall description of a problem instance.[28]

After that, in subsection 5.7 we establish the intractability of counting the fixed point configurations for the SDSs and SyDSs with their update rules restricted to *symmetric* Boolean functions.

Last but not least, the final subsection summarizes and briefly discusses the importance of the results established earlier in the Section, as well as sets the stage for Section 6.

## 5.1 Introduction and Motivation

In order to be able to predict the long-term behaviors of various decentralized engineering, social and socio-technical systems and networks, one may want to, first, abstract those infrastructures and translate them into formal dynamical systems, and, second, answer questions about those complex systems' *collective dynamics* [192, 206]. The computational hardness of the resulting problems about the *configuration space* properties of interest would then provide *lower bounds* on analyzing the dynamics and emergent behavior of the actual distributed computational and communication networks and other decentralized infrastructures, and on how predictable their long-term behavior can be expected to be. That is, a formal computational intractability of an idealized configuration space problem defined for an appropriate class of cellular or network automata would certainly imply that, in general, the long-term behavior of the corresponding actual distributed infrastructure cannot be reliably predicted, i.e., that there is no short-cut to a

[28] For a detailed discussion on the implications of the *size* of a problem's description, and in particular whether the local update rules "count" as a part of that description – and, if they do, how exactly are they encoded – we refer the interested reader to our paper [190].

step-by-step system execution[29] [17, 189, 206].

We study certain classes of *network* or *graph automata* that can be used as an abstraction of the classical networked distributed systems, as well as of various multi-agent systems and *ad hoc* communication networks, and as a theoretical model for the computer simulation of a broad variety of computational, physical, social, and socio-technical distributed infrastructures [12]. In several research papers that are either related to or have *de facto* contributed portions of this technical report (see, e.g., [12, 13, 14, 15, 16, 17, 18, 19, 20, 132, 198, 192, 196, 200, 204, 206]), the general approach has been to study mathematical and computational *configuration space properties* of such network automata: what are the possible *global behavior patterns* of the entire system, given the simple local behaviors of its components, and the interaction pattern among those components.

In this Section as well as the next one, the emphasis will be given to the problems of determining *how many* configurations of a particular kind these network automata have, and *how hard* are the computational problems of *counting* (that is, enumerating) those various types of configurations. Among several different types of configurations that are of interest, *stable* or *fixed point* configurations have been particularly prominent, both in our own work [189, 190, 192, 196, 204, 206], and in the related literature on cellular and network automata in general (e.g., [55, 56, 63, 159]). We will also study the complexity of counting in the context of several other types of configurations, including the unreachable global states (also called *gardens of Eden* [133]), as well as the predecessor and the arbitrary ancestor configurations of a given global state [70].

In a nutshell, the contributions of our research summarized in the rest of this Section are as follows. We prove that both exact and approximate counting of the number of the *fixed point* configurations in *Sequential* and *Synchronous Dynamical Systems* with Boolean update rules are computationally intractable. This intractability holds even when each node is required to update according to either a *symmetric* Boolean function or a *monotone* function. We also show that the problems of exact counting of the *garden of Eden* configurations, as well as of all *transient configurations*, and all predecessors of a given configuration, are in general also computationally intractable. Moreover, all these counting problems remain hard, in case of the general as well as monotone Boolean update rules, when the underlying graphs of Sequential or Synchronous Dynamical Systems are required to be *planar*, *bipartite*, and *very sparse on average* [188, 194].

The Sequential and Synchronous Dynamical Systems (SDSs and SyDSs), as the network automata models of our choice, are introduced next.

## 5.2 Sequential and Synchronous Dynamical Systems

*Sequential Dynamical Systems* (henceforth referred to as SDSs) were originally proposed in [19, 20, 21] as an abstract model for computer simulations. This model has been successfully applied in the development of large-scale socio-economic simulation systems such as the *TRANSIMS* project at the Los Alamos National Laboratory [24].

A Boolean SDS $\mathcal{S} = (G, \mathcal{F}, \pi)$ consists of three components. $G(V, E)$ is an undirected graph with $n$ nodes with each node having a 1-bit state. $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$, with $f_i$ denoting a symmetric Boolean function associated with node $v_i$. $\pi$ is a permutation of (or a total order on)

[29] Or, at least, step-by-step computer simulation; see the motivation of the SDS model below for a more detailed discussion and references.

the nodes in $V$. A *configuration* of an SDS is an $n$-bit vector $(b_1, b_2, \ldots, b_n)$, where $b_i$ is the value of the state of node $v_i$ $(1 \leq i \leq n)$. A single SDS transition from one configuration to another is obtained by updating the state of each node using the corresponding Boolean function. These updates are carried out in the order specified by $\pi$. If the permutation $\pi$ is omitted, and all the nodes update synchronously in parallel (the way the nodes of classical *cellular automata* update), we arrive at the definition of *Synchronous Dynamical Systems* (SyDSs).

SDSs and SyDSs are closely related to the classical Cellular Automata (CA), a widely studied class of dynamical systems in physics and complex systems. SDSs generalize finite sequential CA [198, 200] in that they allow *arbitrary (finite) underlying graphs*, as opposed to restricting cellular spaces to regular Cayley graphs only [63]. SDSs are also closely related to another extension of the classical CA called *graph automata* [135, 119] and to *one-way cellular automata* studied by Roka [153]. The main difference between the graph automata in [135] and the SDSs is the sequential ordering aspect. In fact, the graph automata of [135] are equivalent with SyDSs with arbitrary finite domains.

Several other researchers (e.g., [60, 83, 153]) have also considered this particular aspect of the node update ordering. In particular, Huberman and Glance [83] discuss experimentally how certain simulations of $n$-person games exhibit very different (but probably more realistic) dynamics when the cells are updated sequentially as opposed to when they are updated in parallel. The issue of sequential ordering has been also discussed in [19, 13, 132] in the context of developing a theory of large-scale simulations, as well as in our prior work [191, 198, 200], in the context of investigating appropriate cellular automata based abstractions for studying the collective dynamics of large-scale multi-agent systems.

### 5.2.1 Formal Definitions of SDS and SyDS Models

**Definition 5.1.** *A* Sequential Dynamical System (SDS) $\mathcal{S}$ *is a triple* $(G, F, \Pi)$, *whose components are as follows:*

1. $G(V, E)$ *is a connected undirected graph without multi-edges or self-loops.* $G = G_{\mathcal{S}}$ *is referred to as the* underlying graph *of* $\mathcal{S}$. *We often use* $n$ *to denote* $|V|$ *and* $m$ *to denote* $|E|$. *The nodes of* $G(V, E) = G_{\mathcal{S}}$ *are enumerated* $v_1, v_2, \ldots, v_n$.

2. *Each node is characterized by its* state. *The state of a node* $v_i$, *denoted by* $s_i$, *takes on a value from some finite domain,* $\mathcal{D}$. *In this technical report, we shall primarily focus on* $\mathcal{D} = \{0, 1\}$. *We use* $d_i$ *to denote the degree of the node* $v_i$. *Each node* $v_i$ *has an associated* node update rule $f_i : \mathcal{D}^{d_i+1} \rightarrow \mathcal{D}$, *for* $1 \leq i \leq n$. *We also refer to* $f_i$ *as the* local transition function. *The inputs to* $f_i$ *are* $s_i$ *and the current states* $s_j$ *of the neighbors of* $v_i$. *We use* $F = F_{\mathcal{S}}$ *to denote the* global map *of* $\mathcal{S}$, *obtained by appropriately composing together all the local update rules* $f_i$, $i = 1, ..., n$.

3. *Finally,* $\Pi$ *is a permutation of* $V = \{v_1, v_2, \ldots, v_n\}$ *specifying the order in which the nodes update their states using their local transition functions. Alternatively,* $\Pi$ *can be envisioned as a total ordering on the set of nodes* $V$. *In particular, we can view the global map as a sequential composition of the local actions of each* $f_i$ *on the respective state* $s_i$, *where the node states are updated according to the order* $\Pi$; *that is,* $F_{\mathcal{S}} = (f_{\Pi^{-1}(v_1)}, f_{\Pi^{-1}(v_2)}, \ldots, f_{\Pi^{-1}(v_n)})$.

The nodes are processed in the *sequential* order specified by the permutation $\Pi$. The processing associated with a node consists of computing the new value of its state according to the node's update function, and changing its state to this new value. In the sequel, we shall often slightly abuse the notation, and not explicitly distinguish between an SDS's or SyDS's node itself, $v_i$, and its state, $s_i$.[30]

We shall discuss in more detail several possible interpretations of the global map $F = F_{\mathcal{S}}$, and how this map acts on the (global) configurations of an S(y)DS $\mathcal{S}$, in subsubsection 5.2.2.

**Definition 5.2.** *A Synchronous Dynamical System (SyDS)* $\mathcal{S}' = (G, F)$ *is an SDS* without *the node permutation. In an SyDS, at each discrete time step, all the nodes perfectly synchronously in parallel compute and update their state values.*

Thus, SyDSs are similar to the finite classical parallel *cellular automata* (CA) [63, 68, 70, 75, 225, 226], except that in an SyDS the nodes may be interconnected in an arbitrary fashion, whereas in a classical cellular automaton the nodes are interconnected in a regular fashion (such as, e.g., a line, a rectangular grid, or a hypercube). Another difference is that, while in classical CA all nodes update according to the same rule, in an SyDS different nodes, in general, may use different update rules [196].

In the sequel, we shall often slightly abuse the notation, and not explicitly distinguish between an SDS's or SyDS's node itself, $v_i$, and its state, $s_i$. The intended meaning will be clear from the context. We shall discuss in more detail several possible interpretations of the global map $F = F_{\mathcal{S}}$, and how this map acts on the (global) configurations of an S(y)DS $\mathcal{S}$, in subsubsection 5.2.2.

Most of the early work on sequential dynamical systems has focused primarily on the SDSs with *symmetric Boolean functions* as the node update rules [12, 18, 13, 14, 17, 19, 20]. By *symmetric* is meant that the future state of a node does not depend on the order in which the input values of this node's neighbors are specified. Instead, the future state depends only on $\Sigma_{j \in N(i)}\, x_j$ (where $N(i)$ stands for the *extended neighborhood* of a given node, $i$, that includes the node $i$ itself), i.e., on how many of the node's neighbors are currently in the state 1. Thus symmetric Boolean SDSs correspond to *totalistic (Boolean) cellular automata* of Wolfram [225, 226].

The assumption about *symmetric* Boolean functions can be easily relaxed to yield more general SDSs [17]. We give special attention to the symmetry condition for two reasons. First, our computational complexity theoretic lower bounds for such SDSs imply stronger lower bounds for determining the corresponding configuration space properties[31] of the more general classes of network automata and communicating finite state machines (CFSMs). Second, symmetry

[30] This we do in the constructions where there are different types of nodes involved; for instance, in several situations in this Section as well as the next, an SDS will have those nodes corresponding to *variables* of a Boolean formula from which we are constructing that SDS, the nodes corresponding to *clauses* in the formula, and possibly some additional, auxiliary nodes that do not correspond to either variables or clauses in the formula. Since we do want to distinguish those different types of nodes, having a separate symbol for a given kind of a node and its current state would double the number of symbols used and make the notation rather cumbersome. We will ensure, however, that the intended meaning – that is, when we say, e.g., '$x_i'$, whether we mean a variable in the formula, a node of an SDS, or this node's state – will always be clear from the context.

[31] Configuration spaces of sequential and synchronous dynamical systems will be defined in subsubsection 5.2.2.

provides one possible way to model the *mean field effects* frequently encountered in statistical physics and studies of other large-scale systems. Similar assumptions are made in [33, 204, 206].

Insofar as other restricted classes of Boolean update rules that we shall consider in the sequel, the strongest results of this Section are shown in the context of Boolean S(y)DSs defined on the star graphs and with the node update rules restricted to *monotone functions*.

**Definition 5.3.** *Given two Boolean vectors,* $X = \langle x_1, \ldots, x_n \rangle$ *and* $Y = \langle y_1, \ldots, y_n \rangle$, *define a binary relation "*$\preceq$*" as follows:* $X \preceq Y$ *if* $x_i \leq y_i$ *for all* $i$, $1 \leq i \leq n$, *where the partial order* $\leq$ *on the Boolean domain* $\{0, 1\}$ *is defined by* $0 \leq 1$ *and* $1 \not\leq 0$. *A Boolean function of* $n$ *variables* $f = f(x_1, ..., x_n)$ *is* monotone *if* $X \preceq Y$ *implies that* $f(X) \leq f(Y)$.

Notice that the notion of monotonicity given in Definition 5.3 above allows us to compare only Boolean vectors of the same length.

S(y)DSs with finite domains are a generalization of S(y)DSs with Boolean domains. In the most general class of the S(y)DSs with arbitrary finite domains, denoted (FIN, NONE)-SDSs(or FIN-S(Y)DSs for short), there are no restrictions on the local transition functions. All the hardness results in this technical report, explicitly shown for the Boolean S(y)DSs, clearly also hold for the more general, non-Boolean finite domains – as long as those domains' sizes are $O(1)$, which we will assume throughout. For that reason, as well as in order to elucidate the comparison and contrast with the binary-valued *discrete Hopfield networks* (DHNs) and the finite Boolean-valued cellular automata in the later parts of the next Section, we shall focus on the Boolean SDSs and SyDSs – those whose local update rules are Boolean-valued functions of an appropriate number of Boolean variables as inputs.

An example of a very simple Boolean SDS, with three nodes that take turns and each node updates its state according to the Boolean *OR* rule applied to the *current* states of itself and the other two nodes, is given in *Figure* 3.

In *Figure* 3, if the SDS is *with memory*, then each node's future state depends on its own current state, as in the equations given below the diagram. If, however, the SDS is *memoryless*, then each node's future state depends on the current states of its two neighbors *only*, but not on its own current state; for instance, for the node $x$, instead of $x^{t+1} = x^t \vee y^t \vee z^t$, in the memoryless case we would then have $x^{t+1} = y^t \vee z^t$. The (slight) difference in behavior between the memoryless and the memorizing 3-node *OR* SDS will be shown in the next subsubsection.

### 5.2.2 SDS and SyDS Configuration Space Properties

A configuration of an SDS or SyDS $\mathcal{S} = (G, F, \Pi)$ is a vector $(b_1, b_2, \ldots, b_n) \in \mathcal{D}^n$. A configuration $\mathcal{C}$ can also be thought of as a function $\mathcal{C} : V \to \mathcal{D}$.

The function computed by SDS $\mathcal{S}$, denoted by $F_{\mathcal{S}}$, specifies for each configuration $\mathcal{C}$ the next configuration $\mathcal{C}'$ reached by $\mathcal{S}$ after carrying out the updates of the node states in the order given by $\Pi$. Thus, the function $F_{\mathcal{S}} : \mathcal{D}^n \to \mathcal{D}^n$ is a total function on the set of global configurations. This function therefore defines the dynamics of the SDS $\mathcal{S}$. We say that $\mathcal{S}$ moves from a configuration $\mathcal{C}$ to a configuration $F_{\mathcal{S}}(\mathcal{C})$ in a single transition step. Alternatively, we say that SDS $\mathcal{S}$ moves from a configuration $\mathcal{C}$ at time $t$ to a configuration $F_{\mathcal{S}}(\mathcal{C}) = \mathcal{C}'$ at time $t+1$. Assuming that each node update function $f_i$ is computable in time polynomial in the size

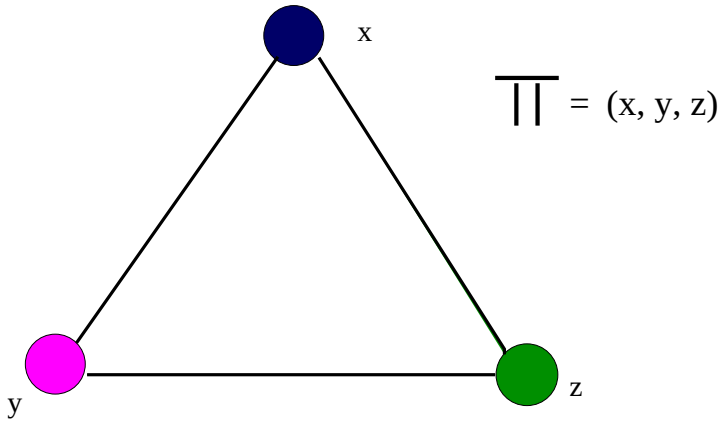



Figure 3: A Boolean SDS with three interconnected nodes. Each node locally updates its state according to the Boolean *OR* function. The sequence of node updates is $\Pi^\omega = (x, y, z)^\omega$.

of the description of $\mathcal{S}$, clearly each transition step will also take polynomial time in the size of the SDS's description. The initial configuration of an SDS $\mathcal{S}$ will be often denoted by $\mathcal{C}^0$ in the sequel. Given an SDS $\mathcal{S}$ with the initial configuration $\mathcal{C}^0$, the configuration of $\mathcal{S}$ after $t$ time steps is denoted by $\mathcal{C}(\mathcal{S}, t)$, or, more succinctly, $\mathcal{C}^t$; hence, in particular, $\mathcal{C}(\mathcal{S}, 0) = \mathcal{C}^0$.

The *configuration space*[32] $\mathcal{P}_\mathcal{S}$ of an SDS or SyDS $\mathcal{S}$ is a directed graph defined as follows. There is a vertex in $\mathcal{P}_\mathcal{S}$ for each global configuration of $\mathcal{S}$. There is a directed edge from a vertex representing configuration $\mathcal{C}$ to that representing configuration $\mathcal{C}'$ if $F_\mathcal{S}(\mathcal{C}) = \mathcal{C}'$. Since every SDS or SyDS is a *deterministic* dynamical system, each vertex in its configuration space has the out-degree of 1. Since the domain $\mathcal{D}$ of state values is assumed finite, and the number of nodes in the SDS is finite, the number of configurations in the phase space is also finite. If the size of the domain (that is, the number of possible states of each node) is $|\mathcal{D}|$, then the number of global configurations in $\mathcal{P}_\mathcal{S}$ is $|\mathcal{D}|^n$.

**Definition 5.4.** *Given two configurations $\mathcal{C}$ and $\mathcal{C}'$ of an SDS or SyDS $\mathcal{S}$, configuration $\mathcal{C}$ is a* **predecessor** *of $\mathcal{C}'$ if $F_\mathcal{S}(\mathcal{C}) = \mathcal{C}'$, that is, if $\mathcal{S}$ moves from $\mathcal{C}$ to $\mathcal{C}'$ in* one *global transition step.*

**Definition 5.5.** *Given two configurations $\mathcal{C}$ and $\mathcal{C}'$ of an S(y)DS $\mathcal{S}$, $\mathcal{C}$ is an* **ancestor** *of $\mathcal{C}'$ if there is a positive integer* t *such that $F_\mathcal{S}{}^t(\mathcal{C}) = \mathcal{C}'$, that is, if $\mathcal{S}$ evolves from $\mathcal{C}$ to $\mathcal{C}'$ in one or more transitions.*

In particular, a predecessor of a given configuration $\mathcal{C}'$ is trivially also its ancestor.

[32] Also sometimes called *phase space* in the literature; we shall treat the two terms as synonymous and use them interchangeably.

**Definition 5.6.** *A configuration $\mathcal{C}$ of an S(y)DS $\mathcal{S}$ is a* **garden of Eden (GE)** *configuration if $\mathcal{C}$ has no predecessor.*

**Definition 5.7.** *A configuration $\mathcal{C}$ of an S(y)DS $\mathcal{S}$ is a* **fixed point (FP)** *configuration if $F_{\mathcal{S}}(\mathcal{C}) = \mathcal{C}$, that is, if the transition out of $\mathcal{C}$ is to $\mathcal{C}$ itself.*

Note that a *fixed point* is a configuration that is always among its own predecessors; in general, a configuration can have zero, one or more predecessors.

**Definition 5.8.** *A configuration $\mathcal{C}$ of an S(y)DS is a* **cycle configuration (CC)** *if there exists an integer $t \geq 2$ such that*
*(i)* $F_{\mathcal{S}}{}^{t}(\mathcal{C}) = \mathcal{C}$; *and*
*(ii)* $F_{\mathcal{S}}{}^{q}(\mathcal{C}) \neq \mathcal{C}$, *for any integer $q$, $0 < q < t$.*
*Integer* **t** *above is called the* **period** *or* **length** *of the temporal cycle.*

In other words, $\mathcal{C}$ is a *cycle configuration* if it is reachable from itself in two or more transitions, but not in a single transition. Equivalently, $\mathcal{C}$ is a cycle configuration *if and only if* it is its own ancestor, but not a predecessor.

**Definition 5.9.** *A configuration $\mathcal{C}$ of an S(y)DS is a* **transient configuration (TC)** *if $\mathcal{C}$ is neither a fixed point nor a cycle configuration.*

As their name suggests, transient configurations, unlike fixed points or cycle configurations, are never revisited. We note that a GE configuration is a special case of a transient configuration; a GE configuration is not reachable from *any* configuration including itself [17]. We observe that a configuration in the phase space of an SDS may have multiple predecessors. This means that the time evolution map $F$ of an SDS or SyDS is in general *not invertible* but is *contractive*. The existence of configurations with multiple predecessors also implies that certain configurations have no predecessors. A configuration with no predecessors is called a *garden of Eden* configuration (see Definition 5.6). Such configurations can occur only as the initial states and can never be generated during the time evolution of an SDS or SyDS.

The configuration space of the triangle SDS from *Figure* 3 is given in *Figure* 4 below.

The configuration space in the upper part of *Figure* 4 pertains to the corresponding *XOR* SDS *with memory*, where the future state of a node depends on the node's own present state. The configuration space at the bottom is that of the corresponding *memoryless* SDS or SCA, where each node computes the Boolean OR of the current states of its two neighbors *only*, that is, *excluding* its own current state.

## 5.3 Summary of Results and Related Work

Given an SDS or SyDS $\mathcal{S}$, let $|\mathcal{S}|$ denote the size of the representation of $\mathcal{S}$. In general, this includes the number of nodes, the number of edges, and the description of the local transition functions. When $\mathcal{D} = \{0, 1\}$ and the local transition functions are given as the truth tables, $|\mathcal{S}| = O(m + |T|n)$, where $|T|$ denotes the maximum size of a table, $n$ is the number of nodes and $m$ is the number of edges in the underlying graph. By *the size of the truth table* we shall throughout the paper mean, for simplicity, just the number of rows in that truth table. Thus, for a node $v_i$ of degree $d_i$, the size of the truth table specifying an arbitrary Boolean function

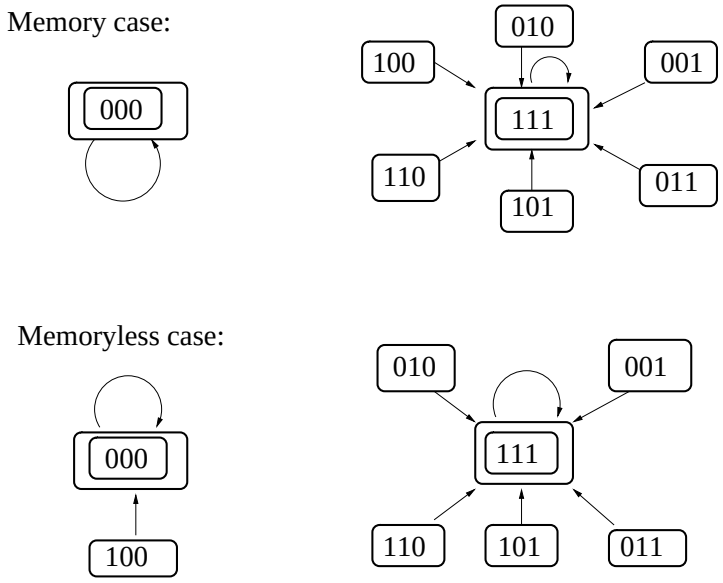



Figure 4: Configuration space of the Boolean SDS given in *Figure* 3.

is $O(2^{d_i})$, and actually, for every sufficiently large positive integer $d_i$, most Boolean functions on $d_i + 1$ inputs cannot be encoded substantially more succinctly than via a truth table of size $\Omega(\frac{2^{d_i}}{p(d_i)})$, for some polynomial $p(d_i)$.

In contrast, the size of the optimally succinct truth table fully specifying an arbitrary *symmetric* Boolean function is only $O(d_i)$ [17, 189, 206].

Another, more common way of specifying the local transition functions is via *Boolean formulae*. Unless explicitly stated otherwise, we shall assume that the local update rules $f_i$ of *non-symmetric* SDSs and SyDSs considered in the sequel are indeed given as (reasonably succinct[33]) Boolean formulae of appropriately restricted kinds. It follows from the discussion above that, for symmetric Boolean update rules, the exact way these update rules are encoded in an S(y)DS is inconsequential, as long as this encoding is reasonably succinct (see the footnote). We shall also assume that evaluating any local transition function $f_i$, given its input values, can be done in polynomial time. That will ensure that a full global update of an SDS or SyDS, where each of the $|V|$ nodes gets to update its state exactly ones, is also done in polynomial time. Since we are mainly interested in establishing the boundary between those S(y)DS configuration space properties that can determined in (deterministic) polynomial time on the one hand, and those that, under the usual assumptions in computational complexity theory, that *cannot*, this coarse-grain view of the resource requirements of an SDS's or an SyDS's computations will suffice for our purposes.

As already remarked in the introduction to this Section, we study the problems of *counting* the fixed point (FP) and other types of configurations of Boolean SDSs and SyDSs. In particular, we prove the following results:

[33]By *reasonably succinct Boolean formulae* we mean, the formulae whose sizes are not artificially blown up by, e.g., repeating the same clause(s) over and over again.

- the problems of counting FPs and GEs in the general Boolean (and, consequently, also in any other finite domain) SDSs and SyDSs are **#P**-complete;
- these two, as well as some other, related counting problems remain **#P**-complete even when the underlying graphs of Boolean S(y)DSs are required to be both *planar* and *bipartite*;
- these hardness results still hold when the node update rules of these S(y)DSs are restricted to *monotone Boolean functions*;
- moreover, the results remain valid even when *only two* different monotone update rules are used, and when the average node degree in the underlying graph is bounded by 2 – that is, when the underlying graph is very sparse *on average*;
- counting FPs of Boolean SDSs and SyDSs is also intractable when the node update rules of these S(y)DSs are restricted to *symmetric Boolean functions*.

We will also show that several other interesting configuration space properties of Boolean SDSs and SyDSs are, in the worst-case, computationally intractable to determine. Those properties will include both several other counting problems beside the problem of counting the fixed points, and some decision problems about the existence of particular types of configurations in an S(y)DS's phase space.

### 5.3.1 A Summary of Related Work on Graph and Network Automata

Various computational (including but not limited to computational complexity) aspects of cellular automata have been studied by a number of researchers; see for example [40, 39, 72, 75, 130, 179, 224, 225, 226]. Much of that work addresses decidability of various properties for infinite CA. Insofar as the computational complexity of fundamental problems about *finite* CA are concerned, we single out the following. The first **NP**-complete problems for CA are shown by Green in [72]; these problems are of a general *reachability* flavor, i.e., they address the properties of the *forward dynamics* of CA. Sutner addresses the *backward dynamics* problems, such as the problem of an arbitrary configuration's *predecessor existence*, and their computational complexity in [179]. In the same paper, Sutner establishes the efficient solvability of the predecessor existence problem for any CA with a *fixed neighborhood radius*. In [48], Durand solves the injectivity problem for arbitrary 2-D CA but restricted to the finite configurations only; that paper contains one of the first results on **coNP**-completeness of a natural and important problem about CA. Furthermore, Durand addresses the *reversibility problem* in the same, two-dimensional CA setting in [49].

As already mentioned in the introductory sections of this technical report, a considerable variety of generalizations of the classical cellular automata can be found in the literature. We point out, however, that the variety of different names used, and motivations behind, those various generalizations considerably exceeds the number of fundamentally different models. Perhaps the most obvious dimension along which CA can be generalized, and the most widely exploited one, is that of allowing more general underlying graphs or cellular spaces. The other most frequently encountered generalization of CA is with respect to homogeneity vs. heterogeneity of the node update rules, i.e., whether all the nodes behave the same, or, in contrast, different nodes are allowed to update according to different update rules. The most common names for the graph or

network automata models that generalize the classical CA in those two respects that one finds in the literature are *network automata*, *random (Boolean) networks* and *automata networks* [63]. The terminology, as well as what motivation is behind generalizing the cellular spaces and/or allowing for the heterogeneous update rules, typically depends on the particular research community. For instance, *Kauffman's networks* in theoretical biology may be elsewhere (say, among the complex dynamical systems research community) simply referred to as *random networks*. In our own work, the motivation primarily stems from our interest in the collective dynamics of large-scale multi-agent systems made of autonomous robotic, software and/or human agents. We provide some motivation for the cellular and network automata based approach to studying behavior of large agent ensembles in [191, 206], as well as in the introductory Sections of this technical report.

Among a variety of network automata models and their applications found in the literature, Sequential and Synchronous Dynamical Systems have been most prominent in the context of computer simulation of various large-scale socio-technical systems and decentralized infrastructures [12]. SDSs and SyDSs investigated in this technical report are also closely related to the *Graph Automata* (GA) models studied in [119, 135] and the *One-Way Cellular Automata* studied by Roka in [153]. In fact, the general finite-domain SyDSs exactly correspond to the Graph Automata of Nichitiu and Remila as defined in [135].

Barrett, Mortveit and Reidys [19, 20, 132, 148] and Laubenbacher and Pareigis [110] investigate the mathematical properties of sequential dynamical systems. Barrett et al. study the computational complexity of several phase space problems for SDSs. These include Reachability, Predecessor existence and Permutation existence [14, 16]. Problems related to the existence of *garden of Eden* and *fixed point* configurations are studied in [17]. In particular, the basic **NP**-completeness results for the problems of the fixed point, garden of Eden and non-unique predecessor existence in various restricted classes of Boolean S(y)DSs are proven in that paper. Algorithms for efficiently finding an FP in certain other restricted classes of S(y)DSs can be also found in [17]. Our results in this Section can be viewed as a natural partial extension of the work in [17]: instead of the appropriate *decision problems* about the fixed points and gardens of Eden in SDSs and SyDSs, we shall focus in the sequel on studying the related *counting problems*.

Among various restricted classes of Boolean SDSs and SyDSs, those with the local update rules restricted to *symmetric functions* have received particular attention (e.g., [20, 110, 132, 204, 206]). Computational complexity of the reachability-related problems in the context of, among other restricted types, the symmetric Boolean SDSs and SyDSs is investigated in [16]. We will show in this Section that, in contrast to the computational feasibility of the problem of their reachability [16], the problem of *counting* the stable configurations (i.e., our FPs) in symmetric SDSs and SyDSs, under the usual assumptions in computational complexity theory, is intractable; see also [204, 206]. We will also show that counting various types of configurations, such as FPs and GEs, is computationally intractable in the context of *monotone* Boolean SDSs and SyDSs; we originally established those results on monotone S(y)DSs (that will be summarized later in this Section) in [190, 188, 196].

## 5.4 On the Computational Complexity of Counting

Our results in this Section constitute an immediate extension of the work presented in [16] and [17]. In particular, the computational complexity of *decision problems* about the fixed point and the garden of Eden configurations in Boolean S(y)DSs is studied in [17]. Once the **NP**-

completeness of these decision problems has been established, a natural next step is to determine the computational complexity of the related *counting problems*: how many FPs, GEs, or other configurations of interest an SDS or SyDS of a given type may have.

One would intuitively expect that, for instance, counting the FPs of an arbitrary Boolean SDS or SyDS is no easier than counting the satisfying truth assignments of an arbitrary instance of the SATISFIABILITY problem [62, 140]. The intuitive notion of computational hardness of counting problems is formalized via the definition of the class **#P** (read: "sharp-P" or "number-P").

**Definition 5.10.** *A counting problem* $\Psi$ *belongs to the class* **#P** *if there exists a polynomial time bounded* nondeterministic Turing machine (NTM) *such that, for each instance* I *of* $\Psi$*, the number of nondeterministic computational paths this NTM takes that lead to acceptance of this problem instance equals the number of solutions of* $\mathrm{I}(\Psi)$.

For an alternative but equivalent definition of the class **#P** in terms of *polynomially balanced relations*, we refer the reader to [140] or [32].

The hardest problems in the class **#P** are the **#P**-complete problems. We define **#P**-completeness with respect to *Turing reducibility* applied to counting problems as a special case of *function problems* (in contrast to *decision problems*).

**Definition 5.11.** *Let* $\Sigma$ *be a finite alphabet. A function* $f : \Sigma^* \to \mathbf{N}_0$ *whose range is the set of nonnegative integers* $\mathbf{N}_0$ *is said to be* polynomial time Turing reducible *to another function* $g : \Sigma^* \to \mathbf{N}_0$ *if there exists a polynomial time algorithm for* $f$ *which has access to an oracle for* $g$*, where the size of the input to* $g$ *is bounded by a polynomial in the size of the input to* $f$.

That is, Turing reducibility captures the notion that, if $g$ has a polynomial time algorithm, then so does $f$. Turing reducibility is a transitive and reflexive relation.

**Definition 5.12.** [32] *A counting problem* $\Psi$ *is* **#P***-complete if and only if*

*(i)* $\Psi \in$ **#P***, and*

*(ii)* $\Psi$ *is* hard *for this class, i.e.,* every *other counting problem in* **#P** *is Turing reducible to* $\Psi$.

Thus, if we could solve any particular **#P**-complete problem in deterministic polynomial time, then all the problems in class **#P** would be solvable in deterministic polynomial time, and the entire class **#P** would collapse to **P**.[34] For more on the class **#P**, we refer the interested reader to *Section 18* of [140] and references therein.

As one would expect, the counting versions of the standard decision **NP**-complete problems, such as SATISFIABILITY or HAMILTON CIRCUIT, are **#P**-complete [140]. What is curious, however, is that the counting versions of some tractable decision problems, such as BIPARTITE MATCHING or MONOTONE 2CNF SATISFIABILITY, are also **#P**-complete [211, 212].

If we could reduce the problem of counting the satisfying truth assignments of an instance of, say, Boolean 3CNF-SAT or PE3SAT formulae [62] to counting the fixed points of a corresponding SDS, this would establish the **#P**-completeness of the latter. However, for instance, the

[34] Strictly speaking, since **#P** is a class of *function problems* (as opposed to classes of *decision problems* and hence the formal languages associated with those decision problems, such as the familiar language classes **P**, **NP** or **PSPACE**), if any **#P** -complete problem turns out to be solvable in deterministic polynomial time, this would imply that $\mathbf{P}^{\#\mathbf{P}} = \mathbf{P}$.

reduction from ODD-PE3SAT that is used in [17] to establish the **NP**-completeness of the Fixed Point Existence (FPE) problem for SDSs would not suffice, since it does not map the satisfying assignments of an instance of ODD-PE3SAT to the fixed points of the corresponding SDS in a one-to-one fashion. That is, in order to prove the intractability of counting FPs of Boolean SDSs and SyDSs, not any *polynomial time* reduction from a known **#P**-complete problem suffices. What is required is a kind of an efficient reduction that *preserves the number of solutions*. We define this special kind of efficient reductions next:

**Definition 5.13.** *Given two decision problems* $\Pi$ *and* $\Pi'$, *a* PARSIMONIOUS REDUCTION *from* $\Pi$ *to* $\Pi'$ *is a polynomial-time transformation* g *that preserves the number of solutions; that is, if an instance* I *of* $\Pi$ *has* $n_I$ *solutions, then the corresponding instance* g(I) *of* $\Pi'$ *also has* $n_{g(I)} = n_I$ *solutions.*

In practice, one often resorts to reductions that are "almost parsimonious", in a sense that, while they do not *exactly* preserve the number of solutions, $n_I$ in the previous definition can be efficiently recovered from $n_{g(I)}$ .

**Definition 5.14.** *Given two decision problems* $\Pi$ *and* $\Pi'$, *a* WEAKLY PARSIMONIOUS REDUCTION *from* $\Pi$ *to* $\Pi'$ *is a polynomial-time transformation* g *such that, if an instance* I *of* $\Pi$ *has* $n_I$ *solutions, and the corresponding instance* g(I) *of* $\Pi'$ *has* $n_{g(I)}$ *solutions, then* $n_I$ *can be computed from* $n_{g(I)}$ *in polynomial time.*

We observe that every *parsimonious* reduction is also, trivially, *weakly parsimonious*.

All of our results on the computational complexity of counting various kinds of configurations in SDSs and SyDSs will be obtained by reducing counting problems about certain types of Boolean formulae that are *known to be* **#P**-complete to the problems about S(y)DSs. That this suffices follows from the well-known property of any problem that is *hard* for a given complexity class; for the record, we state that property in the Proposition below.

**Proposition 5.1.** [140] *Given two decision problems* $\Pi$ *and* $\Pi'$, *if the corresponding counting problem* $\#\Pi$ *is known to be* **#P***-hard, and if there exists a (weakly) parsimonious reduction from* $\Pi$ *to* $\Pi'$, *then the counting problem* $\#\Pi'$ *is* **#P***-hard, as well.*

### 5.4.1 Approximate Counting and Randomized Approximation

It has been observed that most counting problems of interest, insofar as the *exact enumeration* is concerned, are computationally intractable, i.e., **#P**-hard. Finding a feasible algorithm for exactly enumerating the solutions of a **#P**-hard problem is extremely unlikely – in particular, the existence of such an algorithm would imply the collapse of the entire *polynomial hierarchy* to the class **P**). Consequently, since the mid 1980s researchers have been seeking alternatives to solving the counting problems of interest *exactly* and *deterministically*.

Relaxing the task of the exact enumeration that is computationally feasible and works well for all problem instances has taken place along two main lines [32]. One line is to seek an *approximate* (as opposed to *exact*) solution to a counting problem. The other line is to seek *randomized* (as opposed to *deterministic*) algorithms for counting. The greatest success in that endeavor, insofar

as *positive* results are concerned, combine the two ideas; thus, over the past two decades, approximate randomized algorithms have been devised for a number of important counting problems. However, we warn the reader that there are still quite a few counting problems, however, that defy even satisfactory approximate solutions (with or without the use of randomization).

Studying *randomized algorithms* for the enumeration problem of interest is beyond the scope of this technical report. However, in the subsections to follow, in addition to our main results that address the computational complexity of *exact enumeration*, we will also establish several results on the hardness of *approximate counting*. Those results of ours will be, for the most part, straightforward corollaries to the hardness of approximation results about various static combinatorial structures, such as those about *approximately counting* the satisfying truth assignments of appropriately restricted types of Boolean formulae [157, 210]. For the sake of completeness, we include in this subsubsection the basic definitions pertaining to the hardness of approximate counting. For much more details on approximate counting, we refer the interested reader to references [32, 92, 93, 94, 101, 102, 167, 168].

Work on finding approximate solutions to **#P**-hard problems was initiated by Karp and Luby in 1985 [101]. The idea of fruitfully combining approximation and randomization to tackle hard counting problems originates in [102], where *Fully Polynomial Randomized Approximation Schemes* (FPRAS), as a more realistic model of tractability for the counting problems, were originally proposed. In a nutshell, FPRAS provide satisfactory – that is, computationally feasible – randomized approximate algorithms for counting problems that are hard to solve exactly [32]. Again, while many important and natural **#P**-hard problems do admit FPRAS algorithms, some counting problems defy a satisfactory solution even in the loose sense as provided by the FPRAS model of (probabilistic) approximability.

The notion of approximation most frequently encountered in the research literature, and the one we have in mind in each of a handful of the (non)approximability results in this technical report, is that of *relative approximation* [94, 176]. Following [157], we define what we mean by saying that one positive number provides a good relative approximation of another:

**Definition 5.15.** *Let $M$ and $M'$ be two positive integers or rational numbers, and let $\Delta > 0$. Then $M'$ approximates $M$ within $\Delta$ if and only if*

$$\frac{M'}{1+\Delta} \;\leq\; M \;\leq\; M' \cdot (1+\Delta) \tag{20}$$

Thus *relative approximability* refers to whether one can approximate the desired number (typically, of solutions to a combinatorial problem's instance) to *within a constant multiplicative factor*.

Approximate counting is closely related to *random generation* from an (almost) uniform probability distribution [168]. For concreteness, let's say we are given a cellular or network automaton with $n$ binary-valued nodes. If the probability distribution of randomly selecting a configuration is uniform, then the probability that a randomly selected configuration is, say, a fixed point equals $\frac{|\#FP|}{2^n}$, where $|\#FP|$ denotes the total number of fixed points. Based on a similar observation, which was originally in the context of approximately counting the satisfying assignments of various types of Boolean formulae, and relating that problem to the problem of approximating the *degree of belief* in a propositional statement represented by an appropriate Boolean formula, Roth defined in [157] a decision problem of the following general form:

*Let a configuration space with $2^n$ elements be given, and assume that each configuration is equally likely to be selected at random. Given an $\epsilon > 0$, how hard is the problem of approximating the probability that property $P$ holds for a randomly selected configuration to within $2^{n^{1-\epsilon}}$ ?*

The details of how one arrives at this problem formulation starting from the standard definition of relative approximability (see Definition 5.15) is given in the proof of *Theorem 4.2* in *Appendix* of [157]. The version of the above decision problem on the hardness of approximation, as applied to SDSs and other network automata of interest, can be re-phrased as follows:

*Given a cellular or network automaton with $n$ binary-valued nodes, and given an $\epsilon > 0$, how hard is the problem of approximating the number of configurations that possess property $P$ to within $2^{n^{1-\epsilon}}$ ?*

Some examples of *property $P$* above that are of our interest include *being a fixed point configuration*, *being a garden of Eden*, and similar. All our (non)approximability results in this Section, as well as in Section 6, will be stated with respect to the generic problem formulation given above, which is an immediate adaptation of a similar problem definition from [157] discussed earlier in this subsubsection. Needless to say, each of those results will have a specified *concrete* property that all configurations that are *positive instances* of the particular problem being addressed will be required to share (see examples above).

## 5.5 Counting Fixed Points of General Boolean SDSs and SyDSs

We start the presentation of our original research on the computational complexity of counting the fixed point configurations and other dynamical structures in Boolean SDSs and SyDSs with the least surprising (and least difficult) result, namely, that counting the fixed points of an *arbitrary* Boolean SDS or SyDS is, in the worst case, computationally intractable.

We shall use reductions from the known **#P**-complete problems, such as the counting version of Positive-Exactly-One-in-Three-Satisfiability, to the problems of counting FPs in certain classes of Boolean SDSs and SyDSs. These reductions will formally establish the **#P**-completeness of those counting problems about S(y)DSs. We define the variants of Satisfiability [62, 140] that we shall use in the sequel:

**Definition 5.16.** Exactly-One-in-Three-Satisfiability *(or* **E3SAT** *for short), is a version of* 3CNF-SAT *[62] such that, first, each clause in a given* 3CNF *formula contains exactly three literals, and, second, where a truth assignment is considered to satisfy the given* 3CNF *formula if and only if* exactly one of the three literals *is* true *in each clause.* Positive-Exactly-One-in-Three-Satisfiability *(***PE3SAT***) is further restricted: no clause in the* 3CNF *formula is allowed to contain a negated literal.*

Hunt *et al.* show in [85] that the counting versions of both E3SAT and PE3SAT are **#P**-complete. To establish **#P**-completeness of counting the fixed points of an SDS or SyDS, let's consider the following reduction from PE3SAT to #FP-SDS, where #FP-SDS denotes the problem of counting the fixed point configurations of an arbitrary Boolean SDS.

Let an arbitrary instance I of PE3SAT be given. We construct the corresponding instance of an SDS $\mathcal{S} = \mathcal{S}(I)$ as follows. We remark that $\mathcal{S}$ in this subsubsection will be "nearly symmetric"; we will modify our construction to a fully symmetric Boolean SDS (or SyDS) in the next subsubsection.

Assume that I has $n$ variables and $m$ clauses. The underlying graph of $\mathcal{S}$ has a distinct node for each variable $x_i$, $1 \leq i \leq n$, and for each clause $C_j$, $1 \leq j \leq m$. The node labeled $x_i$ is connected to the node labeled $C_j$ if and only if, in the Boolean formula I, variable $x_i$ appears in clause $C_j$. In addition, our graph has one additional node, labeled $y$, that is adjacent to the nodes $C_j$ for all indices $j = 1, ..., m$. Hence, each $C_j$ has exactly four neighbors, and node $y$ has $m$ neighbors.

The node update functions of our SDS $\mathcal{S}$ are as follows:

– Each node $C_j$ evaluates the logical *AND* of the current value of node $y$, the value evaluated by the PE3SAT function of the three variables $\{x_{j_1}, x_{j_2}, x_{j_3}\}$ that appear in the corresponding clause $C_j$ of I, and the current value of itself; that is, the node update function $C_j$ evaluates to 1 if and only if:

(i) *exactly* one out of the three neighboring nodes $x_{j_1}, x_{j_2}, x_{j_3}$ currently holds the value 1; and

(ii) the node $y$ currently holds the value 1; and

(iii) the current value of $C_j$ itself is 1.

– The "special" node $y$ evaluates the *AND* of its own current value and the entire set of current values held in the clause nodes $C_j$, $1 \leq j \leq m$. This will enable us to argue that the node $y$, in effect, evaluates the Boolean formula for the specified truth assignment $\{x_1, ..., x_n\}$, provided that the initial value stored in node $y$ is $y^{t=0} = 1$, and, likewise, that $C_j^{t=0} = 1$, for all $j$, $1 \leq j \leq m$.

– Each node $x_i$ evaluates the logical *AND* of itself and the current values stored in the clause nodes $C_{j(i)}$ such that, in the original formula I, variable $x_i$ appears in clause $C_{j(i)}$.

The order of the node updates is $(C_1, ..., C_m, y, x_1, ..., x_n)$.

Since $\mathcal{S}$ has $n + m + 1$ nodes, the corresponding phase space will have $2^{n+m+1}$ configurations.

We now claim that the reduction from #PE3SAT to #FP-SDS based on the above SDS construction from an instance I of PE3SAT is weakly parsimonious; it will then immediately follow that

**Theorem 5.1.** *The problem of counting the fixed points of an arbitrary Boolean SDS (and therefore also of any more general finite domain SDS) is* **#P***-complete.*

*Proof.* That #FP-SDS is a member of the class **#P** is immediate from the definition of SDS and the assumptions stated in subsection 5.3. The **#P**-hardness will follow from the **#P**-hardness of the corresponding counting version of PE3SAT, once we establish that the reduction from #PE3SAT to #FP-SDS is, indeed, (weakly) parsimonious.

First, assume we pick an initial configuration $\mathcal{C}^0$ such that its sub-configuration $(x_1^0, ..., x_n^0)$ is an unsatisfying truth assignments for the variables $(x_1, ..., x_n)$ in the corresponding instance of PE3SAT. Then, at the first step, at least one of the clause nodes will evaluate to 0, and hence the node $y$ will subsequently evaluate to 0. Once the node $y$ holds the value 0, at the next step *all* clause nodes $C_j$ will evaluate to 0, and subsequently they will force all the variable nodes $x_i$ to evaluate to 0, as well[35]. Thus, it follows that, if initially the sub-configuration $(x_1^0, ..., x_n^0)$ corresponds to a falsifying truth assignment for I, then the fixed point configuration $0^{n+m+1}$ is reached in (at most) two global transition steps.

[35] We shall assume in this and all other constructions in this technical report that each Boolean variable in any given formula I appears in at least one clause.

Let us assume now that the initial configuration $\mathcal{C}^{t=0}$ of $\mathcal{S}$ has a sub-configuration $(x_1^0, ..., x_n^0)$ that corresponds to a satisfying truth assignment to the corresponding Boolean variables in the instance I of PE3SAT and, in addition, that $y^{t=0} = 1$ and $C_j^{t=0} = 1$. Then each $C_j$ will evaluate to 1, thereby causing the node $y$ to remain evaluated to 1, as well. Since all $C_j = 1$, each node $x_i$ will keep its original value: $x_i^1 = x_i^0$. Since these values form a satisfying truth assignment, at the next step of the dynamic evolution of $\mathcal{S}$, again each $C_j$ will evaluate to 1, causing $y$ to re-evaluate to 1, and all of $x_i$ to remain the same; in other words, a fixed point configuration has been reached. Hence, if the initial configuration $\mathcal{C}^0$ has $y^0 = 1$ and $C^0 = 1^m$, and it encodes a satisfying truth assignment $(x_1^0, ..., x_n^0)$ of I, then $\mathcal{C} = \mathcal{C}^0$ already is a fixed point, given by $(C_1, ..., C_m, y, x_1, ..., x_n) = (1, ..., 1, 1, x_1^0, ..., x_n^0)$. Thus, it follows that each satisfying truth assignment $(x_1, ..., x_n)$ of I gets mapped into a *distinct* fixed point $(1, ..., 1, 1, x_1, ..., x_n)$ of the corresponding SDS $\mathcal{S} = \mathcal{S}(I)$.

Finally, it is easy to see that, if $y^0 = 0$, then $\mathcal{S}$ reaches the fixed point $0^{n+m+1}$ in a single step, and if there exists at least one index $j$ such that initially $C_j^0 = 0$, then the sink $0^{n+m+1}$ is reached in at most two steps. Since each initial configuration that encodes a falsifying truth assignment $(x_1, ..., x_n)$ to I yields the fixed point configuration $0^{n+m+1}$ in at most two steps, we conclude that there cannot be any fixed points of $\mathcal{S}$ except for $0^{n+m+1}$ and those fixed points that correspond to the satisfying assignments to I. Therefore, if I has $L$ satisfying assignments, where $0 \leq L \leq 2^n$, then the SDS $\mathcal{S}$ as constructed above will have exactly $L + 1$ fixed points.

This reduction establishes that, in general, counting fixed points of an arbitrary Boolean SDS is no easier than counting satisfying truth assignments of instances of PE3SAT formulae, and the #**P**-hardness of #FP-SDS follows, thereby establishing the claim of the theorem.

□

Similarly, by a straightforward modification of the given SDS construction, the problem of exactly enumerating FPs of general Boolean (and therefore any finite domain) SyDSs is #**P**-complete, as well:

**Corollary 5.1.** *The problem #FP-SyDS for the general Boolean and other finite domain SyDSs is* #**P**-*complete.*

We remark that the underlying graph in the construction above is *bipartite*, as there are no lateral edges among the variable nodes or among the clause nodes, and hence only even-length closed paths[36] are possible. Moreover, by the results of Hunt *et al.* in [85] on the computational complexity of counting problems for the planar graphs, the underlying graph $G_{\mathcal{S}}$ in our construction can be also made *planar*, while still preserving the hardness of the counting problem #FP-S(y)DS. Likewise, as a straightforward corollary to the complexity results by Vadhan on counting in sparse graphs and sparse Boolean formulae [210], a $O(1)$ bound on the maximum node degree for the variable nodes can be imposed, while preserving the #**P**-completeness of #FP-S(y)DS.

[36] We purposefully avoid using the word "cycle" here, even though it is the more common term in the graph theory literature, in order to avoid possible confusion between closed paths, or cycles, in the underlying graph of an SDS on one hand, and the temporal cycles characterizing this dynamical system's behavior (that is, the directed cycles in the resulting configuration space), on the other.

Therefore, when all these restrictions on $G_{\mathcal{S}}$ are imposed simultaneously, the conclusion is that the #FP-S(y)DS problem is **#P**-complete even when the underlying graph is (i) planar, (ii) bipartite, and (iii) with only one node of degree greater than $O(1)$.

We will elaborate more on the restrictions on an SDS's or SyDS's underlying graph that still maintain the hardness of counting FPs in the next Section. In the rest of this subsection, however, we will explore several other consequences of the basic SDS construction preceding the statement of Theorem 5.1.

### 5.5.1 Computational Complexity of Several Other Configuration Space Properties of General Boolean SDSs and SyDSs

The construction of an SDS from a Boolean formula in Theorem 5.1 resembles of how one constructs a Boolean circuit from a Boolean expression. That is, not only is the construction *preserving the number of solutions* but, furthermore, one can view it as a *very literal translation* from a Boolean formula into an SDS. Additionally, the resulting SDS has a rather simple configuration space: every initial configuration that encodes a satisfying truth assignment of I, and initially has $y^0 = 1$ and all $C_j^0 = 1$, is already a fixed point, whereas all initial configurations that encode falsifying truth assignments of I, as well as those that have $y^0 = 0$ or at least one $C_j^0 = 0$, reach the "sink" fixed point $0^{n+m+1}$ within two steps. These scenarios are, indeed, the only possible evolutions of $\mathcal{S}$: its configuration space has no temporal cycles whatsoever, no chains of transient configurations longer than two, and no fixed points except for $0^{n+m+1}$ and those that are of the form $(1, ..., 1, 1, x_1, ..., x_n)$, where $(x_1, ..., x_n) \in \{0, 1\}^n$ is a truth assignment that satisfies the original Boolean formula I.

We summarize the implications of the construction in Theorem 5.1 in the Lemma below:

**Lemma 5.1.** *Let an arbitrary instance* I *of* PE3SAT *be given, and let the corresponding SDS* $\mathcal{S} = \mathcal{S}(I)$ *be constructed from it as in* Theorem 5.1. *Then the following statements are equivalent:*

*(i)* I *is satisfiable, i.e., there exists a truth assignment to the variables appearing in* I *so that the underlying* PE3SAT *formula evaluates to* **true**.

*(ii) SDS* $\mathcal{S}$ *has a fixed point configuration* $\mathcal{C} \neq 0^{n+m+1}$.

*(ii') SDS* $\mathcal{S}$ *has two or more fixed points.*

*(iii)* $\mathcal{S}$ *has a configuration* $\mathcal{C}$ *such that* $\mathcal{C} \neq 0^{n+m+1}$ *and* $\mathcal{C}$ *has a 2-ancestor, that is,* $pred(pred(\mathcal{C}))$ *exists.*

*(iii')* $\mathcal{S}$ *has a configuration* $\mathcal{C} \neq 0^{n+m+1}$ *such that, for any* $k \geq 1$, $\mathcal{C}$ *has a* $k$*-ancestor.*

Proving the claims of Lemma 5.1 is rather straightforward, and will be omitted.

We recall the definition of Ambiguous-SAT, sometimes also called Double-SAT (see, e.g., [62]): *Given an instance of a Boolean CNF formula, does it have two or more solutions*, i.e., are there two or more satisfying truth assignments? One of the consequences of Lemma 5.1 is that determining whether an SDS has more than one fixed point is no easier than determining whether an instance of PE3SAT has a satisfying truth assignment. We notice the similarity between the problem of non-uniqueness of FPs in an SDS or SyDS, and the Ambiguous-SAT problem, which can also be viewed as the problem of *non-uniqueness* of satisfying truth assignments.

We recall that Ambiguous-SAT is **NP**-complete in general [62].

**Definition 5.17.** *The* AMBIGUOUS-FPE *problem: Given an arbitrary Boolean-valued SDS or SyDS* $\mathcal{S}$*, does it have more than one fixed point?*

It is now immediate from Lemma 5.1 that the following result holds:

**Corollary 5.2.** *For general Boolean and other finite domain SDSs and SyDSs, the* AMBIGUOUS-FPE *problem is* **NP***-complete.*

We conclude this subsection by establishing two more results on the hardness of *exact enumeration* in the context of arbitrary Boolean SDSs and SyDSs. Those two results will follow from the construction used to establish Theorem 5.1, and from the discussion at the beginning of this subsubsection.

The first of the two results is a direct corollary to Theorem 5.1. Namely, since the S(y)DSs constructed from the PE3SAT CNF formulae in Theorem 5.1 do not have any cycle configurations, it is immediate that $|\#TC| = 2^{m+n+1} - |\#FP|$. Since the class of function problems **#P** is closed under taking the complement, it follows that *exactly counting* all transient configurations of Boolean and other finite domain S(y)DSs is **#P**-complete.

**Theorem 5.2.** *The problem* #TC *of* exactly counting *all transient configurations of an arbitrary Boolean (or any other finite domain) SDS or SyDS is* **#P***-complete.*

We next focus on the complexity of enumerating only those transient configurations that are gardens of Eden. The sequel of lemmata that follows has the purpose of establishing two facts. One, SDSs and SyDSs constructed as in Theorem 5.1 have *many* GE configurations. Consequently, approximating of the number of those gardens of Eden, denoted $|\#GE|$, to within a constant multiplicative factor is trivial for such S(y)DSs. (This fact, however, by no means implies the easiness of the problem of approximate enumeration of GEs for Boolean S(y)DSs in general.) Two, and more importantly for our purposes, determining $|\#GE|$ *exactly* is intractable.

**Lemma 5.2.** *Let an SDS* $\mathcal{S}$ *be constructed from an instance* I *of* PE3SAT *as in Theorem* 5.1*. Assume that* I *contains* $n$ *Boolean variables and* $m$ *clauses, where each clause contains three unnegated variables. Let* $\mathcal{C} = (C, y, x)$ *denote a generic global configuration, where* $C \in \{0,1\}^m$*,* $y \in \{0,1\}$ *and* $x \in \{0,1\}^n$*. Then*

*(i) Every configuration* $\mathcal{C}$ *of the form* $(C, y, x) = (C, 1, x)$ *with* $C \in \{0,1\}^m - \{1^m\}$ *and any* $x \in \{0,1\}^n$ *is a GE.*

*(ii) Every configuration* $\mathcal{C}$ *of the form* $(C, y, x) = (1^m, 0, x)$ *with* $x \in \{0,1\}^n$ *is a GE.*

*Proof.* Recall that the node update ordering is $\Pi = (C_1, ..., C_m, y, x_1, ..., x_n)$.

Assume there exists a configuration $\mathcal{C} = (C, y, x) = (C, 1, x)$ with $C \neq 1^m$ that actually is not a GE. Then this configuration must have a predecessor. Let $\mathcal{C}' = pred(\mathcal{C})$. Since the node $y$ updates according to Boolean *AND* that includes its own old value, $y(\mathcal{C}') = 1$. Assume there exists $j_\star \in \{1, ..., m\}$ such that $C_{j_\star}(\mathcal{C}') = 0$. Then, at the next time step, the node $C_{j_\star}$ has to reevaluate to 0: $C_{j_\star}(\mathcal{C}) = 0$. Consequently, since $y^{t+1} \leftarrow y^t \cdot \Pi_{j=1}^m C_j^{t+1}$, it follows that $y(\mathcal{C}) = 0$, a contradiction. Therefore, it must be that $\forall j \in \{1, ..., m\} : C_j(\mathcal{C}') = 1$, i.e., $\mathcal{C}$' is of the form $(C', y', x') = (1^m, 1, x')$, for some $x' \in \{0,1\}^n$.

Let $TRUE \subset \{0,1\}^n$ denote the set of those Boolean $n$-vectors that, if used as a truth assignment to $x = (x_1, ..., x_n)$, make the given PE3SAT formula I evaluate to *true* : I($x$) $= 1$. Let the set $FALSE$ be defined analogously. For any of the $2^n$ $(m+n+1)$-vectors of the form $(C, y, x) = (1^m, 1, x)$, there are two exhaustive and mutually exclusive possibilities: either $x \in TRUE$, or else $x \in FALSE$. If $x \in TRUE$, then $\mathcal{C}^\star = (1^m, 1, x_{sat})$ is a fixed point; that is, $F(\mathcal{C}^\star) = \mathcal{C}^\star \neq \mathcal{C}$, so this $\mathcal{C}^\star$ cannot be a predecessor of $\mathcal{C}$. On the other hand, if $x \in FALSE$, then there exists $j_\star$ such that the clause $C_{j_\star}$ of formula I is falsified for this truth assignment $x = (x_1, ..., x_n)$. Hence, the corresponding clause node in the SDS $\mathcal{S}$ constructed from I will evaluate to zero at the next time step, thereby coercing the node $y$ (that gets its turn only once all the nodes $C_j$ have already updated their respective states) to update to zero, as well. This, however, contradicts our assumption that $y(\mathcal{C}) = 1$. Hence, a configuration of the form $\mathcal{C} = (C, y, x) = (C, 1, x)$ cannot possibly have a predecessor, i.e., all such configurations are gardens of Eden.

To show that any configuration of the form $\mathcal{C} = (C, y, x) = (1^m, 0, x)$ also must be a GE, we observe that, since each node $C_j$ updates according to Boolean $AND$ on several inputs including this node's own current state, a candidate predecessor $\mathcal{C}$' of $\mathcal{C}$ must satisfy $\mathcal{C}' = (1^m, y, x')$ for some Boolean $n$-vector $x'$. If $y(\mathcal{C}') = 1$, then, since also $C(\mathcal{C}) = 1^m$, at the next time step the node $y$ would update to 1 again; hence, $F(\mathcal{C}')$ cannot be of the form $(C, y, x) = (1^m, 0, x)$. But if $y(\mathcal{C}') = 0$, that would force $C(\mathcal{C}) \leftarrow 0^m$, thereby violating the assumption that $\mathcal{C} = (C, y, x)$ is of the form $(1^m, y, x)$. Hence, each configuration of the form $(C, y, x) = (1^m, 0, x)$ is a GE. □

An immediate consequence of the above result is that the GE configurations are, indeed, abundant in those SDSs that are constructed from the PE3SAT formulae as in Theorem 5.1:

**Lemma 5.3.** *Let an SDS $\mathcal{S}$ be constructed from an instance of a* PE3SAT *formula with $n$ variables and $m$ clauses, as in Theorem* 5.1. *Then $\mathcal{S}$ has a number of garden of Eden configurations that is* exponential *in both $n$ and $m$.*

*Proof.* There are $(2^m - 1) \cdot 2^n$ configurations of $\mathcal{S}$ that are of the form $(C, y, x) = (C, 1, x)$, where $x \in \{0,1\}^n$ is arbitrary, and $C \in \{0,1\}^m - \{1^m\}$. By Lemma 5.2, each of these $\Theta(2^{m+n})$ configurations is a GE. □

Since there are also $2^n$ GE configurations that are of the form $(C, y, x) = (1^m, 0, x)$, the next result is immediate.

**Corollary 5.3.** *Let an SDS $\mathcal{S}$ be constructed from an instance of a* PE3SAT *formula with $n$ variables and $m$ clauses, as in Theorem* 5.1. *Then at least a half of all global configurations in the phase space of $\mathcal{S}$ are gardens of Eden.*

As already mentioned, we would like to argue that the problem #GE-SDS of *exactly* counting the gardens of Eden in an SDS constructed from an instance of PE3SAT is, in general, intractable. To that end, we establish a one-to-one correspondence between the *falsifying* truth assignments of a PE3SAT formula, and the GE configurations in the corresponding SDS of the general form $(C, y, x) = (1^m, 1, x_{false})$, where $x_{false} \in FALSE$. It will follow from that

correspondence that, in order to determine $|\#GE|$ exactly, we need to be able to exactly enumerate all falsifying truth assignments to the underlying PE3SAT Boolean formula; the latter computational task, however, is known to be **#P**-complete.

**Lemma 5.4.** *Let an SDS $\mathcal{S}$ be defined as in the construction of Theorem* 5.1. *Then* every *configuration that is of the form $(C, y, x) = (1^m, 1, x_{false})$, with $x_{false} \in \{0,1\}^n$ corresponding to a* falsifying *truth assignment of the underlying* PE3CNF *Boolean formula, is a garden of Eden of $\mathcal{S}$.*

*Proof.* Let $\mathcal{C} = (C, y, x) = (1^m, 1, x_{false})$ and let's assume there exists a configuration $\mathcal{C}$' such that $pred(\mathcal{C}) = \mathcal{C}'$. Since the node $y$ updates according to Boolean $AND$ that includes its own current value, $y(\mathcal{C}') = 1$ must hold. Similarly, for all $j \in \{1, ..., m\}$, $C_j(\mathcal{C}') = 1$ must also hold. Hence, the assumed predecessor configuration itself must be of the form $\mathcal{C}' = (C', y', x') = (1^m, 1, x')$, for some $x' \in \{0,1\}^n$. Now, just like in the proof of Lemma 5.2, there are only two possibilities: either $x' \in TRUE$, or else $x' \in FALSE$. Either possibility clearly leads to a contradiction. Hence, $\mathcal{C}$ cannot have a predecessor, i.e., it must be a garden of Eden. □

The stage has now been set for the second important result on the complexity of *the exact enumeration* of GEs and TCs in Boolean and other finite domain SDSs and SyDSs:

**Theorem 5.3.** *The following two problems are* **#P***-complete: Given an arbitrary Boolean or other finite domain SDS or SyDS $\mathcal{S}$' with n nodes, an integer k such that $1 \leq k < n$, a subset of nodes $W \subset V$ such that $|W| = k$, and an arbitrary k-vector $b = (b_1, ..., b_k) \in \mathcal{D}^k$, how many*

- *garden of Eden configurations, or*
- *arbitrary transient configurations*

*$\mathcal{C}'$ such that $W(\mathcal{C}') = b$ does S(y)DS $\mathcal{S}$' have?*

We shall show in subsection 5.6 that the problem #GE remains intractable when no restrictions on the allowable states of any of the SDS's or SyDS's nodes are imposed – that is, when $k = 0$ and $W = \emptyset$ in Theorem 5.3 above. That is, the exact analog of Theorems 5.1 and 5.2 holds for the problem of enumerating the *garden of Eden* configurations, as well. Moreover, the intractability of exactly enumerating FPs, GEs and all TCs of a Boolean S(y)DS will be established even when the underlying graphs are required to be simultaneously planar, bipartite and with $|E| = O(|V|)$.

## 5.6 Some Properties of Boolean SDSs and SyDSs Defined on Planar Bipartite Graphs

We show in this subsection that the problems of counting FPs and GEs in Boolean SDSs and SyDSs remain intractable, when these discrete dynamical systems are defined over very restricted underlying graphs – in particular, when these underlying graphs are required to be *both planar and bipartite*. While the hardness of these counting problems can be obtained from Theorem 5.1 by imposing appropriate restrictions on the instances of PE3SAT formulae and then using the

work of other researchers on the complexity of counting in planar, bipartite and/or sparse graphs (see discussion at the end of subsubsection 5.5), we prefer to prove the desired properties in an alternative manner, that is both simpler and independent of the prior work. To that end, we now focus on the *star graphs* [188]. We observe that just about any interesting graph-theoretic problem is trivial to solve on a star graph. Hence, the computational hardness results on various problems about general planar, bipartite and/or sparse graphs, such as those in [85, 210], cannot be conveniently translated into the context of computational complexity of determining various configuration space properties of SDSs and SyDSs that are defined on the star graphs.

*Star networks* are a very simple and broadly studied class of communication topologies among computational agents, processes or processors. The star networks capture the simplest possible way of coupling multiple entities such that one designated entity has the central, leader or "master" role, whereas the remaining entities are the followers or "slaves". In particular, in a star graph, the central node is connected to all other nodes, yet no two peripheral (non-central) nodes are connected to each other. If the edges in such a graph correspond to communication links, the immediate implication is that no pair of peripheral nodes can communicate directly: all the communication among different nodes has to go through the central node.

We will first show that counting FPs and GEs of a Boolean SDS or SyDS defined on a star graph is, in general, **#P**-complete – as long as the local update rule of the central node is *not given as a truth table*. Moreover, we will then prove that counting these configurations in the star graphs remains intractable even when the central node's update rule is given as a *monotone Boolean formula* of a modest size. This is in stark contrast to the proven tractability of both resolving the fixed point existence (the FPE problem), and actually finding a fixed point, in every Boolean SDS or SyDS all of whose nodes update according to the *monotone Boolean functions* [17].

### 5.6.1 Counting FPs and GEs of SDSs and SyDSs Defined on Star Graphs

We now consider SDSs and SyDSs with arbitrary Boolean update rules, but such that the underlying graphs of these S(y)DSs are required to be the star graphs.

If the node update functions are encoded as the truth tables, then for general Boolean S(y)DSs defined on the star graphs, we argue that essentially any computational problem about the configuration space properties is solvable in time polynomial in the size of the S(y)DS's description. However, if we allow the node update rules either to be considered "black boxes" (i.e., oracles), or else be given as sufficiently succinct *Boolean formulae,* then the counting problems of interest will be shown to become intractable.

Let us first consider SDSs and SyDSs whose node update functions are given as the truth tables.

Assume node $y$ is the center of the star, and that it is adjacent to $n$ periphery nodes $x_1, ..., x_n$. Since each node $x_i$ has only one neighbor and therefore depends on only two inputs (i.e., the current values of $y$ and itself), the table size for each $x_i$ is only $O(1)$, and any local update rule computation via a table search will only take $O(1)$ time. However, the central node $y$ has $n$ neighbors; therefore, assuming an arbitrary Boolean function $f_y = g(x_1, ..., x_n, y)$ as the node update rule of $y$, the truth table for $f_y$ will have $\Theta(2^n)$ rows. Hence, for an S(y)DS defined on a star graph, and assuming an arbitrary Boolean function at the central node, the size of the

overall description is $\Theta(2^n)$.

When the node update functions are given as the truth-tables, then, for arbitrary Boolean SDSs and SyDSs defined on the star graphs, counting fixed points and gardens of Eden can be easily accomplished in time polynomial in the size of the S(y)DS representation. This is due to the exponential (in the number of nodes) amount of storage that the central node needs for representing its update rule as a truth table. It is fairly straightforward to convince oneself that, in that case, virtually *every* configuration space property of interest, including the problems of exactly determining $|\#FP|$, $|\#GE|$ or $|\#TC|$, can be done in a number of steps that is *polynomial* in the size of the S(y)DS's encoding, $\Theta(2^n)$.

These observations about arbitrary Boolean SDSs and SyDSs defined on the star graphs that include the truth tables for each node as a part of their description can now be contrasted with the scenario where the truth tables are *not* a part of the description of an SDS or SyDS. In this latter case, the S(y)DSs are represented much more succinctly, and it turns out that (unless $\mathbf{P} = \mathbf{P}^{\#\mathbf{P}}$) the fundamental counting problems for such S(y)DSs become intractable. Throughout the rest of this subsubsection, therefore, we assume the size of the S(y)DS's encoding is $O(n)$, where $n$ is the number of nodes, i.e., that the central node does not need to store the truth table (and then compute by searching that table).

Clearly, if the central node has its update rule given as a "black box", i.e., an oracle that does not contribute to the overall (size of) the S(y)DS's description, the overall size of the automaton's encoding is indeed going to be $O(n)$, as desired. We will first establish the hardness of counting results under that assumption; subsequently, we will show that counting remains intractable in the worst case if the central node's rule is given as a Boolean formula, where the size of the formula is considered a part of the overall SDS's or SyDS's encoding.

Let $f_y$ denote the update rule of the central node. We assume that $f_y$ is an entirely arbitrary Boolean function on $n+1$ inputs. We also assume (for now) that the central node, $y$, has an oracle for evaluating $f_y$ in a single unit of time.

**Theorem 5.4.** *When the truth tables are* not *a part of the S(y)DS description, the following counting problems about Boolean SDSs and SyDSs: given an S(y)DS $\mathcal{S}$,*

*(i) the problem* #FP *of determining the number of fixed point configurations of $\mathcal{S}$;*

*(ii) given an arbitrary configuration $\mathcal{C}$, the problem of determining the number of predecessors of $\mathcal{C}$ (abbreviated as #PRED);*

*(iii) the problem* #TC *of determining the number of all transient configurations of $\mathcal{S}$; and*

*(iv) the problem* #GE *of determining the number of garden of Eden configurations of $\mathcal{S}$*

*are all $\#\mathbf{P}$-complete, even when the underlying graph of $\mathcal{S}$ is required to be a star.*

*Proof.* We construct an SyDS $\mathcal{S}$' that, in essence, emulates an *unbounded fan-in* Boolean circuit that evaluates an arbitrary Boolean-valued function $f$ of $n$ Boolean variables.

– The underlying graph $G_{\mathcal{S}} = G(V, E)$ has $n+1$ nodes, that is, one node for each input variable $x_i$, and one special node, $y$, that will, under appropriate circumstances, store the value of $f(x_1, ..., x_n)$;

– Node $y$ is adjacent to all nodes $x_i$ (that is, the graph $G_{\mathcal{S}}$ is a star), and it updates its state according to the rule $y^{t+1} = y^t \cdot f(x_1^t, ..., x_n^t)$;

– Each node $x_i$ is adjacent only to the central node $y$, and updates its value according to $x_i^{t+1} = x_i^t \cdot y^t$.

If $y^0 = 0$, then the node $y$ will remain holding the value 0 after all future updates, regardless of whether the truth assignment $(x_1^0, ..., x_n^0)$ satisfies the Boolean formula $f$ or not. Since each $x_i^1 = x_i^0 \cdot y^0 = x_i^0 \cdot 0$, it follows that all $x_i$ values will be updated to 0, regardless of their initial values. Hence, if $y^0 = 0$, then $\mathcal{S}$' collapses to the "sink" $0^{n+1}$ in a single step of its evolution.

On the other hand, if $y^0 = 1$, then $y^1 = y^0 \cdot f(x_1^0, ..., x_n^0) = f(x_1^0, ..., x_n^0)$, i.e., the node $y$ will update to 1 iff $(x_1^0, ..., x_n^0)$ is a satisfying truth assignment for $f$. Insofar as the nodes $x_i$ are concerned, they will evaluate to their previous values at time $t = 1$, whereas at time $t = 2$ they will reevaluate to the same value iff $y$ has evaluated to 1 at $t = 1$, and to 0, otherwise. Thus, after two steps, the configuration reached will either be $0^{n+1}$, if the initial choice of $(x_1^0, ..., x_n^0)$ corresponds to a falsifying truth assignment of $f$, or it will be of the form $(y^2, x_1^2, ..., x_n^2) = (1, x_1^0, ..., x_n^0)$, if $(x_1^0, ..., x_n^0)$ is a satisfying truth assignment of $f$. Either way, $\mathcal{S}$' will stay at the configuration it has reached after two transitions.

To summarize, the configuration space of $\mathcal{S}$' has no cycles or long transients, and its fixed points are precisely $0^{n+1}$ and those configurations that are of the form $(y^0 = 1, x_1^0, ..., x_n^0)$, where $(x_1^0, ..., x_n^0)$ is a satisfying truth assignment of the corresponding Boolean function $f$.

It is almost immediate that, if the node values are updated sequentially instead of synchronously in parallel, i.e., if we consider an SDS corresponding to the SyDS described above, our analysis of the fixed points remains valid. Namely, the fixed points are invariant with respect to the choice of a node update ordering. For the sake of definiteness, but also in order to be able to show the hardness of counting configurations other than fixed points for SDSs defined on the star graphs, we convert the above SyDS $\mathcal{S}$' into a concrete SDS $\mathcal{S}$ by specifying the node update ordering $\Pi = (y, x_1, ...., x_n)$. Thus the node update functions of $\mathcal{S}$ are

$y^{t+1} \leftarrow y^t \cdot f(x_0^t, ..., x_n^t)$ ;
$for \;\; i = 1, ..., n$
    $x_i^{t+1} \leftarrow x_i^t \cdot y^{t+1}$ ;
$end \;\; for$

We omit a detailed analysis of this SDS's behavior (the analysis is rather similar to that for the corresponding SyDS), and summarize what the configuration space of $\mathcal{S}$ looks like. The fixed points of $\mathcal{S}$ are precisely the "sink" $0^{n+1}$ and the configurations of the form $(y^0 = 1, x_1^0, ..., x_n^0)$ where $f(x_1^0, ..., x_n^0) = 1$. Thus $\mathcal{S}$ has $T + 1$ fixed points *if and only if* the corresponding Boolean function $f$ has $T$ satisfying truth assignments. We also observe that the fixed point configurations of the form $(1, x_1^0, ..., x_n^0)$ have no incoming transients, simply because all transients lead to the sink $0^{n+1}$. Furthermore, the configuration space of $\mathcal{S}$ has no cycles and no transient chains longer than one.

Since the convergence from any transient configuration to the sink $0^{n+1}$ takes exactly one step, every TC is necessarily also a garden of Eden. We recall that every configuration $\mathcal{C}$ with $y(\mathcal{C}) = 0$ and different from the sink $0^{n+1}$ is a transient state. Thus, if the function $f$ has $T$ solutions, then the corresponding SDS $\mathcal{S}$ has exactly $|\#FP| = T + 1$ fixed points and exactly $|\#TC| = |\#GE| = (2^n - 1) + (2^n - T) = 2^{n+1} - T - 1$ transient configurations, each of which is also a garden of Eden.

We also observe that, since the transition from *every* transient configuration leads to the sink FP $0^{n+1}$, this fixed point configuration has the number of predecessors equal to $|\#TC| + 1 =$

$|\#GE| + 1 = 2^{n+1} - T$.

We remark that the construction described above establishes only the hardness part for the counting problems (i) – (iv) in the theorem. It is easy to see, however, that all these counting problems belong to the class **#P**, since the corresponding decision problems can be readily seen to be in the class **NP**. □

While our primary goal in this subsection is to establish the hardness of (exactly) enumerating the fixed point and the garden of Eden configurations for the Boolean and other finite domain SDSs and SyDSs whose underlying graphs are severely restricted, the given constructions also have some implications for the *decision problems* about certain S(y)DS configuration space properties of interest. We next list a few of those properties. The hardness of the configuration space properties below follows directly from the hardness of appropriate satisfiability problems for Boolean functions, and the constructions in the proof of Theorem 5.4.

**Lemma 5.5.** *When the truth tables are* not *a part of an SDS's or SyDS's description, the following decision problems about Boolean SDSs and SyDSs are* **NP***-complete, even when the underlying graph is required to be a star:*

*(i) The* AMBIGUOUS-FPE PROBLEM: *Given a Boolean S(y)DS $\mathcal{S}$', does it have more than one fixed point?*

*(ii) Given a Boolean S(y)DS $\mathcal{S}$', a subset of nodes $W \subset V$ such that $|W| = k$ with $k \geq 1$, and an arbitrary Boolean vector $b = (b_1, ..., b_k)$, does $\mathcal{S}$' have a fixed point configuration $\mathcal{C}'$ such that $W(\mathcal{C}') = b$ ?*

*(iii) Given a Boolean S(y)DS $\mathcal{S}$', a subset of nodes $W \subset V$ such that $|W| = k$ where $k \geq 1$, a Boolean vector $b = (b_1, ..., b_k)$, and a configuration $\mathcal{C}$, does $\mathcal{C}$ have any predecessors that satisfy $W(pred(\mathcal{C})) = b$ ?*

We recall that the TAUTOLOGY PROBLEM for Boolean functions ("Given an arbitrary Boolean function $f$, does $f$ evaluate to *true* for *all* truth assignments to its variables ?") is a paradigmatic **coNP**-complete problem. The SDS $\mathcal{S}$ in the proof of Theorem 5.4 will have no more than $2^n - 1$ transient configurations or gardens of Eden (out of $2^{n+1}$ configurations in total) *if and only if* $\mathcal{S}$ *does not have* any TC or GE configurations with $y = 1$ *if and only if* $f$ is a tautology. Hence, another corollary to our construction in the proof of Theorem 5.4 is that determining whether an SDS defined over a star graph will have any *nontrivial* general transient or garden of Eden configurations is **NP**-hard (where, in our example, *nontrivial configurations* would be those configurations $\mathcal{C}$ such that $y(\mathcal{C}) = 1$). One can view this corollary as an extension of the already known result on **NP**-completeness of the GARDEN OF EDEN EXISTENCE problem for Boolean and finite range SDSs [17]. Likewise, complexity results about the existence of GEs and TCs analogous to the claims in parts (ii) and (iii) of Lemma 5.5 also hold:

**Corollary 5.4.** *When the truth tables are* not *a part of an S(y)DS's description, the following decision problems are* **NP***-complete, even when the underlying graph is required to be a star: given a Boolean S(y)DS $\mathcal{S}$', a subset of nodes $W \subset V$ such that $|W| = k$, and an arbitrary Boolean vector $b = (b_1, ..., b_k)$, does $\mathcal{S}$' have (i) a GE configuration, or (ii) a transient configuration $\mathcal{C}'$ such that $W(\mathcal{C}') = b$ ?*

### 5.6.2 Counting FPs and GEs of *Monotone* S(y)DSs Defined on Star Graphs

All results in the previous subsubsection have been shown under the unrealistic assumption that the central node updates its state by using an oracle. Moreover, all the hardness of counting results in this Section thus far have been shown for the SDSs and SyDSs for which the nontrivial corresponding decision problems, such as Ambiguous-FPE, are intractable in general.

We shall now drop the oracle assumption, and assume that the central node is given its update rule $f$ as a Boolean formula that is a part of the S(y)DS's description. Furthermore, we will also choose $f$ to be from a class of Boolean formulae for which the corresponding satisfiability problem is tractable – thereby assuring that the related decision problems about the resulting SDS or SyDS, such as FPE and Ambiguous-FPE, are also tractable.

The purpose of this exercise is twofold. One, we show that counting configurations of S(y)DSs over the star graphs is, indeed, intractable – as long as the central node's update rule is encoded *reasonably succinctly* [194]. Two, just like what has been known for the Boolean formulae and many other combinatorial problem domains, we show in the context of discrete dynamical systems of our interest that there is an intrinsic aspect of computational hardness that is *peculiar to the counting problems,* and that does not have an analog among the corresponding decision problems.

We now re-state Theorem 5.4, only without the artificial "black box" assumption about the nodes' update rules:

**Theorem 5.5.** Exactly enumerating *each of the following types of configurations:*
*(i) all fixed points;*
*(ii) all predecessors of an arbitrary configuration;*
*(iii) all transient configurations; and*
*(iv) precisely those transient configurations that are gardens of Eden*
*of Boolean and other finite domain S(y)DSs is, in general,* **#P***-complete, even when the underlying graph is restricted to a* star graph, *and each node updates according to a* monotone *Boolean-valued function, where the local update rules are considered a part of the S(y)DS description, and are given as negation-free Boolean* 2CNF *formulae.*

*Proof.* Consider the class of Mon-2CNF Boolean formulae: each clause has exactly two literals, and no negated variables are allowed. These formulae are a classical example where the problem of existence of a solution (i.e., a satisfying truth assignment) is trivial, yet enumerating all satisfying assignments is, in general, **#P**-complete [212]. Moreover, it has been shown much more recently that the counting problem #Mon-2CNF remains **#P**-complete even if no variable appears in more than four clauses [210]. In particular, if no variable appears in more than $O(1)$ clauses, it immediately follows that, given such a Mon-2CNF formula that contains $n$ Boolean variables and some number of 2CNF clauses with those variables, the length of the entire formula is $O(n)$.

Now consider an SyDS or SDS as constructed in the proof of Theorem 5.4, except that, instead of the central node's update rule being treated as an oracle, this node, $y$, is given its update rule $f_y$ as a Mon-2CNF formula with each variable $x_i$ appearing in only $O(1)$ clauses (say, at most four). Similarly, for the sake of consistency, assume that the update rules for the peripheral nodes $x_i$ (which are just the Boolean $AND$ functions of two variables) are also given as formulae, and considered a part of the problem instance's description. Now the size of such an S(y)DS's encoding is at most $|E|+|f_{max}|\cdot|V|$, where $|f_{max}|$ stands for the maximum size of an update rule

formula. Under the stated assumptions, and since $|E| = O(|V|)$ holds for the star graphs, in our case a straightforward substitution shows that this encoding is of a size that is at most $O(n^2)$. In fact, a more careful analysis shows that the size is only $\Theta(n)$. Consequently, incorporating the encodings of the local update rules into the description of such an S(y)DS does not affect much the size of that dynamical system's overall description. The rest of the argument is identical to the proof of Theorem 5.4. □

We summarize the main results of this subsection in the following

**Corollary 5.5.** *Exactly enumerating each of the following types of configurations: (i) the fixed points, (ii) the gardens of Eden, (iii) the predecessors, and (iv) the transient configurations of finite domain SDSs and SyDSs is, in the worst case, computationally intractable, even when all of the following restrictions on the underlying graph and the local update rules* simultaneously *hold:*

- *the underlying graph is* planar, bipartite, *and with* $|E| \leq |V|$*;*
- *all local update rules are* monotone Boolean functions;
- *these update rules are considered a part of the SDS's or SyDS's description, and are given as (monotone) Boolean formulae; and*
- *SDS or SyDS uses only two different update rules from the given class of functions.*

## 5.7 Counting Various Configurations of Symmetric Boolean SDSs and SyDSs

The hardness results for symmetric Boolean SDSs and SyDSs will be based on an appropriate reduction from the PE2-IN-3SAT problem. We define PE2-IN-3SAT similarly to how we defined PE3SAT, only this time we require each clause to have *exactly two* true variables (rather than *exactly one* as was the case in PE3SAT). We observe that, since PE3SAT is **NP**-complete, so is PE2-IN-3SAT, and moreover the **#P**-completeness of the counting version of the former, let's denote it #PE3SAT, also implies the **#P**-completeness of the counting version of the latter, #PE2-IN-3SAT.

Let an instance I of PE2-IN-3SAT be given. Assume that there are $n$ Boolean variables, denoted $x_1, ..., x_n$, and $m$ clauses, $C_1, ..., C_m$, in I. We recall that each clause $C_j$ contains *exactly* three unnegated variables, $x_{j_1}, x_{j_2}, x_{j_3}$. An instance I is a *positive* or *satisfying* instance of PE2-IN-3SAT if and only if there exists a truth assignment to $x_1, ..., x_n$ such that *exactly* two variables in each clause are true.

We now prove that counting the fixed point configurations of a symmetric Boolean SyDS or SDS is **#P**-complete. We recall that fixed points are invariant under the node update ordering; that is, regardless of whether the nodes update synchronously in parallel, or sequentially according to an arbitrary ordering $\Pi$, the fixed points of the underlying dynamical system as specified by its graph and the local node update functions remain the same (see [132] for a proof).

**Theorem 5.6.** *The problem of counting fixed points of a* symmetric *Boolean Synchronous Dynamical System, abbreviated as* #FP-SYM-SYDS*, is* **#P***-complete.*

*Proof.* To show #**P**-hardness, we reduce the problem of counting the satisfying truth assignments of an instance of PE2-IN-3SAT to counting the fixed points of a symmetric Boolean SyDS. We construct an SyDS, $\mathcal{S}$, from an instance of PE2-IN-3SAT as follows. (We recall that such a Boolean formula is assumed to have $n$ variables, labeled $x_1, ..., x_n$, and $m$ clauses, labeled $C_1, ..., C_m$.) We let the underlying graph of $\mathcal{S}$ have $m + n + 1$ vertices: one for each variable, one for each clause, and one additional vertex, denoted by $y$. Next, we define the edges of the underlying SyDS graph. Each vertex node $x_i$ is adjacent to those and only those clause nodes $C_{j(i)}$ such that the corresponding variable $x_i$ appears in the corresponding clause $C_{j(i)}$ of formula I. Each clause node $C_j$ is adjacent to all other clause nodes $C_k$ (for all $k$, $1 \leq k \leq m$, $k \neq j$), to the special node $y$, and to the three nodes $x_{j_1}, x_{j_2}, x_{j_3}$ corresponding to the Boolean variables that appear in the clause $C_j$ in the formula. Finally, by symmetry, the node $y$ is adjacent to all the clause nodes $C_j$, $1 \leq j \leq m$.

We define the node update functions as follows:

$x_i^{t+1} = x_i^t \wedge (\wedge_{j(i)} C_{j(i)}^t)$;

$C_j^{t+1} = \text{ALL-BUT-ONE } \{x_{j_1}^t, x_{j_2}^t, x_{j_3}^t, C_1^t, ..., C_m^t, y^t\}$;

$y^{t+1} = y^t \wedge (\wedge_{j=1}^m C_j^t)$,

where the Boolean function ALL-BUT-ONE $\{z_1, ..., z_q\} = 1$ if and only if *exactly* one of its inputs $z_l$ is 0, and all the rest are 1$s$ (for $1 \leq l \leq q$).

The underlying graph of the SyDS we have just described looks as in *Figure* 5 below:

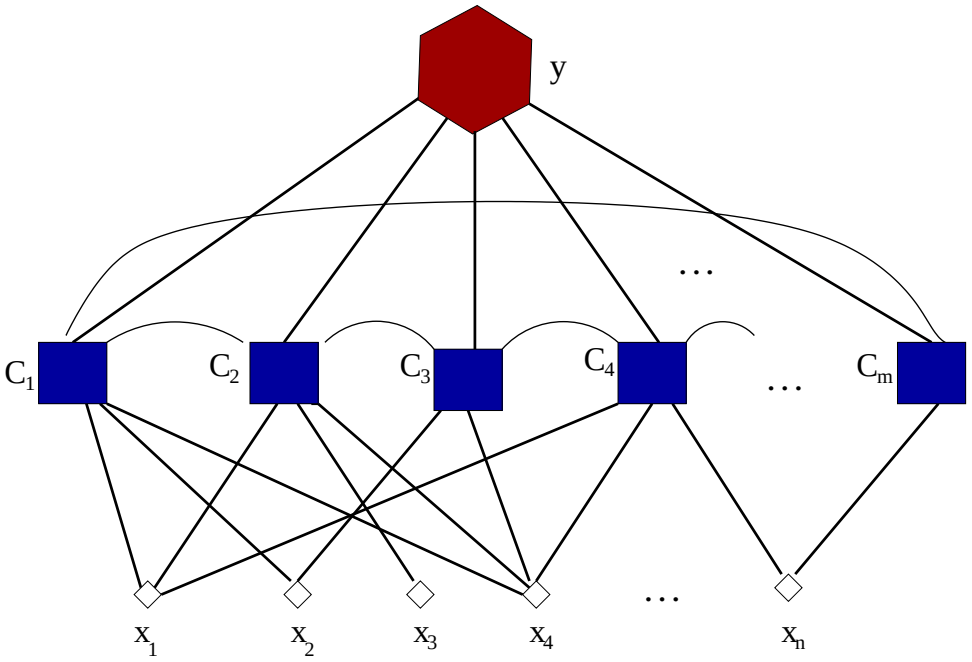



Figure 5: The graph of a symmetric Boolean SyDS in the construction of Theorem 5.6.

We now claim that the constructed synchronous dynamical system has $|T| + 2$ fixed points if and only if the corresponding instance of PE2-IN-3SAT has $|T|$ satisfying truth assignments.

To prove the claim, we will carefully analyze all possible scenarios of the dynamic behavior of $\mathcal{S}$, based on its initial configuration. We shall adopt the notation that $x$ and $C$ without any subscripts denote Boolean $n$- and $m$-vectors, respectively, the former being a shorthand for $(x_1, ..., x_n)$ and the latter for $(C_1, ..., C_m)$. Hence, using this abridged notation, we can now write arbitrary configurations of $\mathcal{S}$ as ordered triples $(x, C, y)$.

We start with a simple observation that, since the node update functions at the variable nodes $x_i$, as well as the special node $y$, are conjunctions of inputs that include the old value of the node in question itself, once any $x_i$ or the node $y$ evaluates to 0, it remains 0 thereafter. We split the analysis of the dynamic behavior of $\mathcal{S}$ into two parts.

**Case 1:** $y^0 = 0$. First consider the case when, initially, $x_i^0 = 1$ for all $i$, $1 \leq i \leq n$, and also $C_j^0 = 1$, for all $j$, $1 \leq j \leq m$. At time $t = 1$, all the variable nodes $x_i$ will remain in the state 1. Also, since each clause node update function $C_j$ at time $t = 1$ will have all inputs equal to 1 except for a single one (namely, the input $y^0 = 0$), $C_j^1 = 1$. On the other hand, clearly $y^t = 0$ for $t = 1, 2, ...$, irrespective of the remaining inputs $C_j^{t-1}$. Hence, we conclude that the configuration $(x, C, y) = (1^n, 1^m, 0)$ is a fixed point of $\mathcal{S}$. Notice, however, that this configuration does not correspond to a satisfying truth assignment of the corresponding instance $I$ of PE2-IN-3SAT, since, if all $x_i = 1$, then no clause $C_j$ of I will be satisfied, as each clause requires exactly two inputs equal to 1 and one input equal to 0.

Now consider a starting configuration where there exists an index $j_\star$ such that $C_{j_\star}^0 = 0$. Then, at time $t = 1$, all the clause nodes $C_j$ will have at least two 0 inputs (namely, $y^0$ and $C_{j_\star}^0$), and, since they evaluate the ALL-BUT-ONE function of their inputs, they will all evaluate to 0: $C_j^1 = 0$, for all $j$, $1 \leq j \leq m$. Hence, at the next step, $x_i^2 = x_i^1 \wedge (\wedge_{j(i)} C_{j(i)}^1) = 0$ for all $i$, $1 \leq i \leq n$, and it is easy to see that, for $t \geq 2$, $(x^t, C^t, 0) = 0^{n+m+1}$, i.e., the fixed point $0^{n+m+1}$ is swiftly reached – in at most two transition steps. Similar analysis, and the same conclusion, hold if we assume that there is at time $t = 0$ at least one index $i_\star$ such that $x_{i_\star}^0 = 0$. We observe that, just like the fixed point $(1^n, 1^m, 0)$, the fixed point $(0^n, 0^m, 0) = 0^{n+m+1}$ does not correspond to a satisfying truth assignment $(x_1, ..., x_n)$ of formula I. This completes the analysis of all possible scenarios when $y^0 = 0$.

**Case 2:** $y^0 = 1$. There are two sub-cases to consider. The first sub-case is when there exists an index $j_\star$ such that $C_{j_\star}^0 = 0$. The second sub-case is when, initially, $C_j^0 = 1$, for all $1 \leq j \leq m$.

We shall first assume that there exists $j_\star$ such that $C_{j_\star}^0 = 0$. Then $y^t = 0$ for all $t \geq 1$, and, furthermore, the three variable nodes $\{x_{j_\star,1}, x_{j_\star,2}, x_{j_\star,3}\}$, corresponding to the variables that appear in the clause $C_{j_\star}$, will also evaluate to 0 at time $t = 1$, and remain 0 thereafter. At time $t = 2$, all $C_j$ will have more than one input equal to 0. Consequently, all $C_j^2 = 0$, $1 \leq j \leq n$. Thus, a single $C_{j_\star}^0 = 0$ assures the quick collapse to the *sink* stable configuration $0^{n+m+1}$.

Next, we examine the most interesting scenario, when the initial configuration $(x^0, C^0, y^0)$ is of the form $(x^{t=0}, 1^m, 1)$; that is, we assume that, initially, all $C_j^0 = 1$ as well as $y^0 = 1$. There are two possibilities: either $x^0$ is a satisfying truth assignment of the PE2-IN-3SAT instance $I$, or it is not a solution of $I$. If $I(x^0) = \mathit{false}$, then there must be at least one index $j$ such that the clause $C_j = 0$. If so, then the corresponding clause node $C_j$ of our SyDS will evaluate to zero, as well: $C_j^1 = 0$. Hence, at time $t = 2$, $y^2 = 0$, and also $x_{j,1}^2 = x_{j,2}^2 = x_{j,3}^2 = 0$. Thus, the resulting SyDS dynamics is the same as in case of an initial configuration with $C_j^0 = 0$, only

beginning one time step later. In particular, after three time steps, $(x^3, C^3, y^3) = 0^{n+m+1}$, and, of course, $(x^t, C^t, y^t) = 0^{n+m+1}$ for all $t \geq 3$.

Finally, we now assume that $x^0 = (x_1^0, ..., x_n^0)$ is a *satisfying* truth assignment of the PE2-IN-3SAT formula. Then, at time $t = 1$, all the clause nodes $C_j^1$ will re-evaluate to 1, since each clause $C_j$ in the Boolean formula will have exactly two *true* inputs if and only if each clause node $C_j$ of the corresponding SyDS has exactly $m + 1 + (3 - 1) = m + 3$ (i.e., *all but one*) of its inputs equal to 1. Similarly, $y^2 = y^1 = y^0 = 1$. Since all the nodes $C_j$ satisfy $C_j^1 = C_j^0 = 1$, it follows that each variable node $x_i$ will retain its old value: $x_i^1 = x_i^0 \wedge (\wedge_{j(i)} C_{j(i)}^0) = x_i^0 \wedge 1 = x_i^0$, and, similarly, also $x_i^2 = x_i^1 \wedge 1 = x_i^1 = x_i^0$. It is now immediate that any starting configuration of the form $(x^0, 1^m, 1)$, where the Boolean $n$-vector $x^0$ is a satisfying truth assignment of the given PE-2-IN-3SAT instance $I$, is a fixed point of $\mathcal{S}$.

By the above analysis, we see that the phase space of $\mathcal{S}$ has a rather simple structure: no cycles whatsoever, only short transients (the longest chains of transient states are of length 3), and the fixed points of $\mathcal{S}$ are precisely the *sink* $0^{n+m+1}$, the configuration $(x, C, y) = (1^n, 1^m, 0)$, and those configurations $(x, C, y)$ such that $C = 1^m$, $y = 1$, and the Boolean $n$-vector $x = (x_0, ..., x_n)$ is a satisfying truth assignment of $I$. In particular, if I has $|T|$ satisfying assignments, then $\mathcal{S}$ will have exactly $|T| + 2$ fixed points, and the claim of the theorem follows.

□

By the aforementioned invariance of fixed points with respect to the node update ordering, the next result on the hardness of counting FPs in symmetric Boolean SDSs is not at all surprising.

**Theorem 5.7.** *The problem of counting fixed point configurations of symmetric Boolean SDSs (abbreviated as* #FP-SYM-SDS*) is* **#P***-complete.*

*Proof.* In order to prove the theorem explicitly, as well as establish several other complexity-theoretic counting results for symmetric Boolean SDSs, we consider the following construction of an SDS $\mathcal{S}$' from the SyDS $\mathcal{S}$ used in the proof of the previous theorem.

- The underlying graph and the local node updating functions are as in the SyDS construction in the previous theorem.
- Let the node ordering be given by $\Pi = (y, C_1, ...., C_m, x_1, ..., x_n)$. Thus,
  $y^{t+1} = y^t \wedge (\wedge_{j=1}^m C_j^t)$,
  $C_j^{t+1} = \textit{ALL-BUT-ONE}\ \{y^{t+1}, C_1^{t+1}, ..., C_{j-1}^{t+1}, C_j^t, C_{j+1}^t, ..., C_m^t, x_{j_1}^t, x_{j_2}^t, x_{j_3}^t\}$,
  and, for any $i$ such that $1 \leq i \leq n$,
  $x_i^{t+1} = x_i^t \wedge (\wedge_{j(i)} C_{j(i)}^{t+1})$,
  where, as before, $C_{j(i)}$ denotes precisely those clause nodes that correspond to the clauses in the original Boolean formula in which the variable $x_i$ appears.

We will only sketch the analysis of what the phase space of $\mathcal{S}$' looks like, since much of the case analysis coincides with that for SyDS $\mathcal{S}$ in the previous theorem.

**Case 1:** $C^{t=0} \neq 1^m$. Since $y^1 = y^0 \wedge (\wedge_{j=1}^m C_j^0)$, and at least one of $C_j^0$ (say, $C_{j_\star}^0$) is 0, the node vertex $y$ will evaluate to $y^1 = 0$. Hence, each $C_j^1$ will have at least two zero inputs,

namely $y$ and $C_{j_\star}$, and hence the *ALL-BUT-ONE* function at node $C_j$ will evaluate to zero at time $t = 1$, for all $j$. Hence, if not all $C_j^0$ are initially equal to 1, $\mathcal{S}$' will collapse to the sink $0^{n+m+1}$ in a single step. We observe that there are exactly $(2^m - 1) \times 2^{n+1} = 2^{m+n+1} - 2^{n+1}$ configurations $\mathcal{C}$ such that $C^0 \neq 1^m$, all of which except for the sink $0^{n+m+1}$ are transient configurations and, moreover, each of these TCs is also a garden of Eden.

**Case 2:** $C^{t=0} = 1^m$. This is the more interesting case with several sub-cases to consider. First, if $(y^0, C^0, x^0) = (0, 1^m, 1^n)$, then it is straightforward to verify that this configuration is a fixed point that DOES NOT correspond to a solution of the corresponding instance $I$ of PE2-IN-3SAT. If, on the other hand, $y^0 = 0$ and $x^0 \neq 1^n$, then at time $t = 1$, there are at least two nodes holding the value 0; in particular, there exists $C_{j_\diamond}$ such that $C_{j_\diamond}^{t=1} = 0$ since the node update function at $C_{j_\diamond}$ has at least two zero inputs at time $t = 1$. Consequently, at time $t = 2$, *every* clause node $C_j$ will have at least two zero inputs, namely, $y^{t=2}$ and either $C_{j_\diamond}^{t=1}$ (if $j \leq j_\diamond$), or $C_{j_\diamond}^{t=2}$ (if $j > j_\diamond$). Therefore, $C_j^{t=2} = 0$ for all $j = 1, ..., m$, and subsequently $x_i^2 = 0$, for all $i = 1, ..., n$. Thus, in this case, the collapse to the sink $0^{n+m+1}$ takes (at most) two steps. Furthermore, the convergence from an initial state $(0, 1^m, x^0 \neq 1^n)$ to $0^{n+m+1}$ takes two steps *if and only if* $C_1^{t=1} = 1$ (and only one step, otherwise).

Finally, the remaining sub-cases to consider correspond to the initial configurations of $\mathcal{S}$' of the form $(y^0, C^0, x^0) = (1, 1^m, x^0)$. In this case, if $x^{t=0} = x_{sat}$ is a satisfying truth assignment for the PE2-IN-3SAT formula $I$, then the configuration $(1, 1^m, x^0)$ will be a fixed point of $\mathcal{S}$'. If, however, $x^0 = x_{false}$ is a falsifying truth assignment for $I$, then, at time $t = 1$, at least one of the $C_j^{t=1}$ will evaluate to 0, and consequently, at time $t = 2$, first the node $y$ will update to $y^{t=2} = 0$, and, since each $C_j^{t=2}$ will have at least two zero inputs, all the clause nodes will then evaluate to 0, and subsequently so will all the variable nodes $x_i^{t=2}$; i.e., $\mathcal{S}$' will converge to the sink $0^{n+m+1}$ in at most two steps. We observe that, in the case of an initial global configuration of the form $(1, 1^m, x_{false}^0)$, the convergence to $0^{n+m+1}$ will always take *exactly* two steps: that it cannot take more than two steps follows from the discussion above, whereas that it cannot take only one step stems from the observation that $y^1 = y^0 \wedge (\wedge_{j=1}^m C_j^0) = 1$, implying that $(y^1, C^1, x^1) \neq 0^{n+m+1}$.

Given the above analysis, it is immediate that $\mathcal{S}$' will have $|T| + 2$ fixed points if and only if the corresponding PE2-IN-3SAT formula has $|T|$ satisfying truth assignments. Hence, the **#P**-hardness of counting the fixed points of this restricted class of symmetric Boolean SDSs follows from the **#P**-hardness of counting the satisfying truth assignments of instances of PE2-IN-3SAT formulae. Since the membership of #FP-SYM-SDS in the class **#P** is easy to show, the claim of the theorem follows. □

### 5.7.1 Discussion: Estimating the Number of GEs and TCs

The configuration space of SDS $\mathcal{S}$' constructed in the proof of Theorem 5.7 above looks as follows. Since there are $n + m + 1$ nodes, there are $2^{n+m+1}$ global configurations in total. Among these, there are precisely $|T| + 2$ fixed points, where $|T|$ is the number of solutions of the corresponding PE2-IN-3SAT formula I. The number of these solutions is in the range $\{0, 1, ..., 2^n\}$. All of the $|T|$ fixed points corresponding to the solutions of I, as well as the fixed point $(y = 0, C = 1^m, x = 1^n)$, are *isolated fixed points*, in a sense that they do not have any in-coming transients. In other words,

each such configuration has a unique predecessor, namely, itself. The state $0^{n+m+1}$ is the "sink" for $\mathcal{S}$', in that all transient chains eventually end in $0^{n+m+1}$. All the remaining configurations are transient states, and, in particular, $\mathcal{S}$' does not have any temporal cycles. Furthermore, all the transient chains are very short, since every transient configuration is either a garden of Eden, or its predecessor is a garden of Eden; this is immediate from the fact that every convergence to the sink $0^{n+m+1}$ takes at most two steps.

How many transient configurations, then, does SDS $\mathcal{S}$' have? Let $|F| = 2^n - |T|$ denote the number of *falsifying* truth assignments for the PE2-IN-3CNF formula I. Since there are $|T| + 2$ fixed points and no temporal cycles, it is immediate that there are exactly $2^{m+n+1} - |T| - 2 = 2^{m+n+1} + |F| - 2^n - 2$ transient states; we denote the number of transient configurations by $|\#TC|$. Since $0 \leq |T| \leq 2^n$, it follows that $2^{m+n+1} - 2 \geq |\#TC| \geq 2^{m+n+1} - 2^n - 2$. Therefore, in order to determine the *exact* number of transient states of $\mathcal{S}$', one has to determine the number of satisfying truth assignments of the corresponding PE2-IN-3SAT formula I; but, even without knowing anything about the number of solutions of I, one can always readily estimate $|\#TC|$, since the fraction of all global configurations of $\mathcal{S}$' that are TCs lies, roughly, between $1 - \Theta(2^{-m})$ and 1. Hence, determining $|\#TC|$ for this class of symmetric Boolean SDSs *exactly* is hard, but *approximating* this number in a sense discussed earlier in this Section is relatively easy, and it gets easier as the number of clauses $m$ grows with respect to the number of variables $n$.

To determine the number of gardens of Eden is more involved, but we can estimate $|\#GE|$ as follows. Since any garden of Eden is also a transient state, one approach is to attempt to estimate the number of those transient states that are not gardens of Eden, i.e., that do have a predecessor. We recall that, in the phase space of $\mathcal{S}$', the longest transient chains are of length two; hence, by determinism of SDSs, it is immediate that *at least a half* of all transient states will actually be gardens of Eden. We argue that the fraction of the transient states that are actually also gardens of Eden can be (and, assuming a uniform distribution over the set of PE2-IN-3CNF instances, most often will be) considerably larger than just a half of all TC.

First, recall that any state of the form $(y, C, x) = (1, 1^m, x_{false})$, where $x_{false}$ stands for a choice of Boolean $n$-vector $x$ that corresponds to a *falsifying* truth assignment for formula $I$, is necessarily a garden of Eden; and the number of these configurations is exactly $|F| = 2^n - |T|$, the number of unsatisfying truth assignments for the PE2-IN-3SAT formula $I$. Likewise, any state of the form $(y, C, x) = (0, 1^m, x)$, where $x \neq 1^n$, is also necessarily a garden of Eden, and there are $2^n - 1$ such configurations.

The situation is more complicated with the configurations of the form $(y, C, x)$, where $C \neq 1^m$, whereas $y \in \{0, 1\}$ and $x \in \{0, 1\}^n$ are arbitrary. While each of these $2^{m+n+1} - 2^{n+1} - 1$ configurations, except for $0^{n+m+1}$ (as already discussed), is a transient configuration, it is not obvious at all which ones are gardens of Eden and which have a predecessor. A configuration of the form $(1, C, x)$ with $C \neq 1^m$ is either a GE configuration or else it has a predecessor among the $|F| = 2^n - |T|$ configurations of the form $(1, 1^m, x_{false})$. Similarly, each of the configurations $(0, C \neq 1^m, x)$ either is a GE or else has a predecessor among the $2^n - 1$ configurations of the form $(0, 1^m, x)$ such that $x \neq 1^n$. However, we recall that, while a transient chain that starts in a configuration of the form $(1, 1^m, x_{false})$ has to pass through an intermediate configuration of the form $(1, C, x)$ (where $C \neq 1^m$), before it reaches the sink $0^{n+m+1}$, for the configurations of the form $(0, 1^m, x)$ such that $x \neq 1^n$, it is possible that they immediately yield $0^{n+m+1}$, without

having to pass through an intermediate transient configuration $(0, C, x)$ (where $C \neq 1^m$); thus, it is possible that all configurations of the form $(0, C, x)$ such that $C \neq 1^m$ are not only transient, but also gardens of Eden.

In the sequel, we will often use the abbreviated notation whereby, for instance, the standard and formal (but quite long) '$(0, C, x)$ *such that* $C \neq 1^m$' is abbreviated to '$(0, C \neq 1^m, x)$'.

To provide rigorous upper and lower bounds on the total number of GE configurations, let us consider the two extreme cases. At one extreme, let's assume all of the configurations $(1, 1^m, x_{false})$ lead to the same state $(1, C_\star \neq 1^m, x_\star)$, and that all the configurations $(0, 1^m, x \neq 1^n)$ yield the sink $0^{n+m+1}$ immediately. If this is the case, then all the transient states, except for $(1, C_\star, x_\star)$, are also gardens of Eden. This gives an upper bound on the number of GE states: $|\#GE| \leq |\#TC| - 1$. At the other extreme, we consider the scenario where each state $(1, 1^m, x_{false})$ yields a distinct configuration $(1, C \neq 1^m, x)$, and each state $(0, 1^m, x \neq 1^n)$ evolves after one step into a distinct state $(0, C \neq 1^m, x) \neq 0^{n+m+1}$. Then the total number of garden of Eden states is only $|F|+2^n-1+((2^{m+n+1}-2^{n+1}-1)-|F|-(2^n-1)) = 2^{m+n+1}-2^{n+1}-1$. Hence, $|\#TC|-1 \geq |\#GE| \geq 2^{m+n+1}-2^{n+1}-1$. Since $|\#TC| = 2^{m+n+1}+|F|-2^n-2$ is maximized when the corresponding instance of PE-2-IN-3CNF SAT is *not satisfiable*, i.e. $|F| = 2^n$, the general bound on the number of garden of Eden states becomes $2^{m+n+1} - 3 \geq |\#GE| \geq 2^{m+n+1} - 2^{n+1} - 1$.

It is possible to make the given bounds on $|\#TC|$ and $|\#FP|$ sharper, if we notice that the nontrivial instances of the CNF-type Boolean formulae in general, and our PE2-IN-3SAT in particular, are *never* tautologies, and furthermore one can use combinatorial arguments to come up with lower bounds for the number of falsifying truth assignments. We shall not dwell upon a detailed combinatorial analysis based on various features of the underlying instance of PE2-IN-3SAT. Instead, we will only establish a crude lower bound for the number of falsifying assignments, $|F|$. First, we observe that both $0^{n+m+1}$ and $1^{n+m+1}$ are *always* falsifying truth assignment, for any *nonempty* instance of PE2-IN-3SAT. Second, consider any satisfying truth assignment, $x_{sat} \in \{0,1\}^n$. By definition of PE2-IN-3SAT, if we assign Boolean values to $x_1, ..., x_n$ according to $x_{sat}$, then each clause of the given instance will contain exactly two variables equal to 1. Hence, the component-wise negation of this Boolean vector will yield exactly one out of three variables being *true* in each clause, and therefore it will be a falsifying truth assignment of this PE2-IN-3SAT formula. These two facts that hold for any nontrivial PE2-IN-3SAT formula together imply that the number of falsifying truth assignments for any instance of PE2-IN-3SAT must satisfy $|F| \geq 2^{n-1} + 1$, or, equivalently, $0 \leq |T| \leq 2^{n-1} - 1$. This enables us to sharpen the previously given bounds on the number of fixed points, transient states and gardens of Eden of an SDS constructed from a PE2-IN-3SAT formula the way we constructed $\mathcal{S}$'. Concretely,

$2 \leq |\#FP| = |T| + 2 \leq 2^{n-1} + 1$;
$2^{m+n+1} - 2^{n-1} - 1 \leq |\#TC| = 2^{m+n+1} + |F| - 2^n - 2 \leq 2^{m+n+1} - 2$; and
$2^{m+n+1} - 2^{n+1} - 1 \leq |\#GE| \leq |\#TC| - |F| \leq 2^{m+n+1} - 2^{n-1} - 1$.

Thus, for the restricted class of symmetric Boolean SDSs constructed from the PE2-IN-3SAT instances as described above, *approximating* the number of fixed points is as hard as approximating the number of satisfying truth assignments of the corresponding instances of PE2-IN-3SAT, but estimating the approximate number of transient states and gardens of Eden, i.e., the fraction of all configurations that happen to be TC (GE), is tractable, and gets easier as the

number of clauses $m$ in the corresponding PE2-IN-3SAT formula grows.

In summary, enumerating the fixed points of Symmetric Boolean SDSs and SyDSs *exactly* is **#P**-complete, and approximating the number of FPs to within, say, $2^{|V|^{1-\epsilon}}$ is **NP**-hard, for any $\epsilon > 0$. Similarly, counting *exactly* all TCs or all GEs of a Symmetric Boolean S(y)DS is **#P**-complete, as well. The complexity of counting GEs and TCs in symmetric S(y)DSs *approximately*, however, cannot be deduced from our constructions herewith and, to the best of our knowledge, is still open.

While these results do establish that counting fixed points of symmetric Boolean SDSs/SyDSs is hard in both exact and approximate settings, the question of how hard in general is the approximation problem for the number of TC and GE for this basic general class of SDSs/SyDSs remains open. In particular, we are yet to encounter an interesting, nontrivial subclass of symmetric Boolean SDSs (or SyDSs) that would have some transient states (and garden of Eden states), i.e., that is not invertible, yet such that the number of these TC (GE) would be relatively small, and, in particular, possibly hard to approximately count. Thus, a possible conjecture, yet to be proved or disproved, could be that approximating the number of gardens of Eden for *non-invertible* SDSs and SyDSs is inherently easier than approximating the number of fixed points. From the formal dynamics perspective, if this conjecture is true, it would imply that for deterministic dynamical systems such as SDSs and SyDSs, there are nontrivial properties of the *backward dynamics* of those systems that are easier to predict than similar properties of the *forward dynamics*. From a distributed computing perspective, being able to estimate the number of GEs would provide some insight into what fraction of the system's configurations are *unreachable*, which, in turn, may have some implications for the hardness of determining *safety properties* of distributed information systems that can be abstracted as an appropriate kind of Sequential or Synchronous Dynamical Systems.

For arbitrary Boolean symmetric S(y)DSs, distinguishing between zero TC/GE states and more than zero such states is as hard as determining whether the given SDS is invertible; we recall that the garden of Eden existence problem (which is equivalent to the transient state existence problem) for this class of SDSs and SyDSs is **NP**-complete.

Several other consequences of the reductions in Theorems 5.6 and 5.7 follow immediately from the **NP**-completeness of the corresponding decision problems for PE2-IN-3SAT:

**Corollary 5.6.** *The following problems for symmetric Boolean SDSs/SyDSs are* **NP***-complete:*

*(i) Given a symmetric Boolean SDS/SyDS $\mathcal{S}$, does it have more than two fixed point configurations?*

*(i') Given a symmetric Boolean SDS/SyDS $\mathcal{S}$, does it have any fixed points different from the pre-specified $O(1)$ given fixed points?*

*(ii) Given a symmetric Boolean SDS/SyDS $\mathcal{S}$, does it have more than one configuration with a* non-unique *predecessor?*

*(ii') Given a symmetric Boolean SDS/SyDS $\mathcal{S}$, does it have any configurations different from a given set of $O(1)$ pre-specified configurations that have more than one predecessor?*

*(ii") Given a symmetric Boolean SDS/SyDS $\mathcal{S}$, does it have any configurations whatsoever with more than one predecessor?*

*(iii) Given a symmetric Boolean SDS/SyDS $\mathcal{S}$, does it have any garden of Eden configurations?*

Notice that we obtain (i') immediately from (i), and likewise with deducing (ii") from (ii').

Similarly, once (ii') is established, (iii) then becomes immediate from (ii'), since, whenever the underlying graph and consequently the configuration space are finite, the global map of an SDS or SyDS is injective if and only if it is surjective.

## 5.8 Section Summary

Large-scale distributed computational and communication systems are often characterized by the property that, while the individual components may be relatively simple and their behavior well-understood, due to the interaction among those components and the interdependencies among different processes taking place at different components, the overall system behavior can become extremely complex. This, in particular, makes the design of reliable such systems rather challenging. Equally importantly, formal verification of various properties of such systems, as well as the forecast of their likely future behavior patterns, become very difficult.

As a step towards understanding the kind of emerging complexity in such large-scale decentralized infrastructures, as well as towards developing a general theory of their computer simulation, we have adopted two particular classes of graph or network automata models, and a discrete dynamical systems perspective on those models. The primary methodological approach to studying properties of a dynamical system is to study its behavior, i.e., its *configuration space*. In this Section, we have considered those network automata as abstract discrete-time, discrete-state dynamical systems. We have specifically focused on the problems of counting how many stable configurations (FPs) and unreachable configurations (GEs) such dynamical systems have in their configuration spaces, when each of their nodes has only two distinct states, and updates according to a fairly simple Boolean function of the states of its neighboring nodes.

We have established that these (and some other, related) counting problems about Sequential and Synchronous Dynamical Systems are **#P**-complete, even when the following constraints on the structure of an SDS or SyDS *simultaneously* hold:

- the underlying graph of this SDS or SyDS is *planar, bipartite,* cycle-free and very sparse *on average* – in particular, this graph can be restricted to be a *star;*
- each local update rule is required to be a *monotone* Boolean function; and
- the nodes of the S(y)DS use *only two* different update rules (e.g., for the star graphs, one rule for the central node, and the other rule for everyone else).

In particular, we have shown in this work that important counting problems about distributed discrete dynamical systems are intractable when two important restrictions simultaneously hold. One, insofar as the *inter-agent local interactions* are concerned, we restricted the *communication topology*, that is, the underlying graphs of an SDS or SyDS, to the star graphs. Two, insofar as each agent's *individual behavior* is concerned, we limited the node update rules to monotone Boolean-valued functions encoded as Boolean formulae. The importance of our results for the SDSs and SyDSs defined on the star graphs stems from the fact that almost any computational graph-theoretic problem of interest is trivial when the graph instances are restricted to the (ordinary, static) star graphs. The importance of the results for the restricted local update rules lies in the fact that the corresponding decision problems for the monotone Boolean S(y)DSs (as well as for the monotone Boolean formulae in general) are tractable.

That there are important configuration space properties of Boolean SDSs and SyDSs that remain intractable even when those S(y)DSs are severely restricted in terms of both the allowable underlying graphs and the allowable local update rules is an indication that a very complex and, in general, unpredictable global dynamics can be obtained by coupling together only rather simple, monotonically behaving local interactions. Indeed, we have formally shown that this general observation holds true as long as there exists a *single agent* in such a dynamical system that is allowed to interact with a large number of other agents.

We have also shown the intractability of counting the stable configurations (FPs) of Boolean SDSs and SyDSs whose nodes are required to update according to another widely studied class of restricted update rules, namely, *symmetric* functions. The underlying graphs in the symmetric S(y)DSs for which we establish that hardness result, however, are also characterized by the existence of a node with a very large neighborhood. From a distributed computing perspective, such a node can be viewed as an abstraction of *central control*. The natural question then arises, for symmetric as well as monotone SDSs and SyDSs: can important configuration space properties, such as the number of stable configurations, remain intractable even when there is nothing resembling of central control, i.e., when no node has more than a handful of neighbors?

In the next Section, we will continue investigating the fundamental configuration space properties of various restricted classes of Boolean SDSs and SyDSs, and those problems' computational complexity, under the *uniform sparseness* restriction on the underlying graphs. By uniform sparseness we simply mean that every node in the graph has only $O(1)$ neighbors. We will establish that, for both monotone and symmetric Boolean S(y)DSs, counting FPs remains hard even for $k$-regular graphs where $k = O(1)$ is a small constant. Moreover, we will prove similar results for *simple threshold* SDSs and SyDSs, whose node update rules are *simultaneously* monotone and symmetric. In contrast, we will show that enumerating the fixed points of a (finite) simple threshold cellular automaton, where all the nodes update according to the same rule, is in principle feasible. It will then follow from these results that the collective dynamics of a large-scale multi-agent system made of simple, deterministically behaving reactive agents, in general, remains infeasible to predict even when both the individual agents' behaviors and the range of inter-agent interactions are severely restricted – as long as at least some heterogeneity among the agents is allowed.

# 6 Counting Problems About Uniformly Sparse Network Automata

In Section 5, we have established a number of computational hardness results about how difficult are the problems of determining several fundamental configuration space properties of two closely related classes of graph and network automata, namely, Sequential and Synchronous Dynamical Systems. The focus of that Section is on *enumeration* or *counting problems* about Boolean SDSs and SyDSs. Moreover, some of the complexity of counting results have been established when considerable restrictions are imposed on the structure of SDSs and SyDSs under scrutiny. The constraints have been imposed both on the nature of the node update rules and on the structure of the underlying network topologies.

We continue the general line of inquiry from Section 5 in the present Section, as well. Insofar as the update rules are concerned, we further pursue the investigation of collective dynamics of Boolean SDSs and SyDSs whose nodes are required to update according to restricted update rules such as the symmetric functions, the monotone functions, and the simple threshold functions. In terms of the network topologies, the focus of the entire Section will be on *uniformly sparse* underlying graphs. Those are the graphs where every node's neighborhood is of size $c = O(1)$ for small values of constant $c$.

In particular, we shall show that perhaps the most fundamental counting problem about discrete dynamical systems in general – that of enumerating the system's stable configurations – remains computationally intractable even for the Boolean SDSs and SyDSs with appropriately restricted update rules, and defined on uniformly sparse graphs. In contrast, we shall also show that enumerating the fixed points of simple threshold cellular automata, where *all nodes update according to the same update rule*, is computationally tractable. One immediate implication is that the *network topology sparseness*, by itself, does not preclude the underlying dynamical system from a complex behavior; however, such network sparseness appears to require to be coupled with some degree of network *nonuniformity*, as well as at least a minimal *heterogeneity* insofar as the individual agent behaviors are concerned, in order to yield a complex resulting dynamics.

Summary of the results in this Section that specifically pertain to the fixed point configurations is given in *Table* 2.

Throughout this Section, the SDSs, SyDSs, CA and SCA we study are *Boolean*, that is, the state space of a single node is $\{0, 1\}$. The Hopfield networks we study are *discrete* both in terms of their possible configurations and the nature of time. In order to be consistent with the existing literature on discrete Hopfield networks, yet also make comparisons with the Boolean (and hence, in particular, *binary-valued*) S(y)DSs and (S)CA a "fair game", we will exclusively consider the binary-valued discrete Hopfield networks (DHNs) such that the state space of a single DHN node is the set $\{-1, +1\}$.

Insofar as the restrictions on the underlying S(y)DSs are concerned, the focus of this Section is to establish several fundamental computational hardness of counting results for the network automata whose underlying graphs are *uniformly sparse*. In particular, the hardness of counting for different restricted classes of Boolean-valued update rules (symmetric, monotone, linear threshold, etc.) is established to hold even when the maximal node degree in the underlying graph is $d_{max}$; see *Table* 2.

| | SDSs and SyDSs | Binary-valued DHNs | CA and *Fair* SCA |
|---|---|---|---|
| symmetric | **#P**-complete for $d_{max} \geq 3$ (incl. 3-regular) | (usually not defined for the update rules that are not at least *linear threshold* ) | |
| monotone | **#P**-complete for $d_{max} \geq 3$ (incl. 3-regular) | **#P**-complete for $d_{max} \geq 3$ (incl. 3-regular) | |
| general linear threshold | **#P**-complete for $d_{max} \geq 3$ (incl. 3-regular) | **#P**-complete for $d_{max} \geq 3$ (incl. 3-regular) | |
| monotone linear threshold | **#P**-complete for $d_{max} \geq 3$ (incl. 3-regular) | **#P**-complete for $d_{max} \geq 3$ (incl. 3-regular) | |
| simple threshold (MSB) | **#P**-complete for $d_{max} \geq 4$ (incl. 4-regular) | **#P**-complete for $d_{max} \geq 4$ (incl. 4-regular) | in **P** for every rule radius $r \geq 1$ (1-D cellular spaces) |

Table 2: Summary of results on the computational complexity of *counting fixed points* in Section 6. For the hardness results, $d_{max}$ denotes the *maximum node degree* in the underlying uniformly sparse graph of an SDS, SyDS or DHN.

Clearly, the property that counting FPs of a particular kind of Boolean SDS or SyDS is intractable even when the maximum node degree in the graph is $d_{max}$ is a monotonic property: for example, if counting FPs of a *symmetric Boolean* SDS or SyDS (SYM-BOOL-S(Y)DS) is intractable when the maximum node degree in the underlying graph is equal to (or, alternatively, does not exceed) $d_{max} = 3$, then certainly this intractability will still hold if the maximum node degree is equal to (alternatively, does not exceed) $d_{max} = 4$. Hence, the statements in the two columns in *Table* 2 pertaining to S(y)DSs and DHNs should be interpreted as follows: a given variant of the #FP problem is hard when the upper bound on the node degrees in the underlying graphs is $d_{bound}$, for every $d_{bound} \geq d_{max}$, where $d_{max}$ is as given in the appropriate field in one of the two middle columns of the table. As a concrete example, for the SDSs and SyDSs with *symmetric* Boolean update rules, this lowest upper bound for which we have proven that the problem #FP is still **#P**-complete is $d_{max}^{symm} = 3$, whereas for Boolean SDSs and SyDSs with the simple threshold rules it is $d_{max}^{MBS} = 4$, etc. So, in particular, *for all* $d_{bound} \geq 3$, the #FP problem for symmetric Boolean SDSs and SyDSs is, in the worst case, intractable when the underlying graphs have their node degrees bounded by $d_{bound}$.

Moreover, as corollaries to the intractability of counting results by Vadhan [210] and Greenhill [73] in the context of (ordinary) sparse graphs and various restricted types of Boolean formulae, virtually all the complexity of counting results that hold for all graphs with a uniform bound on the node degrees that is greater than or equal to the appropriate $d_{max}$ given in *Table* 2, also hold on the special case of $k$-regular graphs, for all $k \geq d_{max}$ (where, again, $d_{max}$ pertains to *the particular* smallest upper bound on the node degrees for the given class of update rules, for which

the intractability of counting has been established).

Insofar as DHNs are concerned, various restrictions on the problem instances are usually imposed in terms of restricting the types of the allowed *weight matrices*, as opposed to the underlying graphs; in fact, in much of the Hopfield nets literature, the underlying graph is assumed to be a clique (that is, a fully connected graph) with an appropriate number of nodes. In order to be able to make the statements about the *hardness* (of counting) in spite of the *sparseness* (of the underlying graphs), clearly, we need a different convention: we consider two nodes, $x_i$ and $x_j$, of a DHN to be connected by an edge if and only if the corresponding weight is nonzero, $w_{ij} \neq 0$. More details will follow in subsection 6.2.

Finally, insofar as the sequential and parallel CA are concerned, we will consistently use the notation and apply the conventions from Section 4. In particular, given that we are focusing on *one-dimensional* cellular spaces, the only parameter, beside the number of nodes and (when applicable) the nature of boundary conditions, is the rule radius, $r$. The results on the number of fixed points of a simple threshold cellular automaton, as indicated in the rightmost column of *Table* 2, hold for *all* finite rule radii $r \geq 1$.

The rest of this Section is organized as follows. Subsection 6.1 addresses the complexity of counting the FPs when it comes to *symmetric* Boolean S(y)DSs defined on uniformly sparse graphs. In subsection 6.2, we address the problems of complexity of counting the fixed points, the predecessors and the ancestors of a given configuration for the Boolean S(y)DSs with *monotone* update rules that are defined over uniformly sparse graphs. We also address the related counting problems about *discrete Hopfield networks* in the second part of that subsection. Next, in subsection 6.3, the problem of counting fixed points is addressed in an even further restricted setting: this time, we require the node update rules to be *both* monotone and symmetric. For the comparison and contrast purposes, we then show in subsection 6.4 that the fixed points of one-dimensional cellular automata with simple threshold update rules actually can be effectively enumerated. Last but not least, in the final subsection we briefly summarize the significance and implications of the Section's results, and outline some open problems.

## 6.1 Counting Fixed Points of Uniformly Sparse Symmetric Boolean SDSs and SyDSs

In Section 5, we have established that counting FPs of *symmetric Boolean* SDSs and SyDSs is, under the usual assumptions in computational complexity, intractable. The constructions there are only *weakly* parsimonious; see [189, 204] for more details. Perhaps more importantly, the underlying graphs in those constructions have one or more nodes with an unbounded number of neighbors; those nodes would correspond to agents that can *directly communicate* with many other agents. In most large-scale MAS, however, an agent can directly communicate with only a handful of other agents, i.e., the underlying communication topology is *sparse*.

In the present subsection, we show that enumerating FPs of symmetric Boolean S(y)DSs remains **#P**-complete, even when the underlying graph, that is, the communication network topology of the agents, is *uniformly sparse* [206]. That is, the intractability of enumerating FPs of Sym-Bool-S(y)DSs still holds, even when each node of an SDS or SyDS has only $O(1)$ neighbors.

We recall that the constructions of symmetric Boolean SDSs and SyDSs in the previous

Section include a *central control* node, $y$, that has an unbounded degree. Also, the clause nodes $C_j$ in Theorems 5.6 and 5.7 are forming a clique, thus also being of unbounded degree. We now transform the SyDS and SDS constructions from subsection 5.7 so that the node $y$ is eliminated altogether, and so that each clause node $C_j$ has only $O(1)$ neighbors. This reduction in the maximum allowed node degree is going to be accomplished at the expense of doubling the number of the clause nodes, so that the resulting symmetric Boolean S(y)DS will have $n + 2m$ nodes in total, where, as before, $n$ is the number of variables and $m$ is the number of clauses in the original 3CNF Boolean formula.

Indeed, we shall eliminate the node $y$ in the constructions in Theorems 5.6 and 5.7, and, instead, for each clause node $C_j$, introduce its *cloned* clause node, $C_j^c$. We now connect each node $C_j$ to its clone $C_j^c$ and also to the clone of the successor clause node, $C^c_{j+1 \ (mod \ m)}$. We also delete all the edges among the original clause nodes $C_j$. Thus, each original clause node $C_j$ will now have *exactly five* neighbors: the three variable nodes, $x_{j_1}, x_{j_2}$ and $x_{j_3}$, and the two cloned clause nodes, $C_j^c$ and $C^c_{j+1 \ (mod \ m)}$.

We will also assume that the 3CNF SAT instance is from a restricted class of *monotone* 3CNF formulae where each variable $x_i$ appears in at most five clauses. This restriction does not affect the **#P**-completeness of the underlying counting problem. In fact, counting satisfying truth assignments of the *positive* (also called *monotone*) 2CNF formulae, abbreviated as MON-2CNF-SAT, is **#P**-complete even when each variable appears in at most five clauses [210]. Each of these MON-2CNF formulae can be converted into a special case of the MAJORITY-MON-3CNF formulae, in which a clause is satisfied if and only if *at least two out of three* unnegated variables (that is, their *majority*) appearing in that clause are true.

Namely, let's introduce a fresh Boolean variable $z$, and expand each monotone clause $(x_{j_1} \vee x_{j_2})$ in the MON-2CNF formula into $(x_{j_1} \vee x_{j_2} \vee z)$, as well as add a new clause, $(z \vee z \vee z)$. Clearly, the satisfying assignments of the original MON-2CNF formula are mapped in a one-to-one manner to the satisfying truth assignments of the resulting MAJORITY-MON-3CNF formula, while the number of appearances of each of the *old* variables $x_i$ has remained the same. Now, since only the new variable $z$ occurs in a number of clauses that is not bounded by $O(1)$, the problem of an unbounded number of appearances of a variable can be taken care of by replacing the single variable $z$ with a sequence of distinct new variables $z_1, z_2, ..., z_m$, by modifying each $C_j = (x_{j_1} \vee x_{j_2})$ from the original MON-2CNF into $C_j = (x_{j_1} \vee x_{j_2} \vee z_j)$, and by adding $m$ new clauses, $C'_j = (z_j \vee z_j \vee z_j)$, to the resulting MAJORITY-MON-3CNF formula.

Since this, restricted type of the counting problem #MAJORITY-MON-3CNF is equivalent to #MON-2CNF, and, therefore, **#P**-complete even when no variable occurs in more than five different clauses, and since the general #MAJORITY-MON-3CNF is clearly in the class **#P**, we conclude that the general problem of counting the satisfying assignments of a monotone 3CNF formula according to the MAJORITY rule is **#P**-complete *even when no variable appears in more than five different clauses*, as well.

We now turn to the construction of a *bounded-degree* symmetric Boolean SDS or SyDS from an instance of MAJORITY-MON-3CNF.

The variable nodes in the S(y)DS constructed from such a 3CNF formula with a restricted number of appearances of each variable will update according to the Boolean *AND* rule on (at most) six inputs. Each variable node, as before, is connected to those, and only those, clause nodes such that the corresponding variable in the Boolean 3CNF formula appears in the corresponding

clause. Hence, each of these variable nodes will have at most five neighbors. Since each of the *original* clause nodes has exactly five neighbors in total, the local update rule at each such node needs to be a symmetric Boolean function of six inputs. So, we define each node $C_j$ to update its state according to the "AT LEAST FIVE OUT OF SIX" rule.

Furthermore, we will also connect all the cloned nodes $C_j^c$ into a ring, so that the only neighbors of $C_j^c$ (beside $C_j$ and $C_{j-1 \ (mod\ m)}$) are $C^c_{j-1 \ (mod\ m)}$ and $C^c_{j+1 \ (mod\ m)}$. Finally, each of the cloned clause nodes $C_j^c$ will update according to the Boolean *AND* function of its five inputs (the states of its four neighbors plus the current state of itself).

Based on the described construction of a *bounded-degree* SYM-BOOL-S(Y)DS, we have the following strengthening of the results in Section 5 on the complexity of counting FPs of symmetric Boolean SDSs and SyDSs:

**Theorem 6.1.** *The problem of counting the fixed points of a Symmetric Boolean SDS or SyDS is* **#P***-complete, even when each node in the underlying graph of that S(y)DS is of a degree* $d_i = O(1)$*, and the nodes of the S(y)DS use only two different symmetric update rules.*

*Proof.* We need to establish that the construction preceding the statement of the theorem is indeed weakly parsimonious. To that end, we summarize the possible behaviors of the constructed S(y)DS.

If a single cloned clause node $C^c_{j_\star}$ at any point updates to 0, this node will eventually force all the remaining cloned clause nodes $C_j^c$, and consequently also all the original clause nodes $C_j$, to become 0s, as well. Similarly, if one of the original clause nodes $C_{j_\star}$ ever evaluates to 0, this will first cause its clone, $C^c_{j_\star}$, to evaluate to 0 (and stay at 0 thereafter), and that will, in turn, subsequently force all the other cloned clause nodes to become 0s. Since each of the original clause nodes $C_j$ will then have at least two neighbors stuck in the state 0, that will also ensure that eventually $C_j = 0$ for all $j = 1, ..., m$.

Therefore, if any of the clauses in the original formula is not satisfied, the corresponding S(y)DS will converge to the sink fixed point $0^{n+2m}$.

In contrast, if initially all $C_j^c = C_j = 1$, and the original Boolean formula is satisfied, then all the cloned clause nodes will remain at 1, and the corresponding global S(y)DS configuration is a fixed point corresponding to a satisfying truth assignment of the original Boolean formula. Hence, the satisfying truth assignments $x_{sat} \in \{0,1\}^n$ of the original Boolean formula are in a one-to-one correspondence with the S(y)DS configurations of the form $(x, C, C^c) = (x_{sat}, 1^m, 1^m)$. □

In fact, the upper bound on the maximum node degree in the underlying graph of a symmetric Boolean S(y)DS can be further reduced: the problem of exactly counting FPs in such SDSs and SyDSs remains **#P**-complete even when each node degree is required not to exceed 4 (instead of 5 as in the construction above). A weakly parsimonious reduction directly from MON-2CNF-SAT, where each variable in the 2CNF formula appears in no more than four clauses, can be used to establish that result. The details will follow in subsection 6.3, in the context of *simple threshold* Boolean SDSs and SyDSs.

### 6.1.1 Symmetric Boolean S(y)DSs on 3-regular Bipartite Graphs

Theorem 6.1, that we have originally established in [189], shows that arguably the most important counting problem about Boolean SDSs and SyDSs with symmetric update rules, namely #FP, remains intractable when each node in the underlying graph has only $O(1)$ neighbors. We have subsequently attempted to push the limits on the allowed underlying graphs even further.

As observed in [210] and further elucidated upon in [196, 206], the knowledge about the complexity of counting when the instances of Boolean logic or graph-theoretic or other combinatorial problems of interest are severely restricted has been rather limited. Following the work of Vadhan [210] and Greenhill [73], we wanted to fill that void when it comes to severely restricted models of *graph automata* [206].

In particular, one of the goals of our work following the early complexity of counting results found in [189, 190, 204] has been to determine the smallest upper bound on the node degrees in the underlying graphs for which counting problems of interest remain hard. In the case of symmetric SDSs and SyDSs, as well as their monotone counterparts, we have managed to push that degree down to 3, as will be shown below.

Furthermore, we have been investigating other restrictive assumptions on the underlying graphs' structure, such as *regularity* (i.e., when all nodes are required to have the same degree), *planarity* and *bipartiteness*. In that quest, the sharpest result on the hardness of counting in the context of symmetric Boolean SDSs and SyDSs is given in Theorem 6.2 below. That Theorem establishes the intractability of the FP enumeration problem even when the underlying graphs are simultaneously required to be both 3-*regular* and *bipartite*. Moreover, our construction used in the proof of this result is *(strongly) parsimonious*, in contrast to the weakly parsimonious reductions of Section 5 and Theorem 6.1.

**Theorem 6.2.** *The problem of enumerating the fixed point configurations of symmetric Boolean SDSs is* **#P***-complete even when the underlying graphs are simultaneously* 3-regular *and* bipartite*.*

*Proof.* We first establish the **#P**-hardness of the problem of enumerating FPs of a symmetric Boolean SDS (Sym-Bool-SDS) when the underlying graph is bipartite and has all node degrees bounded by 3. A simple modification will then yield the desired result for 3-regular graphs.

Let an instance of *monotone* 2CNF Boolean formula be given, such that no variable appears in more than three clauses. We assume that each clause contains *exactly two distinct* unnegated Boolean variables. For the ease of the subsequent SDS construction, we do not allow any redundant clauses or variables in this monotone 2CNF instance; in particular, without loss of generality, we assume that each variable appears in at least one clause. None of these restrictions affects the computational complexity of the underlying enumeration problem.

We recall once again that, by the results of Vadhan [210] and Greenhill [73], enumerating the satisfying assignments of such a monotone 2CNF Boolean formula is still **#P**-complete. From this monotone 2CNF formula we parsimoniously construct a *symmetric* Boolean SDS as follows. As before, we denote the number of variables by $n$ and the number of clauses by $m$. There is a node for each variable $x_i$ in the 2CNF formula, and a node for each clause $C_j$ in the formula. As before, a *variable node* $x_i$ is adjacent to a *clause node* $C_j$ if and only if, in the 2CNF formula, the corresponding variable appears in the corresponding clause. Unlike our prior constructions, no auxiliary nodes or edges are needed. The node update rules are as follows. Each variable node,

$x_i$, updates according to the Boolean *AND* of its own current state, and the current states of (up to three) clause nodes that this variable node is adjacent to. Each clause node, $C_j$, updates according to the following *symmetric* (but *non-monotone*) update rule:

$$C_j \leftarrow \begin{cases} 1, & \text{if } C_j + x_{j_1} + x_{j_2} \neq 1 \\ 0, & \text{otherwise} \end{cases} \tag{21}$$

where, in the 2CNF formula, the clause $C_j$ is given by $C_j = (x_{j_1} \vee x_{j_2})$. We remark that, in expression (21) that specifies the update rule for the clause nodes $C_j$, the operation '+' denotes the ordinary arithmetic addition.

The node update ordering is any permutation $\Pi$ such that first all the variable nodes update (in an arbitrary sequential order), and then all the clause nodes update (again, in an arbitrary order).

We claim that the given reduction from Mon-2CNF Satisfiability to #FP-Sym-Bool-SDS is parsimonious. There are two main cases to consider. Let's first assume that the SDS starts from an initial configuration where there exists a clause node, $C_{j\star}$, such that initially $C^0_{j\star} = 0$. Consider the subconfiguration $(C_{j\star}, x_{j\star_1}, x_{j\star_2})$. If initially $(x^0_{j\star_1}, x^0_{j\star_2}) = (0,0)$, then this subconfiguration is already a part of a (local) temporal two-cycle, i.e., $(C_{j\star}, x_{j\star_1}, x_{j\star_2})$ will alternate between $(0,0,0)$ and $(1,0,0)$ *ad infinitum*, irrespective of the states of other SDS's nodes. If initially $(x^0_{j\star_1}, x^0_{j\star_2}) \neq (0,0)$, then, at time $t = 1$, $(C^1_{j\star}, x^1_{j\star_1}, x^1_{j\star_2}) = (1,0,0)$, and the same local temporal two-cycle $\{(1,0,0),(0,0,0)\}$ is reached. Thus, if initially there exists a clause node that is in the state 0, such starting configuration may be either transient or cyclic, but it cannot be a fixed point.

The second scenario to consider is when, initially, $C^0_j = 1$, for all $j$, $1 \leq j \leq m$. There are two subcases: one is when the Boolean vector $(x_1, ..., x_n)$ corresponds to a truth satisfying assignment of the underlying 2CNF formula, and the other is when this Boolean vector falsifies the formula. In the former case, each clause node will have at least two out of three of its inputs equal to 1 at time $t = 1$, so it will evaluate to 1. Similarly, each variable node that updates according to Boolean *AND* will have all its neighboring nodes in the state 1, so it will itself stay at its current value. Hence, it is immediate that each configuration of the form $(C_1, ..., C_m, x_1, ..., x_n) = (1, ..., 1, x_1, ..., x_n)$, where $(x_1, ..., x_n)$ constitutes a satisfying assignment to the monotone 2CNF formula, is a fixed point.

In contrast, if $(x_1, ..., x_n)$ falsifies the 2CNF formula, then there exists a clause node $C_k$ that has both its neighboring variable nodes in the state 0, and, since this clause node is currently in the state 1, it will have exactly one of its three inputs equal to 1. Hence, this clause node will change its state to 0, i.e., the corresponding starting configuration cannot be a FP.

Moreover, the corresponding variable nodes, $x_{k_1}$ and $x_{k_2}$, update according to Boolean *AND* that includes their respective current states, and hence for all time steps $t \geq 0$, $x_{k_1} = x_{k_2} = 0$ will hold. Hence, the triad subconfiguration $(C_k, x_{k_1}, x_{k_2})$ will keep alternating between $(1,0,0)$ and $(0,0,0)$, thereby ensuring that no global configuration that contains a clause node such as $C_k$ (that is, one that corresponds to a falsifying clause in the Boolean Mon-2CNF formula) can either be, or evolve to, a fixed point.

Therefore, we conclude that the satisfying truth assignments of a given Mon-2CNF formula are in a one-to-one correspondence with the fixed point configurations of the symmetric SDS

constructed from that formula. The claim of the theorem for the SDSs whose underlying graphs have the maximum node degree of 3 is now immediate.

Moreover, two straightforward modifications can ensure that the theorem holds for 3-regular graphs, as well. First, we can construct an SDS from a monotone 3CNF (instead of a monotone 2CNF) formula; this modification certainly does not make the corresponding problem of enumerating the satisfying truth assignments any easier. Secondly, the monotone 3CNF formula can be required to be such that each variable $x_i$ appears in *exactly* three clauses, as opposed to *at most* three clauses (for justification, see, e.g., [73]). These two modifications maintain the **#P**-completeness of the underlying enumeration version of monotone CNF SAT (in this case, monotone 3CNF SAT). Thus, the resulting underlying graph in our SDS construction, in addition to being uniformly sparse and bipartite, can be made to be 3-regular, as well.

□

The claims of Theorem 6.2 have now been established for the symmetric Boolean SDSs. It is straightforward to verify that the same construction works for the corresponding Bool-Sym-SyDSs, where all the nodes update synchronously in parallel, as opposed to sequentially according to a permutation Π that satisfies the constraint discussed in the proof above.

**Corollary 6.1.** *Enumerating the fixed point configurations of symmetric Boolean SyDSs is* **#P***-complete even when the underlying graphs are required to be simultaneously* 3-regular *and* bipartite.

The corresponding case analysis, that establishes that the reduction from Mon-3CNF-SAT to #FP-Sym-Bool-SyDS is parsimonious, is rather similar to the analysis for the Bool-Sym-SDSs; we therefore leave out the details.

To summarize, determining the number of stable configurations is **#P**-complete even for some very simple CFSM-based discrete dynamical systems, such as the symmetric Boolean SDSs and SyDSs defined over uniformly sparse graphs. Moreover, as Theorem 6.2 shows, this intractability holds even when the underlying network topology is regular, bipartite and very sparse. In particular, insofar as the sparseness aspect is concerned, our results for S(y)DSs defined on 3-regular graphs (and on more general sparse graphs whose node degrees do not exceed 3) are, to the best of our knowledge, the sharpest of their kind to date [206]. Namely, all previous results on the computational hardness of counting stable configurations for S(y)DSs (e.g., [189, 190, 204]), as well as for parallel and asynchronous Hopfield networks (e.g., [55, 57, 137]), are for the underlying graphs with the maximum node degree of 4 or higher.

## 6.2 Counting Configurations of Uniformly Sparse Monotone Boolean SDSs and SyDSs

*Monotone* Boolean functions, formulae and circuits have been extensively studied in many areas of computer science, from machine learning to connectionist models in AI to VLSI circuit design [217]. Cellular and other types of network automata with the local update rules restricted to monotone Boolean functions have also been of a considerable interest (e.g., [17, 200]). The problem of counting FPs in *monotone* Boolean SDSs and SyDSs is originally addressed in [188, 190]. It is shown there, and summarized in Section 5, that, in general, counting FPs of such

S(y)DSs either exactly or approximately is computationally intractable. This intractability holds even for the graphs that are simultaneously bipartite, planar, and very sparse *on average* [188, 190]. In particular:

**Lemma 6.1.** [190] *Exactly enumerating the fixed points of a monotone Boolean SDS or SyDS defined over a star graph, and such that the update rule of the central node is given as a* MONOTONE 2CNF *Boolean formula of size $O(n)$, where $n$ is the number of nodes in the star graph, is* **#P***-complete.*

Moreover, by the results of D. Roth in [157], subsequently strengthened by S. Vadhan in [210], the problem of *approximately* counting FPs in the setting as in Lemma 6.1 above can be readily shown to be **NP**-hard [190].

In summary, enumerating the fixed points of *monotone* Boolean SDSs and SyDSs defined on bipartite, planar and sparse on average underlying graphs *exactly* is **#P**-complete, and for any $\epsilon > 0$, *approximating* the number of FPs in such monotone S(y)DSs to within $2^{|V|^{1-\epsilon}}$ is **NP**-hard. Our next goal is to show that the hardness of the exact enumeration of FPs for monotone S(y)DSs holds even when the underlying graphs are required to be *uniformly sparse*. We will also argue that, as a consequence of our construction in the proof of Theorem 6.3 below, the problem of enumerating the stable configurations of certain other types of discrete dynamical systems, such as *discrete Hopfield networks*, is also in general computationally intractable. Moreover, this intractability holds even for such systems that are defined on very sparse underlying graphs or networks.

Given the importance of the number of stable configurations of a Hopfield network viewed as an *associative memory* (e.g., [69]), we now informally introduce *discrete Hopfield networks*, and briefly summarize what has been known about the problem of counting their stable configurations.

A *discrete Hopfield network* (DHN) [81] is made of $n$ binary-valued nodes; the set of node states is, by convention, $\{-1, +1\}$. Associated to each pair of nodes $(v_i, v_j)$ is (in general, real-valued) *weight*, $w_{ij} \in \mathbf{R}$. The *weight matrix* of a DHN is defined as $W = [w_{ij}]_{i,j=1}^{n}$. Each node also has a fixed *threshold*, $h_i \in \mathbf{R}$. A node $v_i$ updates its state $x_i$ from time step $t$ to step $t+1$ according to

$$x_i^{t+1} \leftarrow sgn(\sum_{j=1}^{n} w_{ij} \cdot x_j^t - h_i) \tag{22}$$

In the sequel, we will not bother to explicitly distinguish between an S(y)DS's or DHN's node, $v_i$, and this node's state, $x_i$; the meaning will be clear from the context.

In the standard DHN model, the nodes update synchronously in parallel, similarly to the classical cellular automata and the SyDSs as defined in Section 5. However, *asynchronous Hopfield networks*, where the nodes update sequentially, one at a time, have also been studied [57, 81]. In these sequential DHNs, unlike SDSs, it is not required that the nodes update according to a *fixed permutation*. However, these differences are inconsequential insofar as the fixed points are concerned [132].

In much of the Hopfield networks literature, the weight matrix $W$ is assumed symmetric, i.e., for all pairs of indices $\{i, j\}$, $w_{ij} = w_{ji}$ holds. A DHN is called *simple* if $w_{ii} = 0$ for all $i = 1, ..., n$ [57]. Simple DHNs are thus a generalization of *memoryless* finite cellular automata with linear threshold update rules [63, 225].

In [55], Floreen and Orponen establish the following two interesting results:[37]

(i) the problem of determining the number of fixed point configurations of a *simple* discrete Hopfield network, with a symmetric weight matrix $W = [w_{ij}]$ such that all the weights $w_{ij}$ are integers (and with $w_{ii} = 0$ along the main diagonal), is **#P**-complete; and

(ii) the problem of determining the number of predecessor configurations of a given configuration of a simple discrete Hopfield network, with a symmetric weight matrix $W = [w_{ij}]$ such that all the weights $w_{ij}$ are from the set $\{-1, 0, +1\}$, is **#P**-complete.

For proving (i), Floreen and Orponen devise a Hopfield network that is relatively dense, i.e., with quite a few non-zero weights $w_{ij}$. This would correspond to an SDS or SyDS where there are, informally speaking, a considerable number of nodes (in particular, more than just $O(1)$ of them) each of which has many neighbors. In contrast, our result in Lemma 6.1 allows only for a *single node* that has a large neighborhood; see [188, 190] for more details.

Prior to proving the first main result of this subsection, for the sake of completeness, we state the following

**Lemma 6.2.** *Counting the fixed points of an arbitrary SDS or SyDS all of whose nodes use Boolean-valued linear threshold update rules is* **#P***-complete.*

We shall show next that the result (i) from [55] can be considerably strengthened along several dimensions. That is, the hardness of counting FPs will be proven to still hold even when the following restrictions on problem instances are *simultaneously* imposed:

– the underlying graphs will be required to be *uniformly sparse*, with no node degree exceeding 3;

– all linear threshold update rules will be restricted to *monotone* functions by disallowing negative weights;

– only two (positive) integer values for the weights will be allowed; and

– each S(y)DS node will choose one from only two allowed monotone linear threshold functions.

Since each node of an SDS or SyDS in the Theorem below is required to have only $O(1)$ neighbors, the issue of *encoding* of the local update rules, that is discussed in detail in [190], is essentially irrelevant here. In particular, even a truth table with one row for each combination of the values of a given node's neighbors is permissible [189, 190]. In the sequel, Bool-Mon-S(y)DS will stand for a *monotone Boolean* SDS or SyDS.

**Theorem 6.3.** *Counting the fixed points of* Bool-Mon-S(y)DS*s* exactly *is* **#P***-complete, even when all of the following restrictions on the structure of such an S(y)DS* simultaneously *hold:*

- *the monotone update rules are* linear threshold functions *– in particular, monotonicity of the linear threshold update rules implies that all weights satisfy $w_{ij} \geq 0$;*
- *the S(y)DS is with memory, and such that, along the main diagonal, $w_{ii} = 1$ for all indices $i$, $1 \leq i \leq |V|$ (where $|V|$ denotes the number of the S(y)DS's nodes);*
- *at most two different positive integer weights are used by each local update rule;*

[37] We (slightly) rephrase these results from the language originally used in [55] into the discrete dynamical systems language we use in [188, 189, 190] and this technical report, in order to make the comparison and contrast with our results more transparent.

– *each node has at most three neighbors in the underlying graph of this S(y)DS;*

– *only two different monotone linear threshold rules are used by the S(y)DS's nodes.*

*Proof.* We first describe the construction of a Bool-Mon-SyDS from an instance of a *Boolean monotone 2CNF* (Mon-2CNF) formula [62] such that no variable appears in more than three different clauses. We then outline why is this reduction from the problem of counting satisfying assignments of such a formula to the problem of counting FPs in the resulting SyDS *weakly parsimonious* [62].

Let's assume that a Mon-2CNF Boolean formula is given, such that there are $n$ variables, $m$ clauses, each variable appears in at least one clause, and no variable appears in more than three clauses. In particular, these requirements imply that $m = O(n)$, but we shall keep $m$ and $n$ as two distinct parameters for clarity.

The corresponding SyDS $\mathcal{S}$ is constructed as follows. To each variable in the formula corresponds a variable node, and to each clause, a clause node. In addition, a *cloned clause node* is introduced for each of the original $m$ clause nodes. Thus, the underlying graph of $\mathcal{S}$ has exactly $n + 2m$ nodes. A variable node is adjacent to a clause node if and only if, in the Boolean formula, the corresponding variable appears in the corresponding clause. Each clause node is adjacent to its clone. Finally, the cloned clause nodes form a ring among themselves. Therefore, the underlying graph of this SyDS looks as in *Figure* 6.

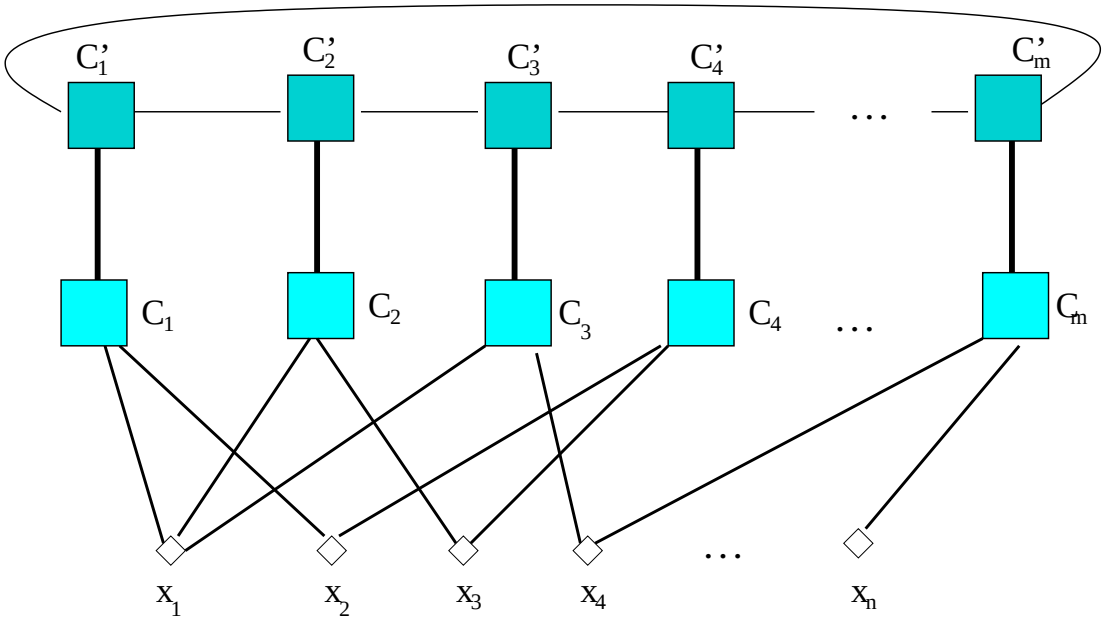



Figure 6: The underlying graph of a bounded-degree monotone linear threshold Boolean S(y)DS in the construction of Theorem 6.3.

The original clause nodes in the figure above are marked $C_j$, the cloned clause nodes are primed, as in $C'_j$, and the variable nodes are denoted by $x_i$.

We remark that we shall continue to slightly abuse the notation the same way we have done in the previous two Sections. In particular, we will use $x_i$ to denote *both* a variable in the Boolean formula, and the corresponding *variable node* in the S(y)DS or discrete Hopfield network we are constructing. Similarly, $C_j$ will denote both clauses in the Boolean formulae and *clause nodes* in the S(y)DSs or Hopfield networks that are being constructed from those formulae. The intended meaning will be clear from the context.

With that convention in mind, we now define the update rules for the clause nodes, the cloned clause nodes, and the variable nodes of the SyDS that we are constructing from a Mon-2CNF Boolean formula. The cloned clause nodes $C'_j$ and the variable nodes $x_i$ will update according to the Boolean *AND* rule. The original clause nodes, $C_j$, will update according to the following monotone linear threshold update rule:

$$C_j \leftarrow \begin{cases} 1, & \text{if } 2C'_j + C_j + x_{j_1} + x_{j_2} \geq 4 \\ 0, & \text{otherwise} \end{cases} \tag{23}$$

where $x_{j_1}, x_{j_2}$ is a shorthand for the two variable nodes that are adjacent to the clause node $C_j$; that is, in the Boolean formula, the $j$-th clause is given by $C_j = (x_{j_1} \vee x_{j_2})$.

The given construction can be slightly rephrased, in order to emphasize that the resulting SyDS also satisfies the *symmetry requirement* as it is usually defined in the Hopfield networks literature, namely, so that the underlying matrix of weights is a symmetric matrix. To that end, the Boolean *AND* rule used by the cloned clause nodes can be written in an equivalent, but more *linear-threshold-like*, form:

$$C'_j \leftarrow \begin{cases} 1, & \text{if } 2C_j + C'_j + C'_{j-1} + C'_{j+1} \geq 5 \\ 0, & \text{otherwise} \end{cases} \tag{24}$$

Notice that the function defined in equation (24) evaluates to 1 if and only if all of its inputs are 1, and thus, indeed, the given formula is nothing but a linear-threshold-like way of writing the ordinary Boolean *AND* of four variables. If this latter convention on how we write the update rules at the cloned clause nodes and the variable nodes is adopted, then the resulting $\mathcal{S}$ can be also viewed as a discrete Hopfield network with parallel node updates; we will formalize this observation in the next subsubsection.

Similarly, the Boolean *AND* rule applied to the variable nodes can be written in the required linear threshold form as follows. For each variable $x_i$ in the Mon-2CNF formula from which we are constructing S(y)DS, let $a_i$ denote *the number of clauses in which* $x_i$ *appears*; thus, under our assumptions, for any $i \in \{1, ..., |V|\}$, we have $a_i \in \{1, 2, 3\}$. We now define the variable node update rules as

$$x_i \leftarrow \begin{cases} 1, & \text{if } x_i + \sum_{\{j: x_i \in C_j\}} C_j \geq a_i + 1 \\ 0, & \text{otherwise} \end{cases} \tag{25}$$

One can easily verify that the resulting weight matrix of the constructed DHN, with the node update rules represented as the three linear threshold rules in equations (23) – (25), is, indeed, *symmetric*.

We now show that the reduction from the counting problem #Mon-2CNF-SAT to the counting problem #FP for the constructed SyDS is, indeed, weakly parsimonious. To that end, we

will just summarize the case analysis. If, at any time step $t$, any of the cloned clause nodes $C'_j$ evaluates to 0, that will ensure that, within no more than $\frac{m}{2}+1$ parallel steps, all the cloned clause nodes will become 0, and stay in state 0 thereafter. This will also cause all the original clause nodes' states $C_k$, and, consequently, also all the variable nodes' states $x_i$, to become 0, as well. Thus, if at any point a single cloned clause node's state becomes 0, the entire SyDS will eventually collapse to the "sink" fixed point $0^{n+2m}$. Clearly, this sink FP does not correspond to a satisfying assignment to the original Boolean formula.

Now, the only way that no cloned clause node ever evaluates to 0 is that the following two conditions simultaneously hold:

– each $C'_k$ and $C_k$ is initially in the state 1, for $1 \leq k \leq m$; and

– the initial states $x_i$ of the variable nodes are such that they correspond to a satisfying truth assignment to the variables in the original Boolean formula.

If these conditions hold, then each such global configuration $(x^n, C^m, C'^m) = (x^n_{sat}, 1^m, 1^m)$ is a fixed point of $\mathcal{S}$, where $x^n_{sat} \in \{0,1\}^n$ is a short-hand for any $n$-tuple of Boolean values that corresponds to a satisfying truth assignment $(x_1, ..., x_n)$ to the original monotone 2CNF formula. Moreover, the satisfying truth assignments of the original Boolean formula are in a one-to-one correspondence with these non-sink FPs of $\mathcal{S}$.

Since no variable in the MON-2CNF formula from which we are constructing the SyDS appears in more than three clauses, each variable node $x_i$ in the SyDS has at most three neighbors. Since we use 2CNF, each clause node $C_j$ has two variable node neighbors, plus one cloned clause neighbor, $C'_j$, for the total of three neighbors. Finally, each cloned clause node $C'_j$ clearly has exactly three neighbors. In particular, by the result of C. Greenhill in [73], we can make the underlying graph of SyDS $\mathcal{S}$ be 3-regular, and the **#P**-completeness of the counting problem #FP will still hold.

We also observe that *only two* different monotone linear threshold functions are used in the DHN construction above.[38] Furthermore, at most two different integer weights are used in each of the (presentations of) linear threshold functions used as the update rules.. Hence, the claim of the Theorem follows insofar as the monotone linear threshold SyDSs are concerned.

Finally, by the invariance of FPs with respect to the node update ordering [132], it follows that exactly enumerating the fixed point configurations of the monotone linear threshold SDSs defined on uniformly sparse graphs is **#P**-complete, as well. □

In the construction above, the SyDS dynamics from *every* starting global configurations that is not of the form $(x^n_{sat}, 1^m, 1^m)$ will eventually converge to the sink state $0^{n+2m}$. In particular, the *basin of attraction* of $\mathcal{C} = 0^{n+2m}$ includes all configurations of the form $(x^n_{unsat}, 1^m, 1^m)$, where $x^n_{unsat}$ is a shorthand for an ordered $n$-tuple of Boolean values that corresponds to an *unsatisfying* (i.e., falsifying) truth assignment to the corresponding variables $x_1, ..., x_n$ in the original MON-2CNF formula. The rest of the configurations in the sink's basin of attraction are such that $(C^m, C'^m) \neq (1^m, 1^m)$ (and where $x^n \in \{0,1\}^n$ is arbitrary).

Hence, in order to determine *exactly* the size of the basin of attraction for the sink state $\mathcal{C} = 0^{n+2m}$, that is, the number of this configuration's ancestors, we must be able to exactly determine

[38] In terms of the update rules *representation*, there are three different such presentations in total, since the Boolean *AND* is written in two equivalent but different forms for the cloned clause nodes and the variable nodes, respectively. However, the total number of distinct functions used is indeed two, regardless of how many different representations of those functions are used.

the number of falsifying truth assignments to the original MON-2CNF Boolean formula. It is easy to see that one can find an ordering Π under which the same claim holds for the corresponding BOOL-MON-SDS. As a consequence, we have

**Corollary 6.2.** *The problem of counting exactly all the* ancestors *of an arbitrary configuration of a* BOOL-MON-S(Y)DS*, denoted* #ANC*, is* **#P***-hard. Moreover, this intractability result holds even when all restrictions from* Theorem 6.3 *are simultaneously imposed on the S(y)DS's structure.*

### 6.2.1 Counting Various Configurations of Discrete Hopfield Networks

We now turn to the corresponding hardness of counting results for discrete Hopfield networks with appropriately restricted weight matrices. We start with the problem of fixed point enumeration in the context of Hopfield nets where each of the nodes has exactly one bit of memory – namely, its own (binary-valued) current state.

**Theorem 6.4.** *Determining the exact number of stable configurations of a standard (that is, parallel or* synchronous*) discrete Hopfield network is* **#P***-complete even when all of the following restrictions on the weight matrix* $W = [w_{ij}]$ *simultaneously hold:*

- *the matrix is symmetric:* $w_{ij} = w_{ji}$ *for all pairs of indices* $i, j \in \{1, ..., |V|\}$ *(where* $|V|$ *denotes the number of nodes in the underlying graph of this DHN);*
- $w_{ii} = 1$ *along the main diagonal for all* $i \in \{1, ..., |V|\}$*;*
- $w_{ij} \in \{0, 1, 2\}$ *for all pairs of indices* $i, j \in \{1, ..., |V|\}$*;*
- *each row and each column of* $W$ *has* at most three *(alternatively,* exactly three*) nonzero entries off the main diagonal.*

*The same claim holds for the sequential (also called* asynchronous *in the literature) discrete Hopfield networks, with the same restrictions on their weight matrices as for the synchronous DHNs above.*

*Proof.* In case of the DHNs whose nodes update synchronously in parallel, the claim holds by virtue of Theorem 6.3, since an SyDS that is constructed in the proof of that theorem can also be viewed as a parallel discrete Hopfield network whose weight matrix satisfies all the above listed conditions.[39] Insofar as the asynchronous DHNs whose nodes update in arbitrary sequential orders are concerned, while indeed these sequences of node updates need not be repetitions of a fixed permutation as in the corresponding SDSs, this difference can be easily shown to be immaterial insofar as the fixed point configurations are concerned. Therefore, Theorem 6.4 about discrete Hopfield networks is nothing but rephrasing Theorem 6.3, with parallel DHNs in place of SyDSs with monotone linear threshold update rules, and asynchronous/sequential DHNs replacing SDSs with the same kind of update rules. □

[39] For simplicity of the argument, in this proof sketch we are ignoring the syntactic difference that the state space of a node in a Hopfield network is $\{-1, +1\}$, not $\{0, 1\}$.

Next, we consider the problems of enumerating predecessors as well as all ancestors of a given Hopfield network configuration. We shall establish the computational complexity of those two related counting problems in the context of *simple* DHNs, whose weight matrices satisfy $w_{ii} = 0$ for $\forall i \in \{1, ..., |V|\}$.

Before we proceed with a formal reduction from the problem #Mon-2CNF-SAT to the problem #Pred-DHN of enumerating all predecessor configurations of a given DHN configuration, we establish the following additional conventions. First, the reduction will be from the Mon-2CNF Boolean formulae with each variable appearing in at least one, and in at most (alternatively, exactly) four clauses. Second, we will abandon the usual convention in the Hopfield networks literature that the underlying graph is fully connected (i.e., a clique), and instead consider those pairs of vertices $\{v_i, v_j\}$ such that $w_{ij} = w_{ji} = 0$ not to be connected by an edge at all. We will require from the underlying DHN's weight matrix $W$ to be *symmetric* in the usual, Hopfield networks terminology sense; as a consequence, the underlying graph of such a discrete Hopfield network will be undirected, which is also in accordance with our convention about S(y)DSs. Third, in the construction used in proving Theorem 6.3, we will eliminate the cloned clause nodes $C'_j$ and, instead, connect the ordinary clause nodes into a ring.

We recall that, in a DHN, the set of possible states of a node is traditionally $\{-1, +1\}$ (instead of $\{0, 1\}$); while not essential, we will adopt this common practice through the rest of this Section insofar as the Hopfield networks are concerned. With that in mind, we define the update rule of a clause node $C_j$ to be

$$C_j \leftarrow \begin{cases} +1, & \text{if } 2C_{j-1} + 2C_{j+1} + x_{j_1} + x_{j_2} > 3 \\ -1, & \text{otherwise} \end{cases} \tag{26}$$

For each variable $x_i$ in the Mon-2CNF formula from which we are constructing our DHN, let $a_i$ denote *the number of clauses in which* $x_i$ *appears*; thus, under our assumptions, for any $i \in \{1, ..., |V|\}$, we have $a_i \in \{1, 2, 3, 4\}$. We now define the variable node update rules as

$$x_i \leftarrow \begin{cases} +1, & \text{if } \sum_{\{j: x_i \in C_j\}} C_j > a_i - 1 \\ -1, & \text{otherwise} \end{cases} \tag{27}$$

Thus a variable node $x_i$ updates to $+1$ if and only if *all* of the clause nodes $C_{j(i)}$ corresponding to those clauses in the formula in which variable $x_i$ appears are currently in the state $+1$.

Finally, we observe that the resulting weight matrix $W$, while symmetric and with all entries $w_{ij} \in \{0, 1, 2\}$, also has $w_{ii} = 0$ along the main diagonal; therefore, the constructed Hopfield network is *simple* (i.e., *memoryless*) [55, 57].

We are now ready to establish the second main result of this subsection:

**Theorem 6.5.** *The problem #Pred of determining the exact number of predecessors of a given configuration of a discrete Hopfield network is* **#P***-complete. Moreover, this claim holds even when all of the following restrictions on the Hopfield net's weight matrix* $W = [w_{ij}]$ *are simultaneously imposed:*

- *the matrix is symmetric:* $w_{ij} = w_{ji}$ *for all pairs of indices* $i, j \in \{1, ..., |V|\}$*;*

- *the Hopfield network is* simple*, i.e.,* $w_{ii} = 0$ *along the main diagonal for all indices* $i \in \{1, ..., |V|\}$*;*
- $w_{ij} \in \{0, 1, 2\}$ *for all pairs of indices* $i, j \in \{1, ..., |V|\}$*;*
- *each row and each column has* at most / exactly four *nonzero entries.*

*Proof.* The claim of the Theorem will follow from the fact that the satisfying truth assignments to the Boolean variables $x_1, ..., x_n$ in the original MON-2CNF Boolean formula are in a one-to-one correspondence with the set of all predecessors of the configuration $(+1)^{n+m}$ in the Hopfield net constructed from that formula. The case analysis is similar to that in the proof of Theorem 6.3. In particular, every configuration with at least one clause node $C_j$ in the state $(-1)$ will eventually converge to the sink fixed point $(x^n, C^m) = ((-1)^n, (-1)^m)$. Among the configurations of the form $(x^n, C^m) = (x^n, (+1)^m)$, those and only those such that the $n$-tuple $x^n$ corresponds to a satisfying truth assignment to the original MON-2CNF Boolean formula[40] will evolve to the other fixed point configuration, $(1^n, 1^m) = (+1)^{n+m}$. Moreover, this convergence to $(+1)^{n+m}$ is easily seen to take a single parallel transition. That is, the predecessors of $(+1)^{n+m}$ are precisely the configurations of the form $(x^n_{sat}, (+1)^m)$. □

The discussion in the proof sketch above also shows that *all* ancestors of the configuration $\mathcal{C} = (+1)^{n+m}$ are that configuration's predecessors; that is, the convergence from every configuration in the basin of attraction of $\mathcal{C}$ takes exactly one global parallel step.

**Corollary 6.3.** *The problem* #ANC *of determining the exact number of all ancestors of an arbitrary configuration of a simple discrete Hopfield network is, in the worst case,* **#P***-hard. Moreover, this intractability holds even when all the restrictions from* Theorem 6.5 *on the Hopfield network instances are simultaneously imposed.*

## 6.3 Counting FPs of Uniformly Sparse Simple Threshold Boolean SDSs and SyDSs

After studying symmetric and monotone Boolean SDSs and SyDSs defined on uniformly sparse underlying graphs separately, we next turn our attention to the intersection of those two restricted classes of Boolean S(y)DSs. That is, we will now require each node's update rule to be *simultaneously* monotone and symmetric. To be consistent with the existing literature (e.g., [16, 17, 198, 200]), we refer to such update rules as to *simple threshold functions.*

It turns out that, for the Boolean SDSs and SyDSs with simple threshold update rules, the problem of counting their FPs remains computationally intractable even when their underlying graphs are uniformly sparse with a small $c = O(1)$ bound on the allowable node degrees:

**Theorem 6.6.** *The problem of counting the fixed points of a Simple Threshold Boolean S(y)DS is* **#P***-complete, even when each node in the underlying graph of the S(y)DS is of a degree* $d_i \leq 4$*, and the nodes of this S(y)DS use only two different Boolean simple threshold update rules. In particular, the result also holds for the 4-regular underlying graphs.*

[40] Here, we identify the Boolean value FALSE of a variable in the MON-2CNF formula with the corresponding discrete Hopfield network variable node's state $-1$, whereas the Boolean value TRUE of a variable is mapped to the state $+1$ of the corresponding DHN variable node.

*Proof.* We assume that an instance of a MON-2CNF Boolean formula is given, such that there are $n$ variables and $m$ clauses in the formula, and each variable appears in at least one clause. The construction of a simple threshold Boolean SDS or SyDS such that no node degree exceeds four from an instance of MON-2CNF Boolean formula proceeds as follows. As before, there is a *variable node* for each variable $x_i$ in the Boolean formula, and a *clause node* for each clause $C_j$ in the formula. For each clause node $C_j$, we introduce its *clone*, $C'_j$. Next, we connect each node $C_j$ to its clone $C'_j$ and also to the clone of the successor clause node, $C'_{j+1 \ (mod \ m)}$. Thus, now each original clause node $C_j$ has *exactly four* neighbors: the two variable nodes, $x_{j_1}, x_{j_2}$, and the two cloned clause nodes, $C'_j$ and $C'_{j+1 \ (mod \ m)}$.

We also assume that the MON-2CNF instance is from a restricted class of monotone 2CNF formulae where each variable $x_i$ appears in *at most four* clauses. This restriction does not affect the **#P**-completeness of the underlying counting problem. In fact, the counting version of the problem of *monotone* 2CNF satisfiability, abbreviated as MON-2CNF-SAT, remains **#P**-complete when each variable appears in at most four clauses [73, 210].

Furthermore, we define the S(y)DS's underlying graph so that the cloned clause nodes $C'_j$ are connected into a ring. In particular, this implies that the only neighbors of node $C'_j$, beside $C_j$ and $C_{j-1 \ (mod \ m)}$, are the nodes $C'_{j-1 \ (mod \ m)}$ and $C'_{j+1 \ (mod \ m)}$.

Therefore, the resulting graph of this SIMPLE THRESHOLD BOOLEAN S(Y)DS looks as in *Figure* 7 below:

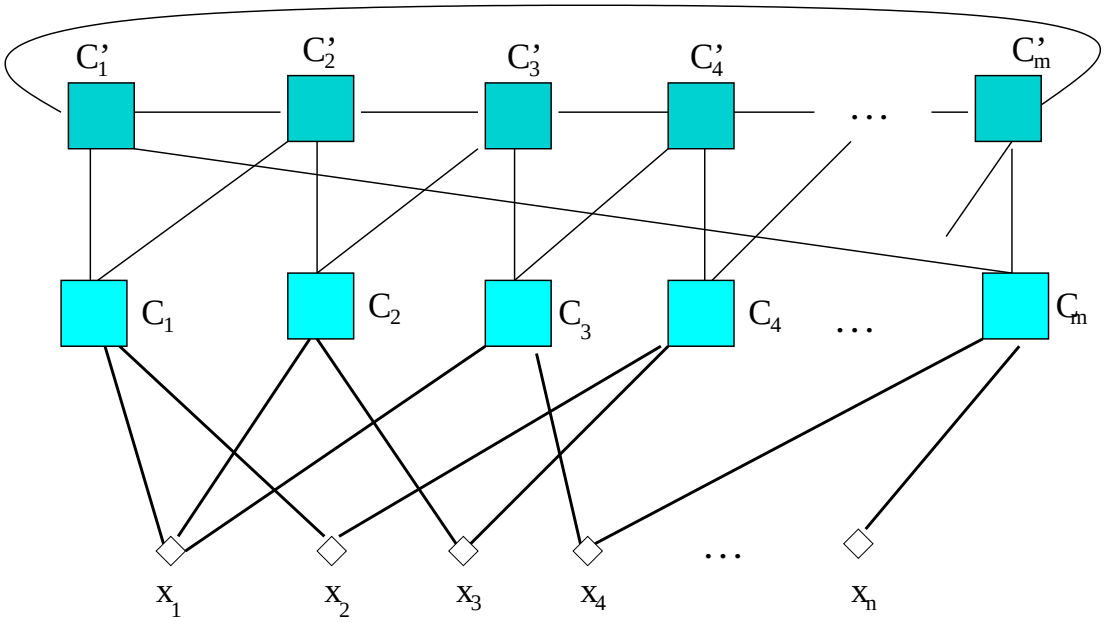



Figure 7: The underlying graph of a bounded-degree simple threshold Boolean S(y)DS in the construction of Theorem 6.6.

Each node $C_j$ will update its state according to the Boolean AT-LEAST-FOUR-OUT-OF-FIVE simple threshold function; this update rule is evaluated on exactly five inputs. We define the variable nodes' update rule in the constructed SDS or SyDS to be the Boolean AND function on

at most five inputs. Finally, we define the update rule of all cloned clause nodes $C'_j$ to be the Boolean AND function on exactly five inputs – namely, the states of such a node's four neighbors plus the current state of that cloned clause node itself. We observe that both AT-LEAST-FOUR-OUT-OF-FIVE and AND update rules are, indeed, simultaneously monotone and symmetric, i.e., they are of a *simple threshold* variety.

Since we are presently interested in the fixed point configurations *only*, there is no necessity to explicitly specify the permutation $\Pi$ for the SDS case: the properties of FPs discussed in the sequel hold for all the SDS node permutations, as well as for the corresponding SyDS where the nodes update synchronously in parallel.

What remains to be established is that the given construction of a simple threshold Boolean SDS or SyDS from an instance of MON-2CNF formula with no variable appearing in more than four clauses, insofar as a mapping from the satisfying assignments of the formula to the fixed points of the resulting S(y)DS is concerned, is actually weakly parsimonious.

We summarize the possible behaviors of the constructed simple threshold S(y)DS. If a single cloned clause node $C'_{j_\star}$ at some point updates to 0, this node will eventually force all the remaining cloned clause nodes, and consequently also all the original clause nodes, to subsequently become 0s, as well. Therefore, every configuration $\mathcal{C}$ that contains at least one cloned clause node in state 0 will eventually converge to the sink FP $0^{n+2m}$.

In contrast, if initially all $C'_j = C_j = 1$, and the original Boolean formula is satisfied, then all the cloned clause nodes will remain at 1, and the corresponding global S(y)DS configuration is a fixed point corresponding to a satisfying truth assignment of the original Boolean formula. If, however, even a single one of the original clause nodes $C_{j_\star}$ ever evaluates to 0, this will first cause its clone, $C'_{j_\star}$, as well as that cloned clause node's right neighbor $C'_{j_\star+1}$, to evaluate to 0 (and stay at 0 thereafter). This will, in turn, subsequently force all the other cloned clause nodes $C'_l$ to eventually update to 0, and then stay stuck at zero, as well. In particular, since each of the original clause nodes $C_j$ will then have at least two neighbors stuck in the state 0, this will also ensure that eventually $C_j = 0$, for all $j = 1, ..., m$. Thus, the FPs of this SDS or SyDS are precisely the sink state $0^{n+2m}$ and those configurations $(x_{sat}, 1^m, 1^m)$ such that the $n$-tuple of Boolean values $x_{sat} = (x_1, ..., x_n)$ corresponds to a satisfying truth assignment to the original MON-2CNF Boolean formula.

Moreover, one can make the underlying S(y)DS graph 4-regular, by requiring that each variable in the MON-2CNF Boolean formula appears in *exactly* four (as opposed to *at most* four) clauses; indeed, it has been shown in [73] that, under that additional constraint, the resulting #MON-2CNF problem remains **#P**-complete. Hence, the problem of exactly enumerating all FPs of the resulting simple threshold SDS or SyDS defined on a 4-regular underlying graph is **#P**-complete, as well.

□

When it comes to *sparseness* of the underlying S(y)DS graphs, we recall from the earlier subsections of this Section that the #FP-S(y)DS problem is **#P**-complete both for the symmetric Boolean SDSs and SyDSs that are not necessarily monotone, and for the monotone linear threshold Boolean SDSs and SyDSs that are not necessarily symmetric, even when those S(y)DSs are defined on graphs of maximal node degree equal to 3 (alternatively, on 3-regular graphs). Whether the maximal node degree in Theorem 6.6 can be reduced from 4 to 3 (alternatively, whether the graph can be made 3-regular), so that the hardness of counting still holds with the

update rules that are *simultaneously* symmetric and monotone linear threshold, remains an open problem.

## 6.4 Counting Fixed Points of Simple Threshold Cellular Automata

We now state the main results on fixed points of simple threshold cellular automata. The results are essentially identical for parallel and sequential CA, so we won't bother making the explicit distinction between the two in the sequel. We shall provide a complete characterization of the possible types of FP configurations, and state the basic results on how many FPs simple threshold CA with different monotone symmetric update rules can have; these results will be in the context of 1-D cellular spaces. Moreover, to be consistent with Section 4, we shall assume the CA *with memory.*

Among all simple threshold rules, that is, all $k$-threshold Boolean-valued functions on $2r+1$ inputs where $k \in \{0, 1, ..., 2r+2\}$, we have argued in [200] that the most interesting one is the MAJORITY, $\delta = MAJ$, rule, for which $k = r+1$. The $MAJ$ rule is the only rule among the $k$-threshold rules that treats 0s and 1s equally. Some consequences of that property are discussed in Section 4; in the sequel, we will specifically focus on the implications of that property on the number of stable configurations.

Namely, it will be shown below that the $MAJ$ CA have *much more numerous* fixed point configurations than any other simple $k$-threshold rules, for $k \in \{0, 1, ...r, r+2, ..., 2r+2\}$ [200]. Before we quantify just how numerous the FPs of $MAJ$ 1-D CA are, and how that number compares with the corresponding numbers for other simple threshold update rules, we first address the following question: what are the possible structures of FPs of the 1-D $MAJ$ CA? We characterize these structures next.

**Lemma 6.3.** *Any fixed point configuration $\mathcal{C}$ of a finite or one-way infinite 1-D MAJ CA belongs to one of the following three types of configurations:*

*(i) a global configuration made of all 0s or all 1s; or*

*(ii) a* spatially periodic *configuration with the spatial period that depends on $r$, but not on the number of nodes $n$ (where the rule radius $r \geq 1$ is a fixed positive integer)* [39]*; or*

*(iii) $\mathcal{C}$ is made of some positive number of* stable blocks *of sufficiently many consecutive 0s and sufficiently many consecutive 1s; i.e., there exist positive integers $l_1, l_2, l_3, ...$ such that $\mathcal{C}$ is either of the type $\mathcal{C} = 0^{r+l_1}1^{r+l_2}0^{r+l_3}...$ or of the type $\mathcal{C} = 1^{r+l_1}0^{r+l_2}1^{r+l_3}...$* [68].

Notice that the configurations of type (i) are just a special case of the type (iii) configurations, such that there is exactly one block of consecutive nodes in the same state (either each node is 0 or each is 1). We remark that the *OR* CA (where Boolean OR is viewed as "1-threshold" function) and the *AND* CA ("(2r+1)-threshold") have fixed points of type (i) *only*, whereas other $k$-threshold rules (for $k \neq r+1$), in general, may have FPs of *both* type (i) and type (ii). For example, spatially periodic configuration on $4m$ nodes given by $(001)^m$ is stable for the 3-threshold function with $r = 3$ (that is, a node updates its state to 1 iff at least 3 out of 7 of its inputs are currently in state 1) and assuming the *circular boundary conditions.*

The nontrivial fixed point, non-spatially-periodic configurations of type (iii), with an arbitrary number of stable blocks (the only restrictions being imposed by the radius $r \geq 1$ and the total

number of nodes in the underlying cellular space), however, are unique to the $\delta = MAJ$ rule, that is, in the 1-D case, the simple $k$-threshold rule with $k = r + 1$.

How many FPs do different simple threshold CA have? For all the rules other than MAJ, this number of FPs is small and can be determined by examining a constant number of *neighborhood (sub)configurations* of at most $O(r)$ nodes, where we recall that $r \geq 1$ is an arbitrary but *fixed* positive integer. For example, for the *AND* cellular automata as well as the *OR* cellular automata, the only two FPs are the configuration made of all 0s and the one made of all 1s. For $(k, r) = (3, 3)$ CA, in addition to these two, the only other FPs are the ones of type (ii), and the only such configurations are of the type[41] $(001)^m$ where $m$ is a positive integer, or $(001)^{\aleph_0}$ in the infinite at-least-3-out-of-7 CA case.

For the special case of the $\delta = MAJ$ update rule, we state the following result:

**Lemma 6.4.** *Let a $\delta = MAJ$ parallel or sequential CA be given on a* finite *1-D cellular space, and let $n$ be the total number of nodes. Then this MAJ CA has a number of fixed points that is* exponential *in $n$ and that can be effectively estimated.*

We remark that the estimation of the total number of type (iii) FPs in such *MAJ* CA is based on a conceptually straightforward, but non-trivial combinatorial counting argument. Only a few more FPs are then to be added to obtain the total FP count – those that are of type (i) (there are exactly two of them) and of type (ii) (there are still only a handful of those, where the exact number of type (ii) FPs depends on $r$, as well as the residue $n$ $(mod\ r)$ and the details of the boundary conditions).

*Proof.* To establish the claims of the Lemma, we will first outline the combinatorial argument that, when carried out in detail, would enable one to determine the *exact* number of FPs of type (i) and (iii). We will then provide an asymptotic *lower bound* on the total number of FPs of a one-dimensional *MAJ* CA that is exponential in the total number of nodes, $n$.

Assume a 1-D Majority CA has $n$ nodes, and let its rule radius $r \geq 1$ be fixed. Without loss of generality, assume that $n$ is even and much larger than $r$.[42] Consider all fixed point configurations that are made of *exactly* $l$ stable blocks, where $l \geq 1$ and each block is of length at least $r + 1$. How many such FPs are there?

If the CA domain is Boolean, then there are exactly two such FPs made of exactly one block (namely, $0^n$ and $1^n$). Similarly, assuming the circular boundary conditions, it turns out that there are exactly $n(n - (2r + 1))$ such FPs made of *exactly two* stable blocks. To see why, consider the FPs made of a block of exactly $r + 1$ consecutive 0s and exactly $n - (r + 1)$ consecutive 1s. Those FPs are $0^{r+1}1^{n-r-1}$, $10^{r+1}1^{n-r-2}$, ..., $1^{n-r-1}0^{r+1}$, $01^{n-r-1}0^r$, ..., $0^r1^{n-r-1}0$. Due to the *wrap-around* effect (courtesy of the circular boundary conditions), there are exactly $n$ such FPs. Similarly, there are exactly $n$ FPs made of exactly $r + 2$ consecutive 0s and $n - r - 2$ consecutive 1s.

The possible values of the number of consecutive 0s, that we denote $bs(0)$ (for the *block size* of a block of consecutive 0s), range from $r + 1$ to $n - r - 1$, i.e., there are $n - 2r - 1$ of them. For

[41] The exact form of the type (ii) FP depends on the details of the boundary conditions; e.g., 1001001 is a FP for appropriate fixed boundary conditions but not for the periodic boundary conditions, and *vice versa* in the case of configuration 001001.

[42] We recall that, while arbitrary, $r$ is fixed, i.e., it is a positive integer constant.

each of those values, and assuming circular boundary conditions, there are exactly $n$ FPs made of a block of $bs(0)$ consecutive 0s and a block of $bs(1) = n - bs(0)$ consecutive 1s. This gives the total of $n(n-2r-1)$ fixed points made of exactly one stable block of zeros and exactly one stable block of ones,for the total of two blocks, and when one takes into account the effects of circular boundary conditions.

One can proceed with a similar analysis for the exact number of FPs made of exactly $l$ stable blocks, for each $l \geq 2$. In doing so, one has to be careful not to count certain configurations two or more times. For instance, is the fixed point $0^{r+2}1^{n-(2r+4)}0^{r+2}$ to be considered as a configuration made of two stable blocks, one of length $2r+4$ and the other of length $n-(2r+4)$, or is it to be treated as an FP made of three stable blocks, since each sub-block of $r+2$ consecutive 0s is, by itself, stable? We observe that the stability of a block of $r+2$ consecutive nodes in the same state holds regardless of the nature of boundary conditions. Thus, a careful application of the combinatorial *inclusion-exclusion principle* is required, in order to count every FP of type (iii) (including the two of type (i)) *exactly once*. However, the somewhat involved combinatorics does not change the fact that this calculation can be done efficiently. Similar argument holds for enumeration of the type (ii) FPs, and a careful consideration of all candidate *spatially periodic* configurations. The bottom line is that, from the computational complexity standpoint, exactly determining the number of FPs of *MAJ* (S)CA is in (the functional analogue of)[43] the class **P**.

Our second goal is to establish that this number of fixed points, denoted —#FP—, is *exponential* in $n$. To that end, we again proceed with analyzing how many FPs made of $l$ stable blocks there are for each *eligible* value of the parameter $l$. This time, we are interested in *lower bounds* that are, however, "safe" from the asymptotic analysis standpoint, in a sense that no overlaps between different values of $l$ are possible (see the discussion in the previous paragraph).

For $l = 2$, if we ignore the boundary conditions and treat the $n$ nodes in the ring as a linearly ordered array, there would be (at least) two FPs for each permissible size of the first block – one for the first block being made of 0s, and the other for the first block being made of 1s. The possible sizes of the first of the two stable blocks range from $r+1$ to $n-r-1$, i.e., there are $n-2r-1$ possibilities. Hence, there are at least $2(n-2r-1)$ FPs made of exactly two stable blocks of consecutive nodes, where the configuration is treated as a linear array, and no boundary conditions effects are taken into account.

More generally, given $l$, it can be shown using the combinatorial technique of *generating functions* that there are *at least*

$$|\#FP(l)| \;\geq\; \frac{2(n-rl-1)!}{(l-1)!(n-rl-l)!} \tag{28}$$

fixed points made of exactly $l$ blocks. A lower bound on the total number of FPs can now be obtained by summing over all values of $l$ (where $1 \leq l \leq \frac{n}{r+1}$). That is, such a lower bound on the total number of FPs of type (iii) (including the two configurations of type (i) as the special case – see Lemma 6.4) is given by

$$|\#FP| \;\geq\; 2 \sum_{l=1}^{\frac{n}{r+1}} \frac{(n-rl-1)!}{(l-1)!(n-rl-l)!} \tag{29}$$

[43]The class of *function problems* (as contrasted to *decision problems*) that are solvable in deterministic polynomial time is often referred to as **FP** in the literature; for more, see, e.g., [140].

To establish the claim of the Lemma about an exponential number of those fixed points, it suffices to consider $l = \frac{n}{2r}$ and prove that there are exponentially many FPs made of exactly $\frac{n}{2r}$ blocks. Namely, using Stirling's formula, it can be shown that the number of FPs made of $l = \frac{n}{2r}$ blocks is asymptotically greater than $6^{\frac{n}{4r}}$. In particular, there exists an $\epsilon = \epsilon(r) > 0$ such that $|\#FP| = \Omega((1+\epsilon)^n)$.

The argument above applies to the *Boolean* (that is, binary-valued) 1-D MAJORITY CA. The essence of the argument readily generalizes to those CA whose nodes have more than two possible states. In fact, establishing the claim of the Lemma for the domains of sizes strictly greater than two is easier than for the binary domain. In particular, if the number of possible states of each node (i.e., the size of the domain) is 3 or higher, then the exponential in $n$ number of fixed points is immediate. Namely, for the 3-valued such cellular automata the number of FPs with exactly $l$ stable blocks with *the fixed size* of each block is $3 \cdot 2^{l-1}$, which is exponential in $n$ for sufficiently large values of parameter $l$.

Again, to obtain the exact total number of FPs of type (iii) in the non-binary-valued MAJ 1-D CA cases, one first needs to consider all possible block sizes and combinations of different values assigned to different blocks for the fixed number of blocks $l$, and then to perform summation over all permissible values of parameter $l$. However, we are not interested in the combinatorial details, but only in establishing the exponential in $n$ nature of the number of fixed points of the sequential and parallel CA with $\delta = MAJ$.

□

In case of an infinite MAJ CA defined over a (one-way or two-way) *infinite line*, the following extension of Lemma 6.4 holds:

**Lemma 6.5.** *An infinite* 1D-(S)CA *with* $\delta =$ MAJ *and any* $r \geq 1$ *has* uncountably many *fixed points.*

*Proof.* See Theorem 4.3. □

To summarize, the FP configurations of simple threshold CA can be very few (e.g., only two for AND and OR rules) or a great many (the MAJ rule and a cellular space with sufficiently many nodes); yet, either way, their number can be computationally efficiently calculated. This is in stark contrast with respect to SDSs and SyDSs all of whose nodes update according to (possibly different) threshold update rules: in case of the SDS (SyDS) network automata, the exact and even the approximate enumeration of their FPs are computationally intractable problems.[44]

## 6.5 Section Summary, Discussion and Open Problems

Section 5 introduced two relatively recent models of *network automata* called Sequential and Synchronous Dynamical Systems, and provided a number of computational complexity results about the (in)tractability of determining various configuration space properties of those network

[44] Of course, these intractability claims, just like all other similar computational hardness results in this technical report, hold under the usual assumptions in computational complexity theory, namely, that the classes **#P** and **NP** are strictly larger than the class **P** of the problems that are solvable in deterministic polynomial time.

automata. The emphasis in that endeavor was on certain *counting problems*. This general line of inquiry is continued in the present Section. The common theme throughout Section 6 is the *uniform sparseness* of the underlying network topology of a multi-agent system abstracted as an appropriately restricted type of a Sequential or Synchronous Dynamical System. The main lesson to be learned is that all fundamental counting problems, in general, remain hard – even when each agent can directly interact with only a small constant number of other agents. Thus, in particular, there are aspects of the sparsely networked multi-agent systems' collective dynamics that are provably hard to predict, as long as some degree of the communication network's nonuniformity and the heterogeneity among the individual agents of those systems are allowed.

The most fundamental problem that we see as the natural extension of the work presented in this Section, is to identify to what extent is this unpredictability of collective dynamics primarily or solely due to the heterogeneity among the agents vs. the nonuniformity of the underlying network topology vs. the coupling between these two system parameters.

In particular, are there S(y)DSs with appropriately restricted node update rules, yet such that *all* nodes update according to one and the same update rule, for which the counting problems of interest (such as #FP, #GE or #PRED) are still intractable? If examples of such CA-like S(y)DSs can be found, is the homogeneity of the individual node behaviors equally affecting the *forward dynamics* counting problems such as #FP, and the *backward dynamics* problems such as #PRED and #ANC? How does the exact nature of the single update rule affect the computational complexity of counting (and other configuration space) problems of interest?

Alternatively, can one find examples of SDSs and SyDSs defined on highly uniform CA spaces (such as the straight lines, rings, or discrete Cartesian 2-D grids), and such that their nodes use only two distinct update rules from an appropriately restricted class of Boolean functions, yet so that the fundamental counting problems – and possibly other problems about various configuration space properties – remain computationally intractable, in contradistinction to the same problems in the context of the classical CA and SCA defined on the same underlying cellular spaces? If such examples exist, is it sufficient that a single node behaves slightly differently than the rest of the nodes? If not, are there examples where only $O(1)$ nodes update according to a rule different from the update rule of the rest of the nodes? Are there fundamental computational complexity differences among the counting and other problems of interest, depending on whether the update rules are restricted to symmetric, monotone, or simple threshold Boolean functions?

Insofar as the counting complexity of *restricted instances* [210] of problems about SDSs, SyDSs and DHNs is concerned, the focus of the present Section has been on the counting problems of interest when the underlying graphs are *uniformly sparse*, i.e., such that *every node* has a small neighborhood. However, there are other kinds of restrictions on the underlying graphs whose implications on the computational complexity of the counting (and other) problems of interest are worth while further investigating. Among various such restrictions on the structure of the underlying graphs commonly found in the literature, in addition to uniform sparseness, we find *bipartiteness* and *planarity* to be particularly interesting. Thus a possible directions for extending the work in Sections 5 and 6 is to study the complexity-theoretic implications of the underlying graphs of S(y)DSs and DHNs being required to be *bipartite* and/or *planar*, especially when these restrictions are combined with *sparseness.*

The results in Section 5 about SDSs defined on *star graphs* indicate that most counting problems of interest can be expected to remain hard for the underlying graphs that are simultaneously

planar, bipartite and sparse on average – as long as there are one or more nodes with large neighborhoods, that can therefore simulate central control. Moreover, the results in subsection 6.1 of this Section suggest that the combination of bipartiteness and uniform sparseness of a discrete dynamical system's network topology can still yield the global dynamics that is, in the worst case scenario, provably intractable to predict. The least explored combination of restrictions on the network topology, therefore, is that of the underlying graphs being required to be both planar and uniformly sparse.

Planarity is an important and frequently encountered restriction on the graph structure that naturally arises in a variety of application domains. For example, in the VLSI circuit layout design, circuits made of logic gates and their interconnections often have to be embeddable in the plane. In statistical physics, the restriction of the general Ising model and spin glass models to the lattice structure where the pairwise interactions satisfy the planarity constraint is of a major importance insofar as its physical meaning is concerned; incidentally, it has been shown relatively recently that the planar instances precisely correspond to the computationally tractable instances of spin glasses [90]. Computational complexity of the fundamental counting problems on planar graphs is addressed in [85], whereas the related problems in the context of spin glasses are studied in [11, 90]. However, insofar as the combinatorial problems on graphs are concerned, to the best of our knowledge, there has been no systematic account of the interplay between planarity and (uniform) sparseness, and this interplay's implications. Similarly, insofar as the discrete dynamical systems defined on planar network topologies are concerned, we are not aware of any attempts to address the interplay of planarity of the underlying graphs, possibly combined with other properties such as sparseness or bipartiteness, and various possible restrictions on the nature of individual agent behaviors, that is, the local update rules. For all these reasons, extending our results proven in the context of uniformly sparse graphs to planar graphs seems to be a potentially very rewarding future undertaking.

### 6.5.1 Some Open Problems on the Complexity of Counting

The discussion in the introductory part of this subsection addresses several promising directions for future work. The concrete open problems that our results in this Section leave unresolved include the following:

- Are the fundamental counting problems about symmetric (alternatively, monotone) Boolean S(y)DSs computationally tractable if the underlying graphs are restricted to those where every node has at most / exactly *two* (as opposed to three) neighbors? (What we know about the complexity of #CNF-SAT problems with the number of each variable's appearances limited to two suggests so; however, S(y)DSs with the appropriate node degree restrictions can be constructed in more general manners than the ways we have done it from the appropriate types of Boolean formulae in our hardness proofs in this Section.)

- What is the computational complexity of the problem #FP for simple threshold SDSs and SyDSs defined on the underlying graphs whose maximum node degree is 3 (alternatively, on the 3-regular graphs)? In particular, is there a *complexity gap* with respect to the allowed node degrees between the symmetric and the monotone Boolean S(y)DSs on the one hand, and the simple threshold ones (that are both monotone and symmetric), on the other?

- Are the counting problems #PRED and #ANC about *uniformly sparse* discrete Hopfield networks still hard, when the underlying weight matrices, instead of being *simple*, are required to have $w_{ii} = 1$ (or, more generally, some or all $w_{ii} > 0$) along the main diagonal? We note that the nodes having (one bit of) *memory* need not necessarily make the problems of interest just as hard as, or harder than, in the *memoryless* case.
- Which of the computational hardness results in this Section, established for the uniformly sparse underlying graphs – that in some cases may also be required to be bipartite, but are in general *non-planar* – carry over to the uniformly sparse *planar* graphs?

# 7 Summary and Future Work

Sections 4, 5 and 6 contain the author's main accomplishments in the realm of abstract discrete dynamical system models for the large scale distributed computing and large-scale MAS. In subsection 1.4, a number of possible generalizations of the already obtained results, as well as several further extensions and open problems, have been identified. These various possibilities for the future work discussed therein include proposed problems in the context of both the CA-inspired abstractions of distributed networked infrastructures and loosely coupled multi-agent systems, and the more concrete algorithmic problems pertaining to multi-agent coordination and autonomous action selection. For the sake of coherence, in this, final Section of this technical report we will elaborate chiefly on those research plans that lie within the realm of cellular and network automata models for distributed and multi-agent computing.

This Section is organized as follows. Subsection 7.1 briefly summarizes the main contributions of this technical report. It also outlines some of the main implications of our results in Sections 4, 5 and 6. Subsection 7.2 addresses some possible directions for the future work. In particular, it discusses several open problems on (S)CA, S(y)DSs and related models that were initially tentatively proposed in subsection 1.4. Subsection 7.3 addresses some more far-fetched, to be addressed farther into the future, research challenges. It focuses on the coordination problem in massively populated open distributed environments, and on different levels of abstraction at which the multi-agent coordination in general, and *consensus problems* such as the leader election and the coalition formation in particular, can be tackled. In that context, some ideas on how to model and solve these coordination problems at different levels of an individual agent's complexity will be outlined.

## 7.1 Report Summary

This technical report attempts to address some problems at the mathematical and computational foundations of large-scale multi-agent systems. The focus of our work is not on modeling, design or reasoning about a single agent that interacts with its outside world, pursues its tasks, etc. Rather, we focus on *collective dynamics* behavior of ensembles made of potentially a very large number of autonomously executing agents. Given that focus, we abstract an individual agent's behavior as a finite state machine. Most of our work focuses on (abstractions of) individual agents where an agent is, at each time step, in one of only two possible states, and the agent updates its state according to a fixed deterministic update rule. This is essentially the simplest possible nontrivial model of an individual agent. The emphasis of our work, however, is on how complex can the collective behavior of the entire multi-agent system get, depending on the details of (i) the exact interaction pattern among the agents, and (ii) the exact nature of the aforementioned simple update rules that capture the individual agents' behaviors. Thus, in essence, the emphasis of our work is on the coupling between *interaction* and *heterogeneity* among many simplistic reactive agents, and what are that coupling's implications insofar as the (un)predictability of the overall multi-agent system's behavior.

Our approach to studying the ensemble or collective dynamics of a multi-agent system is borrowed from physics and, more specifically, the study of complex dynamical systems. In that spirit, we formalize and then study the collective MAS behaviors in terms of *configuration space*

*properties* of the communicating finite state machine (CFSM) models such as the classical cellular automata, Sequential and Synchronous Dynamical Systems, and discrete Hopfield networks [192]. The kinds of questions we ask about these configuration spaces are inspired both by physics and nonlinear dynamics on one hand, and by the theory of distributed computing, on the other. In particular, we address the problems of existence and number of *stable* or *fixed point* configurations, cycle configurations, unreachable or *garden of Eden* configurations, and several other types of configurations of the appropriate CFSM-based discrete dynamical systems. A great deal of our results in that domain are *computational hardness* results: they impose lower bounds on predicting the corresponding collective dynamics properties of the actual multi-agent systems made of many interactive agents, where those MAS are formally abstracted as appropriate cellular or network automata models.

The model parameters with respect to which we attempt to measure the complexity of possible collective behaviors of an underlying MAS include (i) the communication model and its implications, (ii) the implications of the underlying communication network topology, and (iii) the diversity of the individual agent behaviors, that is, the implications of homogeneity vs. heterogeneity insofar as how individual agents update their states.

Section 4 concentrates on the model parameter (i); it studies classical parallel/synchronous and sequential/asynchronous cellular automata where, insofar as parameters (ii) and (iii) above are concerned, the default assumptions are complete uniformity of the inter-agent interactions (that is, the communication network topology), and complete homogeneity insofar as the individual agent behaviors are concerned. That Section is the only core part of this technical report where most results are providing explicit characterizations or predictions of what the possible short- and long-term dynamic behaviors of the underlying system can be. That is, the message of Section 4 is that, while the communication model may have considerable (and, at least in some cases, readily characterizable) implications for the overall system's behavior, the combination of network uniformity, agent homogeneity, and very restricted nature of the individual agent behaviors implies that, in principle, that behavior is, generally speaking, *predictable*.

In contrast, Sections 5 and 6 focus on the coupling of model parameters (ii) and (iii); they address generalizations of the classical cellular automata in which, when it comes to parameter (ii), some degree of nonuniformity is allowed in the underlying communication network topologies, and, moreover, insofar as parameter (iii) is concerned, some (however minimal) heterogeneity is allowed among the individual agents' behaviors. Given these emphases, it turns out that those two Sections mostly contain the computational hardness results. Those results identify a number of fundamental behavioral (that is, configuration space) properties of the underlying graph or network automata models that, under the usual assumptions in computational complexity theory, are *provably impossible to predict* within reasonable bounds on computational resources.

In fact, the only positive or "easiness" result in Section 6 is on the properties of the simple threshold cellular automata that are characterized by the aforementioned network uniformity and the homogeneity of individual behaviors. That result is included chiefly to provide a stark contrast with respect to what, at the structural level, appear to be rather slightly more complex (restricted) SDS and SyDS models, where only the very minimal degrees of network nonuniformity and individual agent behavior heterogeneity are allowed.

In summary, the details of both the underlying pattern of agent-to-agent interaction, that is, the network topology, and the exact behavior of each agent (and, in particular, whether all

agents behave the same or even the slightest individual differences are allowed) may have far-reaching and, in general, difficult to predict implications for the overall collective dynamics of a multi-agent system. The work summarized in this technical report is an early attempt to identify and quantify the possible impact of those two as well as some other modeling parameters. This work also constitutes a rigorous and systematic, although certainly far from complete, effort to establish at least some fragments of the boundary between those collective MAS behaviors that are amenable to analytic characterization and therefore prediction, and those other behaviors that are provably hard or impossible to predict, at least within a feasible amount of computational resources.

## 7.2 Some Ideas for Future Work on CA-based Models

Several concrete directions for extending the results presented in Section 4 have been outlined in subsections 1.4 and 4.5. A selected few will be discussed in some detail in the sequel.

As already emphasized in Section 4, one of the primary motivations for the comparative analysis of the parallel vs. sequential CA is to motivate the introduction and a detailed subsequent study of *genuinely asynchronous cellular automata* (ACA). In the ACA model, asynchrony applies to *both* local computations and inter-agent interaction (i.e., communication). While formally reasoning about and establishing formal properties of various subclasses of ACA can be readily predicted to be, in general, considerably more challenging than reasoning about and proving properties of the corresponding subclasses of parallel CA as well as sequential SCA, the resulting ACA abstraction still represents a descriptively very simple CFSM-based model. For instance, describing an ACA and its dynamic evolution is still likely to be far simpler than doing the same with other, better known abstractions for distributed computing such as, e.g., the *Petri Nets* model of C. A. Petri [143] or the *I/O Automata* of N. Lynch [115].

The questions to be asked insofar as the configuration space properties of various classes of ACA are concerned, in general, are of a similar flavor to the questions posed about the CA and the SCA/NICA models in Section 4. However, some care is advisable. For instance, it is not even meaningful to talk about a (fixed) *length* of a temporal cycle of an ACA. Given an outside observer with a discrete local clock ticking at a fixed rate, the same sequence of transitions in an ACA that leads to recurrent behavior can be interpreted as temporal cycles of different lengths, depending on the ratio of the observer's clock rate vs. all individual agents' local clock rates. In fact, once different clock rates are allowed, the very notion of *the length of a (temporal) cycle* becomes rather elusive. Therefore, while it is still reasonable to distinguish between *recurrent configurations* and *transient configurations* in ACA, and, moreover, between the fixed points and the cycling states among the recurrent configurations, talking about cycle or recurrence lengths the way it is done in the context of the parallel CA (or, for that matter, the SDSs and SyDSs) is meaningless.

Insofar as some prospective applications of the threshold SCA and ACA models are concerned, the desire is to formulate and subsequently solve at least some special cases of *distributed consensus problems* in the abstract setting provided by those cellular automata models. The two specific distributed consensus problems of our interest (see also subsubsection 1.3) are those of *leader election* and *group* or *coalition formation* [193]. Variants of these problems have already been formulated (but not completed or published as of yet) on some simple underlying network

topologies such as rings, stars and wheels. In these preliminary considerations, the cellular automata based model used is the *(fair)* SCA model. The real feast (and challenge), however, is to use ACA instead. Namely, it is well known that many variants of *distributed consensus* are provably more challenging in the asynchronous setting than when the global clock and synchronous communication among the agents are assumed (e.g., [116]). Comparing sequential and genuinely asynchronous CA in the context of distributed consensus problems appears an interesting direction to pursue. However, a thorough such inquiry lies beyond the scope of this technical report.

Another possible application of the cellular automata based formal models is in the general area of *formal verification* of distributed computing systems. For instance, the FP convergence can be seen as an abstraction of *self-stabilization* of a distributed protocol, and how long such a convergence takes in a given setting is a measure of how long the protocol takes before reaching an appropriate equilibrium or stationary state. On the other hand, sometimes reaching a FP is undesirable, since it means that the distributed system that is being abstracted as an appropriate kind of a graph automaton would get stuck. In many situations, it is desirable that some or all components of a system keep *doing something*. In the CFSM-based abstract models, this means that at least some among the nodes should be capable of changing their states. Thus, avoiding the convergence to a FP (if it is possible at all in a given CA or SCA or ACA setting) can be interpreted as an abstraction of the appropriate *liveness* properties.

Similarly, the garden of Eden configurations can be readily interpreted as *unreachable* system configurations: they can only appear as the starting configurations but, assuming the system starts from a non-GE configuration, it can never reach a GE [17, 133]. Determining whether configurations that satisfy certain properties are unreachable is an important problem in verification. If none of the appropriately defined *dangerous* states is reachable, i.e., if they can all be shown to be GEs, then the system can be considered to be safe. Hence, gardens of Eden can be formally related to the *safety* properties of distributed computing systems.

Given an opportunity in terms of time and resources, we would like to formalize and further exploit some of these verification-related ideas. The sequential and/or genuinely asynchronous cellular automata based *verification formalisms* are, in our view, not only a potentially interesting theoretical research project, but also a practically useful application of these abstract models to the analysis of large-scale distributed computing infrastructures.

Last but not least, all the results presented in this technical report are on the *deterministic* parallel and distributed computing models (and, in case of NICA, on their obvious nondeterministic extensions). Among the researchers in the areas of multi-agent systems, *ad hoc* and smart sensor networks, and other distributed infrastructures, however, the stochastic or probabilistic models are of a major interest.

One particular problem of a considerable interest is that of comparing deterministic vs. stochastic threshold CA and SCA, and, in particular, analyzing the *ergodic* properties of the stochastic models as the probability of random state flips approaches 0. It is known that the probabilistic MAJORITY CA with any fixed probability $p$ (where $1 > p > 0$) of a random state "flipping" are ergodic, i.e. that, whatever the initial configuration of such a CA may be, after a finite time they *forget* the starting configuration and become random, with the ratio of 0s and 1s according to probability $p$ [47]. In contrast, the deterministic MAJ (S)CA are characterized by good stability properties: as we have shown in Section 4, there are, in general, exponentially

many globally stable states (FPs), and, starting from a random state, it has been shown elsewhere that such a Fixed Point is reached relatively fast [68]. Exploring the ergodicity and mixing properties when the probability of random flips in a stochastic model approaches zero appears to be an interesting problem with considerable theoretical and practical implications.

### 7.2.1 Some Concrete Open Problems about S(y)DSs and (S)CA

In this subsubsection we briefly discuss some problems that are directly motivated by the work presented in Sections 4, 5 and 6.

Insofar as the future work along the general lines of Section 4, there are three particular directions among the open problems outlined earlier that would be nice to pursue in the intermediate and long terms. One is an appropriate completion of the results presented in Section 4. This would include a complete parametric characterization of the cycle configurations, fixed point configurations and transient configurations of the threshold (S)CA.

The second main direction is to study the *genuinely asynchronous* CA, and prove some theorems about the threshold ACA. The envisioned emphasis would be on comparing and contrasting the threshold ACA vs. the threshold CA, SCA and NICA. At the very least, our future goals would include providing some parametric characterizations of the ACA recurrent states, as well as some estimates on the convergence rates when the appropriate fairness and communication delay assumptions are required to hold.

We also intend to provide at least some applications of the ACA model. In particular, formulating and studying solvability of some consensus reaching type problems within the abstract asynchronous environment framework of ACA is one direction to consider, especially given our desire to apply the CA-like formalisms to modeling and analyzing *very large scale* multi-agent systems where the communication asynchrony usually reigns. We have offered some general discussion of these CA- and ACA-based verification formalisms in the introductory part of subsection 7.2.

The third main direction we envision is that of characterizing the agent coordination capabilities and mechanisms in the general setting of hierarchical autonomous agent models, where the hierarchy chiefly pertains to the limited memory storage of the agents. In that context, the Boolean CA- and/or GA-like models, where an agent has exactly one bit of memory for capturing the information about the current state of the world (as well as the world's history), is at one extreme of this hierarchy. Each agent being (equivalent to) an arbitrary Turing machine with its unbounded memory would be on the other extreme. Given the emphasis of our research on MAS with severe resource limitations [197, 199, 202, 203], only a portion of this whole spectrum of the agent memory models will be studied – namely, the part of the spectrum close to the CA/GA end where each individual agent possesses only a very limited amount of memory storage.

Insofar as some candidate research directions on the work on SDSs and SyDSs and their configuration space properties, we shall briefly outline the following two broad themes for future research.

One is the further comparison-and-contrast of the SDS and SyDS behaviors with respect to *connectivity* and other properties of the underlying graphs considered as the first-order parameters. In particular, we have established that most of the counting problems of interest in, say, monotone or symmetric S(y)DSs are provably computationally intractable in both uniformly

sparse (that is, bounded node degree) graphs (Section 6), and in the simplest possible graphs where one node is allowed to have $\Theta(n)$ neighbors (where $n$, as before, is the total number of nodes in the underlying graph of an SDS or SyDS); see Section 5. In particular, counting fixed points in star or wheel graphs is shown in [188] to be **#P**-complete for several restricted classes of the node update rules.

The results in [188, 196, 204, 206] about the uniformly sparse graphs and the star graphs have considerable implications both in computer science and beyond. Concretely, the SyDSs defined on wheel-like graphs, and those SyDSs' fixed points are highly reminiscent of the combinatorial problems underlying the *spin glasses* studied in statistical physics. More specifically, a spin glass model defined on a one-dimensional lattice of spins with the nearest-neighbor interactions and in the presence of an external magnetic field (which can be captured by the central node of the wheel) is quite similar to a Boolean-valued SyDS with (in general, non-monotone) linear threshold update rules defined over a wheel graph. The problems of finding the ground energy state and the degeneracy (i.e., how many distinct spin configurations actually yield that minimum energy) in the context of spin glasses can be seen as special cases of the Fixed Point Existence and Fixed Point Counting problems respectively for the appropriate types of cellular automata (regular Cartesian lattices, no external magnetic field) or SyDSs (e.g., the aforementioned 1-D spin glasses with an external magnetic field). For more on the spin glass models, their special case known as *the Ising model*, and the interesting algorithmic and computational complexity theoretic problems that arise in their context, we refer the reader to [11, 90, 220].

There are many other analogies between the models and interesting problems in statistical physics, and the CA and S(y)DS models and the problems about their various configuration space properties. Some of these analogies have been already addressed in the literature, mostly by the statistical physicists; some references include [58, 59, 71, 111, 220]. Further analysis and exploitation of the commonalities between the two domains appear to be a promising endeavor.

Last but not least, there is a natural way of unifying the models and problems studied in Section 4 of this technical report, and those in Sections 5 and 6. Namely, in terms of their communication model, clearly SDSs are not the most general model of sequential, let alone asynchronous, network automata. Therefore, the first extension is to keep the global clock, but allow more general sequences of node updates. This would naturally lead to *Sequential Graph Automata* as the most appropriate generalization of the SCA studied in Section 4. Similarly, one can define *Nondeterministic Interleavings Graph Automata* and *Fair Sequential Graph Automata* analogously to how the NICA and Fair SCA models are defined in Section 4. Studying properties of thus generalized SDSs and SGA, and comparing them with the sequential CA models from Section 4, would yield some insights into those aspects of these dynamical systems' behaviors that have nothing to do with the assumptions about the inter-agent communication, but, instead, would be directly attributable to *heterogeneity* of the agent behaviors and/or the *non-uniformity* of the interaction patterns among the agents. Thus, in a sense, the future work along those general lines would be orthogonal but also complementary to what has been addressed in Section 4 (see the discussion in subsection 1.1).

## 7.3 Coordination in Large-Scale Multi-Agent Domains

In this subsection, we make an attempt to relate the work on abstract parallel and distributed computing models, based on CA-like discrete dynamical systems, and the other main theme of our research discussed in Section 1, namely, the problem of coordination in large-scale multi-agent systems.

As briefly outlined in subsection 1.3, one of the central themes of our personal research interests in the context of large-scale multi-agent systems is the problem of inter-agent *coordination*. We have some interesting results in that context, chiefly on the problem of distributed coalition formation in locally constrained collaborative multi-agent environments [193, 199, 202]. The main contribution of that work is a fully decentralized, resource-aware, local algorithm for dynamic set partitioning of an agent ensemble into clique-like groups or coalitions. While the individual agents in the proposed *Maximal Clique based Distributed Coalition Formation* (MCDCF) algorithm are fairly simple – they use strictly local information and perform only very simple local computations, as well as store only a small amount of data about their nearby agents – these agents are nonetheless still considerably more complex than the extremely simplistic *reactive agents* as captured by the cellular and graph/network automata based MAS models.

Consensus reaching problems can be abstractly formulated in the general setting provided by the appropriate CA- or GA-like models, given that each agent is endowed with just a modest additional amount of knowledge (indeed, additional memory to store knowledge) about its neighbors.

One relevant question from both information-theoretic and practical design standpoints is, just *how much memory* is necessary for an agent to be able to determine, based on its local knowledge only, whether certain properties are satisfied. Appropriate abstract versions of *liveness* and *safety* properties are of our primary interest in that context. Another practical issue is, in what form should this information about other agents be stored? In the context of *totalistic* (that is, *symmetric*)[45] cellular or network automata such as those studied in Section 4 and much of Sections 5 and 6, a natural choice would be the appropriate counters with the purpose of keeping track, for each agent $x_i$, of *how many* nodes in the agent's neighborhood $N_i$ currently agree or disagree with $x_i$. Each agent may also need to keep record of how the value of this counter has been changing over time.

There are many important questions to ask once the agents are provided with even a very modest amount of additional memory in comparison to the binary-valued communicating FSM based models such as Boolean (S)CA and S(y)DSs. For instance, under what circumstances does $O(|N_i|)$ additional storage about the history of an agent and its neighborhood suffice for the agent to be able to determine its present or predict its future status insofar as the liveness and/or safety properties of interest are concerned? In what settings can appropriate versions of leader election

[45] The term *totalistic* (update rule or cellular automaton) was introduced by S. Wolfram; it has become a standard term for what we refer to as *symmetric* throughout the technical report. In the context of SDSs and SyDSs, the researchers who have introduced those two models, C. Barrett *et al.*, have been using *symmetric* the same way we use the term – in particular, our terminology and conventions regarding S(y)DSs, for the most part, closely follow those introduced by Barrett and collaborators in their foundational work on SDSs [12, 19, 21]. Last but not least, we remind the reader that both *totalistic* (in the context of CA) and *symmetric* (in the context of SDSs) refer to the nature of *update rules*; in contrast, in the Hopfield networks literature, the term *symmetric* pertains to the nature of the underlying weight matrix and, in particular, the update rule of a DHN with a symmetric weight matrix, in general, need not be a symmetric function as defined in the SDS literature and in Section 5 of this technical report.

and group formation be solved by such agents? How do the assumptions about the inter-agent communication (a)synchrony and possible random fluctuations (i.e., occasional random state flips of some agents) affect the agents' degree of effectiveness and/or the amount of necessary extra memory that they need in order to be able to reach consensus? We emphasize that, while more powerful than the *binary-valued*, 1-bit FSMs studied in the previous sections, these agents would be still very restricted in terms of their computational power, memory storage, and a simplistic abstract version of *bounded rationality*.

Our distributed coalition formation algorithm and its application to the large-scale MAS coordination assumes that each agent has sufficient memory to store a list of its neighbors, the neighborhood lists of those neighbors, and at least some of the potential coalitions, which are subsets of these neighborhood lists. Each agent also stores its program and the necessary auxiliary data needed for evaluating *quality* of various candidate coalitions. While, under appropriate assumptions discussed at length in [193, 197, 202], the overall amount of memory storage per agent is fairly small from the general computer science perspective, it is also considerably above the suggested *binary FSM + a few extra bits of memory (not accessible to other agents)* model alluded to earlier in this subsection. An interesting general problem to consider is, how do the ability and the effectiveness of an agent to locally act and/or coordinate with other agents change as the amount of knowledge that the agent is allowed to store about its environment increases?

One of the issues repeatedly alluded to in the introductory Section 1 of this technical report is that of the *granularity level* at which an autonomous agent and/or a multi-agent system made of autonomously executing agents is modeled. At the very fundamental level, this important issue, along the lines of the present brief discussion, can be viewed as the questions of *how much memory for storing knowledge about its environment* is an agent to be allowed to have, and how much of this stored knowledge is an agent allowed to communicate to other agents? Addressing these questions, and showing how is the ability to coordinate enhanced as the agents are allowed to remember and communicate more data about their environments, is a fundamental theoretical problem in distributed AI with considerable practical implications.

One of the long-term research plans is to address at least some of the aspects of this general problem, and hopefully provide some answers. We also envision providing some concrete and verifiable mechanisms for the multi-agent coordination at different levels of individual agents' computational power, chiefly in terms of how much knowledge are the agents allowed to store about the world. The proposed hierarchical approach to the correlation between the amount of an agent's memory and the richness of its internal structure on the one hand, and the capability of effective coordination on the other, is, however, an ambitious research project well beyond the scope of this technical report.

## Acknowledgments

The author gratefully acknowledges that none of his research achievements in general, and the doctoral dissertation that this technical report is based upon in particular, would have ever been possible without the immense and unconditional love and support from his family in Serbia. Therefore, the author would like to thank, first and foremost, his mother, sister and maternal grandmother from the very bottom of his heart. Second, the author is also greatly indebted to his relatives and his friends, old and new, for their encouragement and support throughout the long years spent in graduate school.

The author would like to express his gratitude to his dissertation advisor, professor Gul Agha, for continued guidance, patience and support since the fall of 2001. The author also acknowledges the support, feedback and word of advice from the rest of his dissertation committee. In particular, the author would like to recognize the most helpful continued support, encouragement and multi-faceted feedback from professors Michael Loui and Les Gasser, as well as the support and feedback from professors Sylvian Ray, Paul Schupp and Mahesh Viswanathan.

In addition, the author would like to acknowledge a number of other professors and/or research scientists whose feedback, criticism, support and encouragement helped a great deal in (i) persevering and (ii) making the author's doctoral dissertation, and consequently also this report, much better than they would have been otherwise. Among many people who have contributed to the author's growth as a scholar and a researcher, yet were not among his dissertation defense committee, the following individuals have had a particularly prominent role: Tom Armstrong, David Berg, Nora Few, Jim Greenberg, Alfred Hubler, Harry Hunt, Madhav Marathe, David Padua, Slobodan Petrovich, Robert Rasera, Joe Rosenblatt, Ruben Rostamian and Alan Sherman.

The author would also like to acknowledge his colleagues from professor Agha's Open Systems Laboratory (OSL) at University of Illinois – especially those who were involved in the TASK research project. Among them, special thanks go to Reza Ziaei, Wooyoung Kim and Myeong-Wuk Jang. The author would also like to thank Amr Ahmed, Tom Brown, Po-Hao Chang, Liping Chen, Joshua Chia, Chris Devaraj, MyungJoo Ham, Nadeem Jamali, Young-Min Kwon, Timo Latvala, Soham Mazumdar, Kirill Mechitov, Abhilash Patel, Smitha Reddy, Koushik Sen, Sudarshan Srinivasan, Sameer Sundresh, Prasannaa Thati, and Abhay Vardhan.

It has been a true blessing to have an opportunity to interact with, as well as get a good advice from, a number of other faculty, staff, visiting scholars and graduate students at UIUC over the course of a decade the author has spent there. In particular, the following faculty (and, where applicable, their graduate students) are gratefully acknowledged: Tom Anastasio, Geneva Belford, Jeff Erickson, Sariel Har-Peled, Michael Heath, Stephen Levinson, Klara Nahrstedt, Madhu Parthasarathy, Constantine Polychronopoulos, Sylvian Ray, Edward Reingold, Grigore Rosu and Dan Roth.

Last but not least, the author is greatly indebted to Alan Sherman and Paul Schupp for introducing him to the beautiful world of theory of computing during his undergraduate and the early stages of his graduate studies, respectively, and to Gul Agha and Les Gasser for stimulating his interest in the exciting areas of multi-agent systems and distributed AI. Insofar as the final stages of the author's doctoral studies are concerned, the constructive criticism, feedback, support and word of advice from Michael Loui were absolutely essential; hence, professor Loui's very special role in author's successful completion of doctoral dissertation is appreciatively acknowledged.

The research summarized in author's doctoral dissertation and this technical report was supported in part by the DARPA IPTO TASK program, contract # F30602-00-2-0586, and the ONR MURI program, contract # N00014-02-1-0715.

## References


[1] M. Ajtai, J. Komlos, E. Szemeredi. "An $O(n \log n)$ sorting network", *Combinatorica*, vol. 3, pp. 1 – 19, 1983

[2] N. Alon, R. B. Boppana. "The monotone circuit complexity of Boolean functions", *Combinatorica*, vol. 7, pp. 1 – 22, 1987

[3] M. Anthony. "Threshold Functions, Decision Lists, and the Representation of Boolean Functions", NeuroCOLT Techn. Report Series (NC-TR-96-028), January 1996

[4] S. Amoroso, Y. Patt. "Decision procedures for surjectivity and injectivity of parallel maps for tessellation structures", *Journal of Computer and System Sciences* (JCSS), vol. 6, pp. 448 – 464, 1972

[5] S. Arnborg, J. Lagergren, D. Seese. "Easy problems for tree-decomposable graphs", *Journal of Algorithms*, vol. 12, 1991, pp. 308 – 340

[6] S. Arora, Y. Rabani, U. Vazirani. "Simulating quadratic dynamical systems is **PSPACE**-complete," *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing* (STOC'94), pp. 459 – 467, Montreal, Canada, May 1994

[7] J. Arpe, A. Jakoby, M. Liskiewicz. "One-Way Communication Complexity of Symmetric Boolean Functions", *Electronic Colloquium on Computational Complexity*, Tech. Report ECCC-TR03-083, 2003

[8] W. Ross Ashby. *"Design for a Brain"*, Wiley, 1960

[9] N. M. Avouris, L. Gasser (eds.) *"Distributed Artificial Intelligence: Theory and Praxis"*, European Courses on Computer & Information Sciences, vol. 5, Kluwer Academic Publ., 1992

[10] R. J. Bagley, L. Glass. "Counting and Classifying Attractors in High Dimensional Dynamical Systems", *Journal of Theoretical Biology*, vol. 183, pp. 269 – 284, 1996

[11] F. Barahona. "On the computational complexity of Ising spin glass models", *Journal of Physics A: Mathematical and General*, vol. 15, pp. 3241 – 3253, 1982

[12] C. Barrett, B. Bush, S. Kopp, H. Mortveit, C. Reidys. "Sequential Dynamical Systems and Applications to Simulations", Technical Report, Los Alamos National Laboratory, Los Alamos, New Mexico, September 1999

[13] C. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, R. E. Stearns. "Dichotomy Results for Sequential Dynamical Systems", Los Alamos National Laboratory Report, LA-UR-00-5984, Los Alamos, New Mexico, 2000

[14] C. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, R. E. Stearns. "Predecessor and Permutation Existence Problems for Sequential Dynamical Systems", Los Alamos National Laboratory Report, LA-UR-01-668, 2001

[15] C. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, R. E. Stearns. "On Some Special Classes of Sequential Dynamical Systems", *Annals of Combinatorics*, vol. 7, pp. 381 – 408, 2002

[16] C. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, R. E. Stearns. "Reachability problems for sequential dynamical systems with threshold functions", *Theoretical Computer Science*, vol. 295, issues 1-3, pp. 41 – 64, February 2003

[17] C. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, R. E. Stearns, P. T. Tosic. "Gardens of Eden and Fixed Points in Sequential Dynamical Systems", Proceedings of *Discrete Models: Combinatorics, Computation, and Geometry* (DM-CCG), in *Discrete Mathematics and Theoretical Computer Science* (DMTCS), conference vol. AA, pp. 95 – 110, 2001

[18] C. Barrett, M. Marathe, H. Mortveit, C. Reidys, J. Smith, S.S. Ravi. "AdhopNET: Advanced Simulation-based Analysis of Ad-Hoc Networks", Los Alamos Unclassified Internal Report, 2000

[19] C. Barrett, H. Mortveit, C. Reidys. "Elements of a theory of simulation II: sequential dynamical systems" *Applied Mathematics and Computation*, vol. 107 (2-3), pp. 121 – 136, 2000

[20] C. Barrett, H. Mortveit, C. Reidys. "Elements of a theory of computer simulation III: equivalence of SDS", *Applied Mathematics and Computation*, vol. 122, pp. 325 – 340, 2001

[21] C. Barrett, C. Reidys. "Elements of a theory of computer simulation I: sequential CA over random graphs" *Applied Mathematics and Computation*, vol. 98, pp. 241 – 259, 1999

[22] C. Barrett, M. Wolinsky, M. Olesen. "Emergent Local Properties in Particle Hopping Traffic Simulations", in *Proc. Traffic and Granular Flow*, Los Alamos Unclassified Internal Report, LA-UR-95-4368, Los Alamos, New Mexico, 1995

[23] M. Bauland, E. Bohler, N. Creignou, S. Reith, H. Schnoor, H. Vollmer. "Quantified Constraints: The Complexity of Decision and Counting for Bounded Alternation", *Electronic Colloquium on Computational Complexity*, ECCC-TR05-24, 2005

[24] R. Beckman *et. al.* "TRANSIMS – Release 1.0 – The Dallas-Forth Worth case study", Tech. Report LA-UR-97-4502, Los Alamos National Laboratory, Los Alamos, New Mexico, 1999

[25] E. Behrends. *"Introduction to Markov Chains with Special Emphasis on Rapid Mixing"*, Advanced Lectures in Mathematics, Vieweg, 2000

[26] F. Blanchard. "Cellular Automata and Transducers: A Topological View", in *"Cellular Automata, Dynamical Systems and Neural Networks"*, Mathematics and Its Application series vol. 282, E. Goles and S. Martinez (eds.), pp. 1 – 22, Kluwer Academic Publshers, 1994

[27] F. Blanchard, P. Kurka, A. Maass. "Topological and measure-theoretic properties of one-dimensional cellular automata", *Proceedings of the workshop on Lattice dynamics*, pp. 86 – 99, Elsevier, Paris, 1997

[28] A. Blum, C. Burch, J. Langford. "Learning monotone Boolean functions", *Proceedings of IEEE Conference on Foundations of Computer Science* (FOCS'98), pp. 408 – 415, 1998

[29] H. L. Bodlaender. "NC Algorithms for Graphs with Bounded Treewidth", in *Proceedings of the Workshop on Graph Theoretic Concepts in Computer Science*, pp. 1 – 10, 1988

[30] A. Bogdanov, L. Trevisan. "Average-Case Complexity", *Electronic Colloquium on Computational Complexity*, Tech. Report No. 73 (ECCC-TR06-73), 2006

[31] R. B. Boppana, M. Sipser. "The complexity of finite functions", in *Handbook of Theoretical Computer Science, Vol. A: Algorithms and Complexity*, J. van Leeuwen (ed.), Elsevier, Amsterdam, pp. 757 – 800, 1992

[32] M. Bordewich. "The Complexity of Counting and Randomised Approximation", Ph.D. dissertation, University of Oxford, 2003

[33] S. Buss, C. Papadimitriou, J. Tsitsiklis. "On the predictability of coupled automata: An allegory about Chaos", *Complex Systems*, vol. 1 (5), pp. 525 – 539, 1991 Preliminary version appeared in *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science* (FOCS'90), October 1990

[34] G. Cattaneo, E. Formenti, L. Margara. "Topological Definitions of Deterministic Chaos: Applications to Cellular Automata Dynamics", in *"Cellular Automata: A Parallel Model"*, M. Delorme, J. Mazoyer (eds.), pp. 213 – 262, Kluwer Academic Publishers, 1999

[35] G. Cattaneo, M. Finelli, G. Manzini, L. Margara. "Ergodicity, transitivity and regularity for linear cellular automata over $Z_m$", *Theoretical Computer Science*, vol. 233, pp. 147 – 164, 2000

[36] G. Cattaneo, M. Finelli, L. Margara. "Investigating topological chaos by elementary cellular automata dynamics", *Theoretical Computer Science*, vol. 244 (1-2), pp. 219 – 241, 2000

[37] B. Codenotti, L. Margara. "Transitive Cellular Automata are Sensitive", *The American Mathematical Monthly*, vol. 103 (1), pp. 58 – 62, 1996

[38] T. H. Cormen, C. E. Leiserson, R. L. Rivest. *"Introduction to Algorithms"*, MIT Press, 1990

[39] K. Culik, L. P. Hurd, S. Yu. "Computation theoretic aspects of cellular automata", *Physica D*, vol. 45 (1-3), pp. 357 – 378, 1990

[40] K. Culik, J. Pachl, S. Yu. "On the limit sets of cellular automata", *SIAM Journal of Computing*, vol. 18(4), pp. 831 – 842, 1989

[41] K. Culik, S. Yu. "Undecidability of CA classification schemes", *Complex Systems*, vol. 2 (2), pp. 177 – 190, 1988

[42] I. Czaja, R. J. van Glabbek, U. Goltz. "Interleaving semantics and action refinement with atomic choice", in *"Advances in Petri Nets"*, G. Rozenberg (ed.), Lecture Notes in Computer Science (LNCS) vol. 609, Springer-Verlag, 1992

[43] P. Dagum, M. Luby. "Approximating probabilistic inference in Bayesian belief networks is **NP**-hard", *Artificial Intelligence*, vol. 60, pp. 141 – 153, 1993

[44] M. Delorme, J. Mazoyer. "An overview of language recognition on one-dimensional cellular automata", in *"Semigroups, Automata and Languages"*, J. Almeida (ed.), pp. 85 – 100, World Scientific, 1996

[45] M. Delorme, J. Mazoyer. "Cellular Automata as Language Recognizers", in *" Cellular Automata: A Parallel Model"*, M. Delorme, J. Mazoyer (eds.), pp. 153 – 179, Kluwer Academic Publishers, 1999

[46] M. Dietzfelbinger, M. Kutylowski, R. Reischuk. "Feasible Time-Optimal Algorithms for Boolean Functions on Exclusive-Write Parallel Random-Access Machines", *SIAM Journal of Computing*, vol. 25 (6), pp. 1196 – 1230, 1996

[47] R.L. Dobrushin, V.I. Kryukov, A.L. Toom (eds.). *"Stochastic Cellular Systems: Ergodicity, Memory, Morphogenesis"*, (Nonlinear Science: Theory and Applications), Manchester Univ. Press, 1991

[48] B. Durand. "Inversion of 2D cellular automata: some complexity results", *Theoretical Computer Science*, vol. 134 (2) , pp. 387 – 401, November 1994

[49] B. Durand. "A random **NP**-complete problem for inversion of 2D cellular automata", *Theoretical Computer Science*, vol. 148 (1) , pp. 19 – 32, August 1995

[50] B. Durand. "Global properties of 2D cellular automata", in E. Goles, S. Martinez (eds.), *"Cellular Automata and Complex Systems"*, Kluwer, Dordrecht, 1998

[51] C. Dyer. "One-way bounded cellular automata", *Information and Control*, vol. 44, pp. 54 – 69, 1980

[52] M. E. Dyer, A. M. Frieze. "Planar 3DM is **NP**-Complete", *Journal of Algorithms*, vol. 7, No. 2, pp. 174 – 184, March 1986

[53] M. Dyer, L. A. Goldberg, C. Greenhill, M. Jerrum. "The Relative Complexity of Approximate Counting Problems", *Algorithmica*, vol. 38, pp. 471 – 500, 2004

[54] P. Floreen. "A short introduction to neural associative memories", *Bulletin of European Association for Theoretical Computer Science*, vol. 51, pp. 236 – 245, October 1993

[55] P. Floreen, P. Orponen. "On the Computational Complexity of Analyzing Hopfield Nets", *Complex Systems* vol. 3, pp. 577 – 587, 1989

[56] P. Floreen, P. Orponen. "Attraction radii in binary Hopfield nets are hard to compute", *Neural Computations* vol. 5, pp. 812 – 821, 1993

[57] P. Floreen, P. Orponen. "Complexity Issues in Discrete Hopfield Networks", *NeuroCOLT Technical Report Series*, NC-TR-94-009, October 1994

[58] P. Gacs. *"Reliable computation with cellular automata"*, ACM Press, New York, NY, 1983

[59] P. Gacs. "Reliable computation with cellular automata", *Journal of Computer and System Sciences*, vol. 32, pp. 15 – 78, 1988

[60] P. Gacs. "Deterministic computations whose history is independent of the order of asynchronous updating", Technical Report, Computer Science Department, Boston University, 1997

[61] P. Gacs. "Reliable Cellular Automata with Self-Organization", *Journal of Statistical Physics*, vol. 103, No. 1-2, pp. 45 – 267, April 2001

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

[63] M. Garzon. *"Models of Massive Parallelism: Analysis of Cellular Automata and Neural Networks"*, Springer, 1995

[64] R. J. van Glabbek, U. Goltz. "Equivalences and refinement", *Proceedings of the LITP Spring School of Theoretical Computer Science*, La Roche-Posay, France (I. Guessarian, ed.), LNCS 469, Springer-Verlag 1990

[65] M. Gouda, C. Chang. "Proving Liveness for Networks of Communicating Finite State Machines." *ACM Transactions on Programming Languages and Systems* (TOPLAS'86), vol. 8 (1), pp. 154 – 182, 1986

[66] M. Goldmann, M. Karpinski. "Simulating threshold circuits by majority circuits", *SIAM Journal of Computing*, vol. 27 (1), pp. 230 – 246, 1998

[67] M. Goldmann, J. Hastad, A. Razborov. "Majority gates vs. general weighted threshold gates", *Computational Complexity*, vol. 2, pp. 277 – 300, 1992

[68] E. Goles, S. Martinez. *"Neural and Automata Networks: Dynamical Behavior and Applications"*, Mathematics and Its Applications series (vol. 58), Kluwer, 1990

[69] E. Goles, S. Martinez (eds.). *"Cellular Automata, Dynamical Systems and Neural Networks"*, Mathematics and Its Applications series (vol. 282), Kluwer, 1994

[70] E. Goles, S. Martinez (eds.). *"Cellular Automata and Complex Systems"*, Nonlinear Phenomena and Complex Systems series, Kluwer, 1999

[71] L. Gray. "The behavior of processes with statistical mechanical properties", in *Percolation Theory and Ergodic Theory of Infinite Particle Systems*, H. Kesten (ed.), Springer-Verlag, New York, 1987

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

[73] C. Greenhill. "The Complexity of Counting Colourings and Independent Sets in Sparse Graphs and Hypergraphs", *Computational Complexity*, vol. 9, pp. 52 – 72, 2000

[74] R. Gunther, B. Schapiro, P. Wagner. "Complex Systems, Complexity Measures, Grammars and Model-Inferring", *Chaos, Solitons and Fractals* vol. 4 (5), pp. 635 – 651, 1994

[75] H. Gutowitz (Editor). *"Cellular Automata: Theory and Experiment"*, North Holland, 1989

[76] M. H. Hassoun. *"Fundamentals of Artificial Neural Networks"*, The MIT Press, Cambridge, MA, 1995

[77] M.H. Hassoun, P. B. Watta. "Alternatives to Energy Function-Based Analysis of Recurrent Neural Networks", Tech. Report, Dept. of Electrical & Computer Engineering, Wayne State Univesity, Detroit, Michigan

[78] J. Hastad. "On the size of weights for threshold gates", *SIAM Journal of Discrete Mathematics*, vol. 7 (3), pp. 484 – 492, 1994

[79] M. Hermann, P. G. Kolaitis. "Computational Complexity of Simultaneous Elementary Matching Problems", *Journal of Automated Reasoning*, vol 23, pp. 107 - 136, 1999

[80] C. A. R. Hoare. *"Communicating Sequential Processes"*, Prentice Hall, 1985

[81] J. J. Hopfield. "Neural networks and physical systems with emergent collective computational abilities", *Proceedings of the National Academy of Sciences* (USA), vol. 79, pp. 2554 – 2558, 1982

[82] J. J. Hopfield, D. W. Tank. "Neural computation of decisions in optimization problems", *Biological Cybernetics*, vol. 52, pp. 141 – 152, 1985

[83] B. Huberman, N. Glance. "Evolutionary games and computer simulations" *Proceedings of the National Academy of Sciences* (USA), vol. 90, pp. 7716 – 7718, 1993

[84] H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, S. S. Ravi, D. J. Rosenkrantz, R. E. Stearns. "Designing Approximation Schemes using L-reductions" *Proceedings of the 14th Conference on Foundations of Software Technology and Theoretical Computer Science* (FST & TCS'94), pp. 342 – 353, Madras, India, December 1994

[85] H. B. Hunt, M. V. Marathe, V. Radhakrishnan, R. E. Stearns. "The complexity of planar counting problems", *SIAM Journal of Computing*, vol. 27, pp. 1142 – 1167, 1998

[86] L.P. Hurd. "On invertible cellular automata", *Complex Systems*, vol. 1 (1), pp. 69 – 80, 1987

[87] O. Ibarra, T. Jiang. "On one-way cellular arrays", *SIAM Journal of Computing*, vol. 16, pp. 1135 – 1154, 1987

[88] O. Ibarra. "Computational Complexity of Cellular Automata: An Overview", in *"Cellular Automata: A Parallel Model"*, M. Delorme, J. Mazoyer (eds.), pp. 181 – 198, Kluwer Academic Publishers, 1999

[89] T. E. Ingerson, R. L. Buvel. "Structure in asynchronous cellular automata", *Physica D: Nonlinear Phenomena*, Volume 10, Issues 1-2, pp. 59 – 68, January 1984

[90] S. Istrail. "Statistical Mechanics, Three-Dimensionality and NP-completeness: I. Universality of Intracatability for the Partition Function of the Ising Model Across Non-Planar Lattices (Extended Abstract)", *Proceedings of the 32nd ACM Symposium on Theory of Computing* (STOC '00), Portland, Oregon, pp. 87 – 96, 2000

[91] M. Jerrum. "Two-dimensional monomer-dimer systems are computationally intractable", *Journal of Statistical Physics*, vol. 48, pp. 121 – 134, 1987. Erratum in vol. 59, pp. 1087 – 1088, 1990

[92] M. Jerrum, A. Sinclair. "Approximating the permanent", *SIAM Journal of Computing*, vol. 18, pp. 1149 – 1178, 1989

[93] M. Jerrum, A. Sinclair. "Polynomial-time approximation algorithms for the Ising model", *SIAM Journal of Computing*, vol. 22, pp. 1087 – 1116, 1993

[94] M. R. Jerrum, L. G. Valiant, V. V. Vazirani. "Random generation of combinatorial structures from a uniform distribution", *Theoretical Computer Science*, vol. 43, pp. 169 – 188, 1986

[95] J. S. Judd. *"Neural Network Design and the Complexity of Learning"*, The MIT Press, Cambridge, Massachusetts, 1990

[96] Y. Kanada. "Asynchronous 1D Cellular Automata and the Effects of Fluctuations and Randomness", Technical Report, Tsukuba Research Center Real World Computing Partnership, Tsukuba, Japan

[97] J. Kari. "Reversibility of 2D cellular automata is undecidable", *Physica D*, vol. 45, pp. 379 – 385, 1990

[98] J. Kari. "Reversibility and surjectivity problems of cellular automata", *Journal of Computer and System Sciences*, vol. 48, pp. 149 – 182, 1994

[99] J. Kari. "Theory of cellular automata: A survey", *Theoretical Computer Science*, vol. 334, pp. 3 – 33, 2005

[100] R. Karp, M. Luby. "Monte-Carlo algorithms for enumeration and reliability problems", *IEEE Symposium on Foundations of Computer Science*, No. 24, pp. 56 – 64, 1983

[101] R. Karp, M. Luby. "Monte Carlo algorithms for the planar multiterminal network reliability problem", *Journal of Complexity*, vol. 1, pp. 45 – 64, 1985

[102] R. Karp, M. Luby, N. Madras. "Monte Carlo approximation algoithms for enumeration problems", *Journal of Algorithms*, vol. 10, pp. 429 – 448, 1989

[103] S. A. Kauffman. "Metabolic stability and epigenesis in randomly connected nets", *Journal of Theoretical Biology*, vol. 22, pp. 437 – 467, 1969

[104] S. A. Kauffman. "Emergent properties in random complex automata", *Physica D: Nonlinear Phenomena*, Volume 10, Issues 1-2, pp. 59 – 68, January 1984

[105] S. A. Kauffman. *"The Origins of Order: Self-Organization and Selection in Evolution"*, Oxford University Press, Oxford, UK, 1993

[106] M. Kaufman, J. Urbain, R. Thomas. "Towards a logical analysis of immune response", *Journal of Theoretical Biology*, vol. 114, pp. 527 – 561, 1985

[107] Z. Kohavi. *"Switching and Finite Automata Theory"*, McGraw-Hill Book Co., New York, NY, 1970

[108] A. D. Korshunov. "Monotone Boolean functions", *Russian Mathematical Surveys*, vol. 58, 2003

[109] P. Kurka. "Languages, equicontinuity and attractors in cellular automata", *Ergodic Theory and Dynamical Systems*, vol. 17, pp. 417 – 433, 1997

[110] R. Laubenbacher, B. Pareigis. "Finite Dynamical Systems" Technical report, Department of Mathematical Sciences, New Mexico State University, Las Cruces, 2000

[111] J. L. Lebowitz, C. Maes, E. R. Speer. "Statistical Mechanics of Probabilistic Cellular Automata", *Journal of Statistical Physics*, vol. 59, #1-2, pp. 117 – 170, 1990

[112] K. Lindgren, M. G. Nordahl. "Universal Computation in Simple One-Dimensional Cellular Automata", *Complex Systems*, vol. 4, pp. 299 – 318, 1990

[113] R. F. Liu, C. C. Chen. "Analytic Proof of the Attractors of a Class of Cellular Automata", LANL online archives, arXiv:nlin.CG/0209005 v2, December 2003

[114] M. Luby, E. Vigoda. "Fast convergence of the Glauber dynamics for sampling independent sets", *Random Structures Algorithms,* vol. 15, pp. 229 – 241, 1999

[115] N. Lynch, M. Tuttle. "An Introduction to Input/Output automata", *CWI-Quarterly*, vol. 2 (3), pp. 219 – 246, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands, September 1999. Also: Technical Memo MIT/LCS/TM-373, Laboratory for Computer Science, MIT

[116] N. Lynch. *"Distributed Algorithms"*, Morgan Kaufmann Publ., Wonderland, 1996

[117] G. Manzini, L. Margara. "Invertible Linear Cellular Automata over $\mathbb{Z}_m$: Algorithmic and Dynamical Aspects", *Journal of Computer and System Sciences* (JCSS), vol. 56, pp. 60 – 67, 1998

[118] G. Manzini. "Characterization of Sensitive Linear Cellular Automata with Respect to the Counting Distance", *Proceedings of Mathematical Foundations of Computer Science* (MFCS'98), Lecture Notes in Computer Science series, vol. 1450, pp. 825 – 833, Springer, 1998

[119] B. Martin. "A Geometrical Hierarchy of Graphs via Cellular Automata", *Proceedings of Mathematical Foundations of Computer Science* (MFCS'98) – Satellite Workshop on Cellular Automata, Brno, Czech Republic, August 1998

[120] M.V. Marathe, H.B. Hunt III, D.J. Rosenkrantz, R.E. Stearns. "Theory of periodically specified problems: Complexity and Approximability" *Proceedings of the 13th IEEE Conference on Computational Complexity*, Buffalo, NY, June, 1998

[121] L. Margara. *"Cellular Automata and Chaos"*, PhD dissertation, Universita di Pisa (Italy), Dipartimento di Informatica, 1995

[122] R. Milner. *"A Calculus of Communicating Systems"*, Lecture Notes in Computer Science (LNCS) series, Springer-Verlag, 1980

[123] R. Milner. "Calculi for synchrony and asynchrony", *Theoretical Computer Science*, vol. 25, pp. 267 – 310, 1983

[124] R. Milner. *"Communication and Concurrency"*, Prentice-Hall, 1989

[125] M. Mitchell. "Computation in Cellular Automata: A Selected Review", in T. Gramms, S. Bornholdt, M. Gross, M. Mitchell, T. Pellizzari (eds.), *"Nonstandard Computation"*, pp. 95 – 140, Weinheim: VCH Verlagsgesellschaft, 1998

[126] P. J. Modi, H. Jung, W. Shen, M. Tambe, S. Kulkarni. "A Dynamic Distributed Constraint Satisfaction Approach to Resource Allocation", in *Proceedings of The Seventh International Conference on Principles and Practice of Constraint Programming*, 2001

[127] P. J. Modi, H. Jung, W. Shen. "Distributed Resource Allocation: Formalization, Complexity Results and Mappings to Distributed CSPs", Tech. Report, November 2002

[128] P. J. Modi, W. Shen, M. Tambe, M. Yokoo. "An asynchronous complete method for distributed constraint optimization", in *Proceedings of The Second International Conference on Autonomous Agents and Multi-Agent Systems* (AAMAS-03), Melbourne, Australia, July 14-18, 2003

[129] C. Moore. "Unpredictability and undecidability in dynamical systems" *Physical Review Letters*, vol. 64 (20), pp 2354 – 2357, 1990

[130] C. Moore. "Generalized shifts: unpredictability and undecidability in dynamical systems" *Nonlinearity*, vol. 4, pp. 199 – 230, 1991

[131] C. Moore. "Quasi-linear cellular automata", *Physica D*, vol. 103 (Proceedings of the International Workshop on Lattice Dynamics '97), pp. 100 – 132, 1997

[132] H. Mortveit, C. Reidys. "Discrete sequential dynamical systems", *Discrete Mathematics*, vol. 226 , Issue 1-3, pp. 281 – 295, 2001

[133] J. Myhill. "The converse of Moore's Garden-of-Eden theorem", *Proceedings of the American Mathematical Society*, vol. 14, pp. 685 – 686, 1963

[134] J. von Neumann. *"Theory of Self-Reproducing Automata"*, edited and completed by A. W. Burks, University of Illinois Press, Urbana, Illinois, 1966

[135] C. Nichitiu, E. Remila. "Simulations of Graph Automata" *Proceedings of MFCS'98 – Satellite Workshop on Cellular Automata*, Brno, Czech Republic, August 1998

[136] P. Orponen. "On the computational complexity of discrete Hopfield nets", *Proceedings of the 20th International Colloqium on Automata, Languages and Programming* (ICALP '93), Springer LNCS series, vol. 700, pp. 215 – 226, 1993

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

[138] P. Orponen. "The computational power of discrete Hopfield networks with hidden units", *Neural Computation*, vol. 8 (2), pp. 403 – 415, 1996

[139] P. Orponen. "Computing with truly asynchronous threshold logic networks", *Theoretical Computer Science*, vol. 174 (1-2), pp. 123 – 136, 1997

[140] C. Papadimitriou. *"Computational Complexity"*, Addison-Wesley, Reading, Massachusetts, 1994

[141] I. Parberry. "A primer on the complexity theory of neural networks", in *Formal Techinques in AI: A Sourcebook*, R. B. Banerji (ed.), North-Holland, Amsterdam, 1990

[142] I. Parberry. *"Circuit Complexity and Neural Networks",* The MIT Press, Cambridge, Massachusetts, 1994

[143] C. A. Petri. *"Kommunikation mit Automaten" (Communicating with automata)*, PhD dissertation, Technical University Darmstadt, Germany, 1962

[144] D. Poole. "On the hardness of approximate reasoning", *Proceedings of the International Joint Conference on Artificial Intelligence* (IJCAI'93), pp. 607 – 612, August 1993

[145] Y. Rabinovich, A. Sinclair, A. Wigderson. "Quadratic dynamical systems", *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science* (FOCS'92), pp. 304 – 313, Pittsburgh, October 1992

[146] A. S. Rao, M. P. Georgeff. "BDI Agents: From Theory to Practice", *Proceedings of the First International Conference on Multi-Agent Systems* (ICMAS'95), San Francisco, USA, 1995

[147] A. Razborov. "Lower bounds for the monotone complexity of some Boolean functions", *Doklady Akademii Nauk*, USSR, vol. 281, pp. 791 – 801, 1985. English translation in *Soviet Mathematical Doklady,* vol. 31, pp. 354 – 357, 1985

[148] C. Reidys. "On Acyclic Orientations & Sequential Dynamical Systems", Los Alamos National Laboratory Report, LA-UR-01-598, 2001

[149] J. C. Reynolds. *"Theories of Programming Languages"*, Cambridge University Press, 1998

[150] D. Richardson. "Tessellations with local transformations" *Journal of Computer and System Sciences* (JCSS), vol. 6, pp. 373 – 388, 1972

[151] N. Robertson, P. D. Seymour. "Graph Minors II, Algorithmic Aspects of Tree-Width", *Journal of Algorithms*, vol. 7, pp. 309 – 322, 1986

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

[153] Zs. Roka. "One-way cellular automata on Cayley graphs" *Theoretical Computer Science*, vol. 132 (1-2), pp. 259 – 290, September 1994

[154] Zs. Roka. "The Firing Squad Synchronization Problem on Cayley Graphs", *Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science (LNCS), vol. 969, Springer, 1995

[155] S. J. Rosenschein, L. P. Kaelbling. "A Situated View of Presentation and Control", *Artificial Intelligence*, vol. 73 (1–2), pp. 149 – 173, February 1995

[156] J. Rosenschein, G. Zlotkin. *"Rules of Encounter: Designing Conventions for Automated Negotiations among Computers",* The MIT Press, Cambridge, Massachusetts, 1994

[157] D. Roth. "On the Hardness of Approximate Reasoning", *Artificial Intelligence*, vol. 82, pp. 273 – 302, 1996

[158] S. Russell, P. Norvig. *"Artificial Intelligence: A Modern Approach",* 2nd ed., Prentice Hall Series in Artificial Intelligence, 2003

[159] B. Samuelsson, C. Troein. "Superpolynomial Growth in the Number of Attractors in Kauffman Networks", Workshop on Random Geometry, Krakow, Poland, 2003

[160] P. Sarkar. "A Brief History of Cellular Automata", *ACM Computing Surveys*, vol. 32 (1), March 2000

[161] T. Schaefer. "The Complexity of Satisfiability Problems", *Proceedings of the 10th ACM Symposium on Theory of Computing* (STOC'78), pp. 216 – 226, 1978

[162] C. Schittenkopf, G. Deco, W. Brauer. "Finite automata-models for the investigation of dynamical systems", *Information Processing Letters*, vol. 63 (3), pp. 137 – 141, August 1997

[163] R. Servedio. "Monotone Boolean Formulas Can Approximate Monotone Linear Threshold Functions", Technical Report, Columbia University, New York, July 2003

[164] R. Sethi. *"Programming Languages: Concepts and Constructs"*, 2nd ed., Addison-Wesley, 1996

[165] I. Shmulevich, A.D. Korshunov, J. Astola. "Almost all monotone Boolean funcions are polynomially learnable using membership queries", *Information Processing Letters*, vol. 79 (5), pp. 211 – 213, 2001

[166] H. A. Simon. *"Models of Man",* J. Willey & Sons, New York City, NY, 1957

[167] A. Sinclair. "Randomized Algorithms for Counting and Generating Combinatorial Structures", *Ph.D. thesis*, Dept. of Computer Science, University of Edinburgh, 1988

[168] A. Sinclair. *"Algorithms for Random Generation and Counting: A Markov Chain Approach"*, Progress in Theoretical Computer Science series, Birkhauser, 1992

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

[170] A. Smith. "Cellular Automata and Formal Languages", *Proceedings of the Eleventh IEEE Annual Symposium on Switching Automata Theory*, vol. 14 (4), pp. 216 – 224, 1970

[171] A. Smith. "Simple computation-universal cellular spaces" *Journal of ACM*, vol. 18 (3), pp. 339 – 353, 1971

[172] A. Smith "Real-time language recognition by one-dimensional cellular automata", *Journal of Computer and System Sciences* (JCSS), vol. 6, pp. 233 – 253, 1972

[173] J. F. Sowa. *"Knowledge Representation: Logical, Philosphical, and Computational Foundations"*, Brooks/Cole, 2000

[174] R. E. Stearns, H. B. Hunt III. "An Algebraic Model for Combinatorial Problems" *SIAM Journal on Computing*, Vol. 25, No. 2, April 1996, pp. 448 – 476

[175] L. Stockmeyer. "On the combinatorial complexity of certain symmetric Boolean functions", *Math. Systems Theory,* vol. 10, pp. 323 – 336, 1977

[176] L. Stockmeyer. "On approximation algorithms for **#P**", *SIAM Journal of Computing*, vol. 14, pp. 849 – 861, 1985

[177] K. Sutner. "Classifying circular cellular automata" *Physica D*, vol. 45 (1-3), pp. 386 – 395, 1989

[178] K. Sutner. "De Bruijn graphs and linear cellular automata" *Complex Systems*, vol. 5 (1), pp. 19 – 30, 1990

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

[180] K. Sutner. "Computation theory of cellular automata" *Proceedings of MFCS'98 – Satellite Workshop on Cellular Automata*, Brno, Czech Republic, August 1998

[181] K. Sutner. "Cellular automata and intermediate reachability problems", *Fundamenta Informaticae*, vol.52 (1-3), pp. 249 – 256, September 2002

[182] V. Terrier. "On real-time one-way cellular automata", *Theoretical Computer Science*, vol. 141, pp. 331 – 335, 1995

[183] V. Terrier. "Language not recognizable in real time by one-way cellular automata", *Theoretical Computer Science*, vol. 156, pp. 281 – 287, 1996

[184] S. Toda. "**PP** is as Hard as the Polynomial-Time Hierarchy", *SIAM Journal of Computing*, vol. 20 (5), pp. 865 – 877, 1991

[185] T. Toffoli, H. Margolus. *"Cellular Automata Machines"*, MIT Press, Cambridge, MA, 1987

[186] T. Toffoli, H. Margolus. "Invertible cellular automata: A review", *Physica D*, vol. 45, pp. 229 – 253, 1990

[187] P. Tosic. "A Perspective on the Future of Massively Parallel Computing: Fine-Grain vs. Coarse-Grain Parallel Models", in *Proceedings of the First ACM Conference on Computing Frontiers* (CF'04), Ischia, Italy, April 2004 (CD-Rom; ACM)

[188] P. Tosic. "On Counting Fixed Point Configurations in Star Networks", *Advances in Parallel and Distributed Ccomputing Models* workshop, in *Proceedings of IEEE International Parallel and Distributed Processing Symposium* (IPDPS'05), Denver, Colorado, April 2005 (CD-Rom; IEEE)

[189] P. Tosic. "On Complexity of Counting Fixed Point Configurations in Certain Classes of Graph Automata", *Electronic Colloquium on Computational Complexity*, Report ECCC-TR05-051, April 2005

[190] P. Tosic. "Counting Fixed Points and Gardens of Eden of Sequential Dynamical Systems on Planar Bipartite Graphs", *Electronic Colloquium on Computational Complexity*, Report ECCC-TR05-091, August 2005

[191] P. Tosic. "Cellular Automata for Distributed Computing: Models of Agent Interaction and Their Implications", in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics* (SMC'05), pp. 3204 – 3209, 2005 (IEEE)

[192] P. Tosic. "On Modeling and Analyzing Sparsely Networked Large-Scale Multi-Agent Systems with Cellular and Graph Automata", in *Modelling of Complex Systems by Cellular Automata* (MCSCA'06) workshop within *The International Conference on Computational Science* ICCS'06; in Springer's Lecture Notes in Computer Science, vol. 3993, pp. 272 – 280, 2006

[193] P. Tosic. "*Distributed Coalition Formation for Collaborative Large-Scale Multi-Agent Systems*", M.S. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, May 2006

[194] P. Tosic. "On the Complexity of Counting Fixed Points and Gardens of Eden in Sequential and Synchronous Dynamical Systems", *International Journal on Foundations of Computer Science* (IJFCS), vol. 17 (5), pp. 1179 – 1203, October 2006

[195] P. Tosic. "Hierarchical Autonomy of Autonomous Agents: A Functionalist, Cybernetics-Inspired Approach", submitted to a journal, 2006

[196] P. Tosic. "Computational Complexity of Some Enumeration Problems in Uniformly Sparse Boolean Network Automata", to appear in *Proceedings of the Second European Conference on Complex Systems* (ECCS'06), paper # 203 (15 pages); European Complex Systems Society, Paris, France, September 2006

[197] P. Tosic, G. Agha. "Modeling Agents' Autonomous Decision Making in Multiagent, Multitask Environments", *Proceedings of the First European Workshop on Multi-Agent Systems* (EUMAS'03), Oxford, England, December 2003

[198] P. Tosic, G. Agha. "Concurrency vs. Sequential Interleavings in 1-D Threshold Cellular Automata", *Advances on Parallel and Distributed Computing Models* workshop within *IEEE International Conference on Parallel and Distributed Processing Systems*, Santa Fe, New Mexico, USA, April 2004 (in Proc. IEEE-IPDPS'04)

[199] P. Tosic, G. Agha. "Maximal Clique Based Distributed Group Formation Algorithm for Autonomous Agent Coalitions", Workshop on Coalitions & Teams (W10), within *The Third International Joint Conference on Autonomous Agents and Multi-Agent Systems* (AAMAS'04), New York City, NY, July 2004

[200] P. Tosic, G. Agha. "Characterizing Configuration Spaces of Simple Threshold Cellular Automata", in *Proceedings of the Sixth International Conference on Cellular Automata for Research and Industry* (ACRI'04), Amsterdam, Netherlands, October 2004; Springer's *Lecture Notes in Computer Science* series, vol. 3305, pp. 861 – 870

[201] P. Tosic, G. Agha. "Towards a Hierarchical Taxonomy of Autonomous Agents", *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics* (SMC'04), vol. 4, pp. 3421 – 3426, The Hague, Netherlands, October 2004

[202] P. Tosic, G. Agha. "Maximal Clique Based Distributed Group Formation for Task Allocation in Large-Scale Multi-Agent Systems", Pre-proceedings of the First Workshop on *Massively Multi-Agent Systems* (MMAS'04), pp. 51 – 63; Kyoto, Japan, December 2004. The revised, post-proceedings version in *Lecture Notes in Artificial Intelligence* (LNAI) series, vol. 3446, pp. 104 – 120, 2005 (Springer)

[203] P. Tosic, G. Agha. "A Distributed Coalition Formation Algorithm for Collaborative Large-Scale Multi-Agent Systems", *Proceedings of the Second European Workshop on Multi-Agent Systems* (EUMAS'04), pp. 703 – 714, Barcelona, Spain, December 2004

[204] P. Tosic, G. Agha. "On Computational Complexity of Counting Fixed Points in Symmetric Boolean Graph Automata", *Proceedings of the Fourth International Conference on Unconventional Computation* (UC'05), Sevilla, Spain, October 2005; Springer's *Lecture Notes in Computer Science* (LNCS) series, vol. 3699, pp. 191 – 205

[205] P. Tosic, G. Agha. "Parallel vs. Sequential Threshold Cellular Automata: Comparison and Contrast", in *Proceedings of the First European Conference on Complex Systems* (ECCS'05), paper # 251 (20 pages); European Complex Systems Society, Paris, France, November 2005

[206] P. Tosic, G. Agha. "On Computational Complexity of Predicting Dynamical Evolution of Large Agent Ensembles", *Proceedings of the Third European Workshop on Multi-Agent Systems* (EUMAS'05), pp. 415 – 426; Flemish Academy of Sciences, Brussels, Belgium, December 2005

[207] For more on the *TRANSIMS* project at the Los Alamos National Laboratory, go to http://www-transims.tsasa.lanl.gov/ (The *Documents* link includes a number of papers and technical reports for the period 1995 – 2001)

[208] A. Treves, D. J. Amit. "Metastable states in assymetrically diluted Hopfield networks", *Journal of Physiscs A: Mathematics and General*, vol. 21, pp. 3155 – 3169, 1988

[209] H. Umeo, K. Morita, K. Sugata. "Deterministic one-way simulation of two-way real-time cellular automata and its related problems", *Information Processing Letters*, vol. 14, pp. 159 – 161, 1982

[210] S. Vadhan. "The Complexity of Counting in Sparse, Regular and Planar Graphs", *SIAM Journal of Computing*, vol. 31 (2), pp. 398 – 427, 2001

[211] L. Valiant. "The Complexity of Computing the Permanent", *Theoretical Computer Science*, vol. 8, pp. 189 – 201, 1979

[212] L. Valiant. "The complexity of enumeration and reliability problems", *SIAM Journal of Computing*, vol. 8 (3), pp. 410 – 421, 1979

[213] L. Valiant. "Short monotone formulae for the majority function", *Journal of Algorithms*, vol. 5, pp. 363 – 366, 1984

[214] J. von zur Gathen. "Parallel Linear Algebra" Section 13 in *Synthesis of Parallel Algorithms*, pp. 573 – 617, J. H. Reif (ed.), Morgan Kaufmann Publishers, San Mateo, CA, 1993

[215] D. J. Watts. *"Small Worlds: The Dynamics of Networks Between Order and Randomness"*, Princeton Univ. Press, Princeton, New Jersey, 1999

[216] D. J. Watts, S. H. Strogatz. "Collective dynamics of 'small-world' networks", *Nature*, vol. 393, 1998

[217] I. Wegener. *"The Complexity of Boolean Functions"*, Teubner Series in Computer Science, Wiley, 1987

[218] G. Weisbuch. *"Complex Systems Dynamics"*, vol. 2 of *SFI Studies in the Sciences of Complexity: Lecture Notes*, Addison-Wesley, 1990

[219] G. Weiss (ed.). *"Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence"*, The MIT Press, Cambridge, Massachusetts, 1999

[220] D. J. A. Welsh. "The computational complexity of some classical problems from statistical physics", *Disorder in Physical Systems*, pp. 307 – 321, 1990

[221] D. J. A. Welsh. *"Complexity: Knots, Colouring and Counting"*, Cambridge University Press, 1993

[222] S. Wolfram. "Statistical mechanics of cellular automata", *Reviews of Modern Physics*, vol. 55 (3), pp. 601 – 644, July 1983

[223] S. Wolfram. "Computation theory of cellular automata", *Communications in Mathematical Physics*, vol. 96, 1984

[224] S. Wolfram. "Twenty problems in the theory of CA", *Physica Scripta*, vol. 9, 1985

[225] S. Wolfram (ed.). *"Theory and applications of cellular automata"*, World Scientific, 1986

[226] S. Wolfram. *"Cellular Automata and Complexity (collected papers)"*, Addison-Wesley, 1994

[227] S. Wolfram. *"A New Kind of Science"*, Wolfram Media, Inc., 2002

[228] M. Wooldridge. *"An Introduction to Multiagent Systems"*, John Wiley and Sons Ltd, 2002

[229] M. Wooldridge, N. Jennings. "Intelligent Agents: Theory and Practice", *Knowledge Engineering Review*, vol. 10 (2), pp. 115 – 152, 1995

[230] K. Yang. "On Learning Correlated Boolean Functions Using Statistical Query", *Electronic Colloquium on Computational Complexity*, Report ECCC-TR01-098, 2001

[231] M. Yokoo. *"Distributed Constraint Satisfaction: Foundation of Cooperation in Multi-agent Systems"*, Springer, 2001

[232] M. Yokoo, K. Hirayama. "Algorithms for Distributed Constraint Satisfaction: A Review", in *Autonomous Agents and Multi-Agent Systems*, vol. 3 (2), pp. 185 – 207, 2000

[233] M. Yokoo, E. H. Durfee, T. Ishida, K. Kuwabara. "The Distributed Constraint Satisfaction Problem: Formalization and Algorithms", *IEEE Transactions on Knowledge and Data Engineering*, vol. 10 (5), pp. 673 – 685, 1998

[234] U. Zwick. "A $4n$ lower bound on the combinational complexity of certain symmetric Boolean functions over the basis of unate dyadic Boolean functions", *SIAM Journal of Computing*, vol. 20 (3), pp. 499 – 505, 1991

# Vita

Predrag T. Tošić was born and raised in Belgrade, Serbia (the former Yugoslavia). He started his undergraduate university education at University of Maryland Baltimore County (UMBC) in 1992, where he subsequently completed B.S. (majoring in mathematics and physics) in May, 1994. He then pursued M.S. in applied and computational mathematics at UMBC, which he completed in August, 1995. After completing M.S. in pure mathematics at University of Illinois at Urbana-Champaign (UIUC) two years later, and teaching and tutoring for a semester, he joined the computer science M.S./Ph.D. program at UIUC in January of 1998. Having spent some fifteen months away from University of Illinois and the world of academia (May 2000 - August 2001), Predrag returned to UIUC for the completion of his M.S. and Ph.D. degrees in computer science in the fall of 2001. He received his M.S. in computer science in the early 2006, defended his doctoral dissertation on September 14, 2006, and subsequently received Ph.D. in computer science from UIUC in December 2006.

Predrag's M.S. thesis, completed in the fall of 2005 and deposited in May 2006, summarizes his work on decentralized coordination via coalition formation in collaborative multi-agent systems. This research, supervised by Predrag's advisor, professor Gul Agha, and supported by the DARPA TASK program, was conducted at University of Illinois during 2002 – 2004.

Predrag's Ph.D. dissertation research that is summarized in this technical report was initiated while he was visiting Los Alamos National Laboratory (LANL) in New Mexico during the 2000 – 2001 academic year, and subsequently continued at University of Illinois until the dissertation completion in the fall of 2006. Predrag's doctoral research is centered at using variants of cellular and network automata for modeling and formal analysis of large-scale multi-agent systems and various aspects of their collective behavior. This research approach is inherently cross-disciplinary, and utilizes techniques, models, paradigms and analytical tools ranging from theoretical computer science and formal methods to distributed artificial intelligence to discrete dynamical systems and complex networks.

In addition to large-scale multi-agent systems, as well as distributed artificial intelligence and theoretical computer science in general, Predrag's research interests also span formal methods, parallel and distributed computing, various distributed information systems, connectionist AI, computational theories of language evolution, discrete dynamical systems, and agent-based modeling of complex networks. As of the fall of 2006, he has authored nearly thirty peer-reviewed research publications.