PARALLEL vs. SEQUENTIAL THRESHOLD CELLULAR AUTOMATA: Comparison and Contrast

Predrag T. Tošić * and Gul A. Agha
Open Systems Laboratory, Department of Computer Science,
University of Illinois at Urbana-Champaign,
1334 Thomas Siebel Center for Computer Science, 201 N. Goodwin, Urbana, IL 61801, USA
Contact author's phone: +1-217-244-1976 Fax: +1-217-333-9386
Electronic mail: {p-tosic, agha}@cs.uiuc.edu

ABSTRACT

Cellular automata (CA) *are an abstract model of a distributed dynamical system, as well as of* fine-grain parallelism *in computing. In a classical cellular automaton, all the nodes execute their operations in parallel and* in perfect synchrony. *We consider herewith the sequential version of* CA, *called* SCA, *and compare those* SCA *with the classical,* parallel CA. *In particular, we show that there are* 1D CA *with very simple node update rules that cannot be simulated by any comparable* SCA, *irrespective of the node update ordering. Consequently, the granularity of the basic* CA *operations and, therefore, the* fine-grain parallelism *of the classical, synchronous* CA, *insofar as the "interleaving semantics" is concerned, turns out to be* not fine enough. *We also study in some detail the properties of the cellular automata whose nodes update their states according to the* Majority *update rule. Finally, we share some thoughts on how to extend the presented results, and, in particular, we try to motivate the study of* genuinely asynchronous cellular automata.

Keywords: analysis and dynamics of complex networks, cellular automata, discrete dynamical systems, configuration space properties, communication models

1. Introduction and Motivation

Cellular automata (CA) were originally introduced as an abstract mathematical model that can capture the behavior of biological systems capable of self-reproduction [24]. 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., [16, 36, 37, 38, 39]). However, CA can also be viewed as an abstraction of massively parallel computers (e.g, [11]). Herein, we study 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 cellular automata's parallelism.

It is well known that CA are 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. We shall compare SCA with the perfectly synchronous *parallel* (or *concurrent*) CA. In particular, we will show that there are *1D CA* with very simple state update rules that cannot be simulated by any comparable SCA, irrespective of the node update ordering. While the

*Contact author.

result would be trivial if one considers a single (S)CA computation, we argue that the result is both nontrivial and important when applied to *all possible inputs* (starting configurations) and to the entire classes of CA and SCA. Hence, the granularity of the basic CA operations, insofar as the (im)possibility of simulating their concurrent computation via appropriate sequential interleavings of these basic operations, turns out not to be quite *fine enough.* Furthermore, we will characterize in some detail the possible computations of certain types of *threshold cellular automata.* Last but not least, we will also share some thoughts on how to extend the work presented in this paper. In particular, we will 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 *communication* among different nodes.

An example of asynchrony in the local node updates (i.e., 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 [19, 20]. It is important to understand, however, that even in the 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) (A)CA as to *sequential cellular automata,* and shall consistently keep the term *asynchronous cellular automata* only for those CA that do not have a global clock (see Section 4).

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 *Chapter* §12 of [31], "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 [31]. In contrast, [29] 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". [29] 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.* That is, when referring to parallel (or, equivalently, concurrent) computation, we shall always assume *perfect synchrony.*

1.1. Capturing Concurrency via 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 parallel computer program, we 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., [10, 12, 18, 22, 23]).

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 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.

We shall illustrate the concept of *sequential interleaving semantics* of concurrency with a simple exercise. Let's consider the following question from a sophomore parallel programming class: *Find an 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

$$x \leftarrow x + 1;\ x \leftarrow x + 1$$
$$\text{vs.}$$
$$x \leftarrow x + 1\ ||\ x \leftarrow x + 1$$

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 informally 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) 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.

We address in this work the (in)adequacy of the sequential interleavings semantics when applied to CA where the individual node updates are considered to be elementary operations. It is tacitly assumed that the complete node update operation includes, in addition to computing the local update function on appropriate inputs, also 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 Section 2; see also discussion in Sections 4 and 5.

In particular, we will 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 (see Section 2), to fail rather dramatically even in the context of the simplest *nonadditive* CA node update rules.

2. Cellular Automata and Types of Their Configurations

We will follow [11] and formally define classical (that is, synchronous and concurrent) CA in two steps: we shall first introduce the notion of a *cellular space,* and then define a *cellular automaton* over an appropriate cellular space.

Definition 1 *A Cellular Space,* Γ*, 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 Γ by L. That is, L may be equal to, or be a proper subset of, the set of all integers.

Definition 2 *A Cellular Automaton A is an ordered triple* (Γ, N, M)*, where*

- *Γ 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 $\delta : Q^{|N|+1} \rightarrow Q$ for CA with memory, and $\delta : Q^{|N|} \rightarrow Q$ for memoryless CA.*

The fundamental neighborhood N specifies which of the near-by nodes provide inputs to the update rule of a given node. In the classical CA, Γ is a regular graph that locally "looks the same everywhere"; in particular, the local neighborhood N is the same for each node in Γ.

The local transition rule δ specifies how each node updates its state (that is, value), based on its current state 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.

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 [33]. In particular, the only *memory content* of a CA is a tuple of the current states of all its nodes. Moreover, as a node can "read" (but not "write") the states 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 (typically, one bit) of its *local memory,* physically separated from other similar *local memories* - yet this local memory is *directly accessible* (for *read* accesses) to some of the other *processors.* In particular, the "reads" from any *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 paper will pertain to a comparison and contrast between the

classical, concurrent threshold CA and their sequential counterparts, we formally introduce two types of the 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 3 *A Sequential Cellular Automaton (SCA)* S *is an ordered quadruple* (Γ, N, M, s), *where* Γ, N *and* M *are as in* Definition 2, *and* s *is an arbitrary sequence, finite or infinite, all of whose elements are drawn from the set* L *of integers used in labelling the vertices of* Γ. *The sequence* s *is specifying the sequential ordering according to which an 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 CA with a *particular* SCA (that is, some concrete SCA with a particular choice of the update sequence s), we would like to compare the parallel CA computations with the computations of the corresponding SCA for *all* possible sequences of node updates. For that purpose, the following class of nondeterministic sequential cellular automata is introduced:

Definition 4 *A Nondeterministic Interleavings Cellular Automaton (NICA)* I *is defined to be the union of all sequential automata* S *whose first three components,* Γ, N *and* M *are fixed. That is,* $I = \cup_s (\Gamma, N, M, s)$, *where the meanings of* Γ, 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* Γ.

We next 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 system* view of CA is helpful. A *phase space* (also called *configuration 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 discuss how these definitions need to be modified in the case of SCA and NICA.

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

Definition 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 FP is a special, degenerate case of a recurrent state with period 1. Due to deterministic evolution, any configuration of a classical, parallel CA is either a FP, a *proper* CC, or a TC. Throughout, we shall make a clear distinction between FPs and proper CCs.

On the other hand, if one considers SCA and NICA, 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 the 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", 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 (see the related discussion and examples in [34])).

For simplicity, heretofore we shall refer to a configuration C of a NICA as a *(weak) fixed point* if there exists some infinite sequence of node updates s such that 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 C' to be a cycle state, if there exists an infinite sequence of node updates s' such that, if NICA nodes update according to s', then C' is a cycle state of period 2 or greater in the usual sense (see *Definition* 5). In particular, in case of the NICA automata, a single configuration can simultaneously be a weak FP, a CC and a TC; see reference [34] for a simple example.

3. 1D Parallel vs. Sequential CA Comparison for Simple Threshold Rules

After the introduction, motivation and the necessary definitions, we now proceed with our main results and their meaning. Technical results are given in this section. Discussion of the implications and relevance of these results, as well as some possible generalizations and extensions, will follow in Section 4.

We will compare and contrast the classical, parallel CA with their sequential counterparts, SCA and NICA, in the context of the simplest nonlinear and nonaffine local update rules possible, namely, the CA in which the nodes locally update according to *linear threshold functions.* Moreover, we choose these threshold functions to be *symmetric,* so that the resulting (S)CA are also *totalistic* (see, e.g., [11] or [38]). We show the fundamental difference in the configuration spaces, and therefore possible computations, between the parallel threshold CA and the sequential threshold CA: while the former can have temporal cycles (of length two), the computations of the latter always either converge to a fixed point, or otherwise fail to finitely converge to any recurrent pattern whatsoever.

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

Definition 6 *A* 1D (sequential) cellular automaton *of radius* $r \geq 1$ *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 to the right that are no more than* r *nodes away. For the* (S)CA with memory, *the next state of a node also depends on that node's own current state.*

Thus, for the Boolean (S)CA *with memory* defined over a one-dimensional cellular space Γ, 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 1D 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, Γ is a two-way infinite string (or "line").

We fix the following conventions and terminology. Throughout, only *Boolean* CA, SCA and NICA are considered; in particular, the set of possible states of any node is $\{0, 1\}$. The phrases

"monotone symmetric" and "symmetric (linear) threshold" functions / update rules / automata are used interchangeably. Similarly, "(global) dynamics" and "(global) computation" are used synonymously. Unless stated otherwise, *CA* denotes a classical, parallel 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 Γ 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.

3.1. On the Existence of Cycles in Parallel and Sequential Threshold Cellular Automata

We begin by defining *linear threshold functions* and *simple threshold functions,* and the CA, SCA and NICA whose updates rules are restricted to such threshold functions.

Definition 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 θ is an appropriate threshold constant, and $w_1, ..., w_m$ are arbitrary (but fixed) real numbers, called weights.

In general, weights w_i in the definition above can be both positive and negative. This is esp. 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, near-by 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., [5, 6, 34, 36, 37].

Definition 8 *A* threshold automaton (threshold (S)CA) *is a (parallel or sequential) cellular automaton whose node update rule δ is a Boolean-valued linear threshold function.*

Therefore, given an integer $k \geq 0$, a k-threshold function, in general, is any function of the form as in *Definition* 8 with $\theta = k$ and an appropriate choice of weights w_i, $i = 1, ..., m$. 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 δ to be a *symmetric function* of all of its inputs. 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* [5, 6, 34].

Definition 9 *A simple threshold (S)CA is a cellular automaton whose local update rule δ is a monotone symmetric Boolean (threshold) function.*

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 θ.

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

$$\delta(x_{i-r}, x_{i-r+1}, ..., x_i, ..., x_{i+r-1}, x_{i+r}) \;=\; \begin{cases} 1, & \text{if } \sum_{j=-r}^{r} x_{i+j} \;\geq\; k \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

where k is a fixed integer from the range $\{0, 1, ..., 2r+1, 2r+2\}$.

Due to the nature of the node update rules, cyclic behavior intuitively should not be expected in such 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 subsection largely stems from the following three factors:

- the local update rules are the simplest nonlinear totalistic rules one can think of;
- given the rules, the cycles are not to be expected - yet they do exist, and in the case of synchronous parallel CA *only;* and, related to that observation,
- it is, for this class of automata, the parallel CA that have more diverse possible dynamics than any of their sequential counterparts, and, in particular, while qualitatively there is nothing among the possible sequential computations that is not present in the parallel case, the parallel threshold CA do exhibit a particular qualitative behavior that cannot be reproduced by any threshold SCA or NICA.

The results that follow hold for two-way infinite 1D (S)CA, as well as for finite (S)CA with the circular boundary conditions (i.e., for the (S)CA whose cellular spaces are finite rings).

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

(i) Any 1D parallel CA with $r = 1$, the MAJORITY *update rule $\delta = MAJ$, and an even number of nodes, has finite temporal cycles in the phase space (PS); the same holds for the two-way infinite* 1D MAJORITY CA.

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

Proof. To show (i), we exhibit an actual two-cycle. Consider either an infinite 1D CA, or a finite one, with circular boundary conditions and an even number of nodes, $2n$. Then the configurations $(10)^\omega$ and $(01)^\omega$ in the infinite case ($(10)^n$ and $(01)^n$ in the finite ring case) form a 2-cycle. A proof of part (ii) based on a straight-forward case analysis can be found in our prior publication [34]. □

We remark that, insofar as the infinite SCA as in *Lemma* 1 are concerned, a nontrivial temporal cycle configuration cannot exist even in the limit.

Part (ii) of *Lemma* 1 above can be readily generalized: even if we consider local update rules δ other than the MAJORITY rule, yet restrict δ to *monotone symmetric (Boolean) functions,* that is, the simple threshold functions [5, 6, 34], such sequential CA still do not have any proper cycles.

Theorem 1 *For any Simple Threshold 1D Sequential CA A with $r = 1$, and any sequence s of the node updates, the phase space PS(A) is temporal cycle-free.*

Proof. Since $r = 1$ and $2r + 1 = 3$, there are only *five* simple threshold functions on three inputs. Two of those five 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 straight-forward 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 simple threshold update rule on three inputs is $\delta = MAJ$, for which we have already argued that the corresponding parallel CA have

temporal two-cycles, but all the corresponding SCA (and therefore the NICA) have cycle-free configuration spaces. □

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

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

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

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

The proof of this Lemma can be found in our earlier work [34].

Generalizing Lemmata 1 and 2, part (i), we have the following

Corollary 1 *For all $r \geq 1$, there exists a* monotone symmetric CA *A such that A has finite temporal cycles in the phase space.*

Namely, given any $r \geq 1$, a parallel CA with $\delta = MAJ$ and $\Gamma =$ *infinite line* has at least one two-cycle in the *PS*: $\{(0^r1^r)^\omega, (1^r0^r)^\omega\}$. If $r \geq 3$ is odd, then such a threshold automaton has at least two distinct two-cycles, since $\{(01)^\omega, (10)^\omega\}$ is also a two-cycle. Analogous results hold for the more general *simple threshold CA* defined on finite 1D cellular spaces, provided that such automata have sufficiently many nodes, that the number of nodes is appropriate (see [35] for more details), and assuming circular boundary conditions. Moreover, the result extends to many finite and infinite CA in the higher dimensions, as well; in particular, *threshold CA* with $\delta = MAJ$ that are defined over the $2D$ Cartesian grids or hypercubes also have two-cycles in their respective phase spaces.

It turns out that the two-cycles in the *PS* 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* δ, and any *finite* regular Cayley graph as the underlying cellular space, the following general result holds [11, 15]:

Proposition 1 [*15*] *Let a parallel simple threshold* CA *$A = (\Gamma, N, M)$ be given, where Γ is any* finite *cellular space, and let this cellular automaton's* global map *be denoted by F. Then for all configurations $C \in PS(A)$, there exists a finite time step $t \geq 0$ such that $F^{t+2}(C) = F^t(C)$.*

In particular, this result implies that, in case of any *finite* simple threshold CA, for any starting configuration 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 Γ 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 to 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 Γ 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 (esp. when the underlying Γ 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 the paper.

For 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 Dynamical Systems* (SDSs); see, e.g., [5, 6, 7, 8]. 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 SCA and NICA to be fair in a sense defined below, without imposing any further restrictions or investigating how are such fair sequences of node updates to be generated in a distributed setting. For our present purposes, the following simple notion of fairness will suffice:

Definition 10 *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., $s^{[q]} = (s_{q+1}, s_{q+2}, s_{q+3}, ...)$, We state the following alternative characterizations of fair sequences:

Lemma 3 *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) *$\forall n \in N,\quad s^{[n]} : \{n+1, n+2, ...\} \to L$ is* onto.

Now that we have defined what we mean by a *single step* of a sequential CA, as well as adopted some reasonable notion of *fairness,* we have set the stage for the following generalization of *Proposition* 1 to both finite and infinite 1D CA and 1D SCA:

Proposition 2 *Let a parallel CA or a sequential SCA be defined over a finite or infinite 1D cellular space, with a finite rule radius $r \geq 1$. Let this cellular automaton's local update rule be a simple threshold function. Let's also assume, in the sequential cases, that the fairness condition from* Definition 10 *holds. Then for any starting configuration $C_0 \in PS(A)$ whatsoever, and any finite subconfiguration $C \subseteq C_0$, there exists a time step $t \geq 0$ such that*

$$F^{t+2}(C) = F^t(C) \tag{5}$$

where, in the case of fair SCA, the Eqn. (5) *can be replaced with*

$$F^{t+1}(C) = F^t(C) \tag{6}$$

In the case of $\delta = MAJ$ (S)CA, a computation starting from any *finitely supported*[a] initial configuration necessarily converges to either a FP or a two-cycle [15]:

Proposition 3 *Let the assumptions from* Proposition 2 *hold, and let the underlying threshold rule be $\delta = MAJ$. Then for all configurations $C \in PS(A)$ whatsoever in the finite cases, and for all configurations $C \in PS(A)$ such that C has a* finite support *when $\Gamma(A)$ is infinite, there exists a finite time step $t \geq 0$ such that $F^{t+2}(C) = F^t(C)$. Moreover, in the sequential cases with* fair *update sequences, there exists a finite $t \geq 0$ such that $F^{t+1}(C) = F^t(C)$.*

[a] Also sometimes called *compactly supported;* see, e.g., [15]. A global configuration of a cellular automaton defined over an infinite cellular space Γ is said to be *compactly supported* if all except for at most finitely many of the nodes are *quiescent* (i.e., in state 0) in that configuration.

Furthermore, if *arbitrary* infinite initial configurations are allowed in *Propositions* $2-3$, 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 that case, and under the fairness assumption in the case of SCA, the limiting configuration $lim_{t\rightarrow\infty}F^t(C) = C^{lim}$ can be shown to be a fixed point.

To summarize, if the computation of a SCA starting from some configuration C converges at all (that is, to *any* finite recurrent structure), it actually has to converge to a fixed point.

To convince oneself of the validity of *Proposition* 2, two basic facts have to be established. One, convergence to finite temporal cycles of length three or higher is not possible. Indeed, *Proposition* 1 establishes that the only possible long-term behaviors of the finite simple threshold CA are (i) the convergence to a fixed point and (ii) the convergence to a two-cycle. The only possibility for fair finite SCA is the convergence to a fixed point. If infinite cellular spaces are considered, it is straight-forward 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 Γ such divergence indeed takes place - even when the starting configuration is finitely (compactly) supported: consider, e.g., the $\delta = OR$ CA 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* 1, 2 and *Theorem* 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 [6] for a class of sequential graph automata called *Sequential Dynamical Systems* that we have already mentioned. In particular, part (ii) in the *Theorem* 2 below and its proof are directly based on [6]:

Theorem 2 *The following dichotomy holds:*

(i) All 1D (parallel) CA *with any odd* $r \geq 1$, *the local rule* $\delta =$ MAJ, *and cellular space* Γ *that is 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* Γ *is either a finite ring with a number of nodes divisible by* $2r$, *or a two-way infinite line.*

(ii) Any 1D SCA *with* any *monotone symmetric Boolean update rule* δ, *for any finite* $r \geq 1$, *defined over a finite or infinite 1D 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.*

Remark: There are also CA defined over finite rings and with even $r \geq 2$ such that the number of nodes in those 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, however, for the sake of clarity and space constraints.

Proof.

Part (i): Consider first $\Gamma =$ *infinite line.* 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 $r \geq 2$. Thus, the "canonical" temporal two-cycle for 1D 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 $2n$ nodes (and assuming 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 [6] for the aforementioned SDSs. A *simple symmetric SDS* is a sequentially updating graph automaton with (possibly different) k-threshold update rules at different nodes, and with the node update ordering given by a *fixed* permutation of the nodes. The central idea of the proof 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, the unordered pair $\{x_i, x_j\}$ is an edge if and only if these two nodes provide inputs to one another, i.e., in the 1D SCA case, if and only if *distance*$(x_i, x_j) \leq r$ (that is, assuming the canonical labelling of the nodes, so that the consecutive nodes always get labeled by the 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 this configuration) is always bounded, the fact that each "flip" of any node's value strictly decreases the overall potential by integer amounts implies that, after a finite number of node flips, an equilibrium where no nodes can further flip is reached; this equilibrium will be a fixed point configuration. □

To summarize, simple threshold CA, depending on the starting configuration, may converge to a fixed point or to a temporal two-cycle; in particular, they may end up "looping" in finite (but nontrivial) temporal cycles. In contrast, the corresponding classes of SCA (and therefore NICA) can never cycle. We also observe that, given any sequence of node updates of a finite threshold SCA, if this sequence satisfies the fairness condition from *Definition* 10, then it can be shown that the computation of such a threshold SCA A is guaranteed to converge to a fixed point (sub)configuration on any finite subset of the nodes in $\Gamma(A)$.

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

3.2. Characterizing Configuration Spaces of 1D (S)CA *with* $\delta =$ MAJORITY

Next, we specifically focus on $\delta =$ MAJORITY 1D CA, and characterize the configuration spaces of such threshold cellular automata. In particular, in the $\Gamma =$ *infinite line* case, we show that the cycle configurations are rather rare, that 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 (S)CA are transient.

In the sequel, for the finite 1D (S)CA circular boundary conditions will be assumed by default. Thus, the cellular spaces in this section will be either infinite lines, or finite rings. In the case of SCA, the fairness condition based on *Definition* 10 will be assumed. Also, when we refer to FPs of NICA, we mean *weak* fixed points. (Recall that a weak FP is a configuration such that there exists an infinite sequence of individual node updates that satisfies the fairness condition and so that, with respect to this sequence, the particular configuration is a "proper" FP - but the same configuration may be, for example, a proper TC with respect to other sequences of node updates).

We begin with some observations about the nature of various configurations in the (S)CA with $\delta = MAJ$ and $r = 1$. We shall subsequently generalize several of these 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 $\delta = MAJ$ 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 *A fixed point configuration of a 1D-(S)CA with $\delta = MAJ$ 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 subconfigurations. Without loss of generality, assume a CC of a parallel MAJORITY CA contains a block of two or more consecutive 0s. Consider, say, the node adjacent to the rightmost 0 in that block. Let's denote that node by x_j. This node x_j is, by assumption, in the state 1. There are two cases to consider. If our node's right neighbor, x_{j+1}, is in the state 0, then, at the very next parallel step, the node x_j, that has at least two out of three of its input bits equal to 0, will itself flip to 0. Since x_j is adjacent to $x_{j+1} = 0$, it will have joined an expanded stable block of zeros, and consequently x_j will remain at 0 thereafter. Hence, the starting configuration cannot be a CC.

The other possibility is that x_{j+1} is also in the state 1; that is, the configuration is of the form $...x_{j-2}x_{j-1}x_jx_{j+1} = ...0011....$ Then the block of (at least) two consecutive 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 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 this configuration must be a fixed point. Either way, the assumption that this configuration was actually a cycle configuration cannot hold. □

Lemma 5 *The FPs of the 1D-(S)CA with $\delta = MAJ$ and $r = 1$ are precisely of the form $(000^* + 111^*)^*$. The CCs of such 1D-CA may exist only in the parallel case, and the temporal cycles are precisely of the form $\{(10)^*, (01)^*\}$. The TCs of CA are all other configurations, 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.*

The claim of Lemma 5 follows by the result of Lemma 4 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 entirely made of the consecutive stable blocks of 0s and/or 1s. Likewise, giving a similar characterization for the higher dimensional cellular spaces is also not as straight-forward as the results in the Lemmata above. Therefore, the nice and clean partition of the configurations into FPs, CCs and TCs obtained in this section is attributable to the peculiarity of the 1D cellular spaces, as well as the assumption that the rule radius is $r = 1$.

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, any 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., Chapter 4 in [15]. 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 $(10)^*$, $(01)^*$ is a fixed point. However, for all $r \geq 1$, the corresponding (parallel) CA are guaranteed to have some temporal cycles, namely, given r, the set of states $\{(1^r0^r)^*, (0^r1^r)^*\}$ forms a two-cycle.

Back to the $r = 1$ case, we also establish the following property:

Lemma 6 *Given any (finite or infinite) simple threshold (S)CA* with memory *and with the rule radius $r = 1$, one of the following two properties always holds:*

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

(ii) if there are cycle configurations in the PS of this 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 δ 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 any CC 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 this transient configuration, and let C denote the cycle configuration in question (so that $F(C') = C$ where F, as before, stands for this cellular automaton's global map). By Lemmata 4 and 5, 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, $F(C')$ 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* 6 actually holds for *arbitrary* rule radii $r \geq 1$, but do not have a proof - nor do we know of a counterexample - as of yet:

Conjecture: Given any (finite or infinite) simple threshold parallel or sequential CA with any rule radius $r \geq 1$, exactly one of the following two properties always holds:

(i) either this simple threshold (S)CA does not have proper cycles and cycle configurations, or else

(ii) if there are cycle configurations in the *PS* of this cellular automaton, then none of those temporal cycles has any incoming transients.

Next, we show that the fixed points of simple threshold cellular automata may be quite numerous when $\delta = MAJ$. Infinite sequential and parallel MAJ 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 the case of $\delta = MAJ$ and the (countably) infinite cellular spaces, equals the cardinality of the entire *PS*:

Theorem 3 *An infinite* 1D-(S)CA *with* $\delta =$ MAJ *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 on the line.

Let us consider FPs of the form $1^{r+k_1}0^{r+k_2}1^{r+k_3}...$ with all k_i being integers such that $k_i \geq 1$ $(i = 1, 2, 3, ...)$. Let a string of $r + k_i$ *consecutive* 1s or 0s as above be mapped into 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 $[0, 1]$. Let the length of the m-th block of consecutive 0s or 1s be denoted by L_m. Then $L_m \geq r + 1$ gets mapped into $L_m - 2$ $(mod\ 10)$. For instance, 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 uncountable many "points" (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 (proper) subset of the set of all fixed points of such an infinite (S)CA, it follows that any 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 $\delta = MAJ$ 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^ω and 1^ω. Other threshold CA may have temporal cycles, as shown in the previous subsection, but they still have only a finite number of FPs.

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

Lemma 7 *If a global configuration of an infinite threshold CA 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.*

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 one.

Proposition 4 *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 simple threshold (S)CA being in the state* 1 *be equal to p. If a global configuration of this threshold automaton is selected at random according to p, then, with probability* 1*, this randomly selected configuration will be transient.*

Proof. 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 of a randomly selected infinite configuration:

- with probability 1, any finite substring of 0s and 1s appears *somewhere* in this infinite configuration;

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

- the same holds of any unstable finite subconfiguration, such as, e.g., 10101 or 01010101.

Consequently, since such a randomly selected configuration contains unstable subconfigurations with probability 1, it follows that, with probability 1, it cannot be a fixed point. Moreover, for any fixed integer m, each of the strings of 0s and 1s of length m appear infinitely often in such a random configuration. In particular, with probability 1, somewhere in the configuration, some finite unstable subconfiguration, let's denote it C_u, is squeezed in between some two stable subconfigurations, say, 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 those nodes is going to flip, causing it to join the (expanded) stable block, either C_{s1} or C_{s2}. It therefore follows that the overall (infinite) configuration C that contains the concatenation $C_{s1} \cdot C_u \cdot C_{s2}$ as its finite subconfiguration, cannot be a cycle configuration. □

In case of the finite threshold (S)CA, as the number of nodes, N, increases, the fraction of all 2^N global configurations that are TCs also grows. In particular, under the same assumptions as in *Lemma* 7 and *Proposition* 4 above, in the limit, as $N \to \infty$, the probability that a randomly picked configuration, C, is a transient state approaches 1:

$$lim_{N\to\infty}\ Pr(random\ C\ is\ transient)\ =\ 1 \tag{7}$$

Thus, a fairly complete characterization of the configuration spaces of threshold (S)CA over 1D cellular spaces can be given. In particular, in the infinite threshold CA cases, *almost every* configuration is a TC. However, a striking contrast between the MAJORITY CA on the one, and the CA with any other threshold rule on the other hand, remains: the former have uncountably many FPs, whereas all other simple threshold CA can have only finitely many FPs.

4. Discussion and Future Directions

The results in Section 3 show that the very existence of temporal two-cycles in simple threshold CA can be ascribed entirely to the assumption of *perfect synchrony* of the parallel node updates. In the actual engineering, physical or biological systems that can be modeled by CA, however, such perfect synchrony is usually hard to justify. In particular, when CA are applied to modeling of various complex physical or biological phenomena (such as, e.g., crystal growth, forest fire propagation, information or gossip diffusion in a population, or signal propagation in an organism's neural system), one ought to primarily focus on the underlying CA behaviors that are, in some sense, *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* [3]. It can be readily argued in a rigorous, probabilistic sense that the typical, statistically robust behavior of threshold (S)CA computations is a relatively short transient chain, followed by convergence to a fixed point. In particular, the non-fixed-point temporal cycles of the threshold CA with $r = 1$, for example, not only lack any nontrivial basins of attraction (in terms of the incoming transient 'tails'), but are themselves statistically negligible for all sufficiently large finite, as well as for all infinite CA.

We now briefly discuss some possible extensions of the results presented thus far. In particular, we are considering extending our study to *non-homogeneous threshold CA,* where not all the nodes necessarily update according to *one and the same* threshold update rule. Likewise, we are interested in exploring the implications of heterogeneity of the underlying network

structure, that is, what are the effects on the possible behaviors of threshold automata whose underlying cellular spaces are not necessarily one-dimensional, or, more generally, not even regular graphs. We remark that the two particular classes of graph 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., [5, 6, 8, 30] and references therein.

Another future direction is to consider other communication models in cellular automata. We argue that the classical parallel 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 δ 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.

In the most general setting, we would like to consider various types of *asynchronous cellular and graph automata,* where the nodes are not assumed any longer to update in unison and, moreover, where no global clock is assumed. We again emphasize that such cellular automata would entail what can be viewed as *communication asynchrony,* thus going beyond the kind of asynchrony in local computations at different nodes *only,* that has been studied since at least 1984 [19, 20].

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

$$x_i^{t+1} = f(x_i^t, x_{i_1}^t, ..., x_{i_{2r}}^t) \tag{8}$$

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

$$x_i^{t+1} = f(x_i^t, x_{i_1}^{t_1}, ..., x_{i_{2r}}^{t_{2r}}) \tag{9}$$

We observe that t in *Eqn.* (8) 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 *Eqn.* (9), each t_j (for $j \in \{1, ..., 2r\}$) 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 a global clock has considerable implications. The challenge arises, how to meaningfully relate these different local times, so that one can still mathematically analyze such ACA - yet without making the model's description too complicated, that is, while staying away from introducing the explicit *sends* and *receives,* message buffers, etc.? Yet, if we want to study *genuinely* asynchronous CA models (rather than arbitrary sequential models with global clocks), changes along the indicated general lines in the definition of the node update rules seem unavoidable.

We point out that this, genuine (that is, communication) asynchrony in CA (see *Eqn.* (9)) can also be readily interpreted in the nondeterministic terms: at each time step, 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 the particular value of x_{i_j} that x_i

had used as its input on its most recent previous turn to update. That is, insofar as what are the current inputs to any given node's update function δ, 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 (Γ, 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 the results in Section 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 be readily observed that, indeed, the FPs are also invariant for the asynchronous CA and graph 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 [5] on the FP invariance for sequential and parallel graph automata are just special cases of this, more general result.

Theorem 4 *Given an arbitrary asynchronous cellular or graph automaton, any fixed point configuration 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.

In addition to studying invariants under different assumptions on asynchrony and concurrency, we also consider a broad qualitative comparison-and-contrast of the asynchronous CA that we propose, with the parallel CA and the sequential SCA and NICA. Such a study would shed more light on those emerging collective behaviors that are solely due to network delays.

5. Summary and Conclusions

We have presented in this paper 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 model of the sequential interleaving semantics of concurrency, we have tried to apply this metaphor to parallel CA, thereby motivating the study of sequential cellular automata (SCA) and the sequential interleavings cellular automata (NICA). In particular, we have undertaken a comparison and contrast between the sequential SCA/NICA and the classical, parallel CA models when the node update rules are restricted to *simple threshold functions.* Concretely, we have shown that, even in some very simplistic cases, the sequential "interleaving semantics" of NICA fails to capture concurrency of the parallel CA. One lesson is that, simple as they may be, the basic local operations of the classical CA cannot always be considered atomic. 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:

- Fetching all the neighbors' values ("receiving" or "reading shared variables");
- Updating one's own state according to the update rule δ (that is, performing the local computation); and
- Informing the neighbors of the update, i.e., making available one's new state/value to the neighbors ("sending" or "writing a shared variable").

Motivated by the early results on the sequential and parallel 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* (ACA) and to formally study their properties. Our hope is that the models along the lines of ACA would lead to some significant future insights

into the fundamental issues related to bounded vs. unbounded asynchrony, formal sequential semantics for parallel and distributed computation, and, on the dynamical systems side, to the identification of many of those parallel CA, SCA, NICA and/or ACA phase space properties that are solely or primarily due to the assumed communication model, that is, the (a)synchrony in both the local node updates and the inter-node interactions.

To conclude, we find appropriate 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, yet mathematically elegant and comprehensive level.

Acknowledgments: This work was supported in part by the ONR MURI Grant, contract number N00014-02-1-0715, as well as the DARPA IPTO TASK Program, contract number F30602-00-2-0586. The first author would also like to thank the ECCS'05 organizers and European Complex Systems Society for the partial travel support. Many thanks to Reza Ziaei for valuable discussions related to the work presented in this paper.

References

1. M. Anthony. "Threshold Functions, Decision Lists, and the Representation of Boolean Functions", NeuroCOLT Techn. Report Series (NC-TR-96-028), January 1996
2. S. Amoroso and Y. Patt. "Decision procedures for surjectivity and injectivity of parallel maps for tessellation structures", *J. of Computer and System Sciences (JCSS),* vol. 6, pp. 448-464, 1972
3. W. Ross Ashby. *"Design for a Brain",* Wiley, 1960
4. C. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz and R. E. Stearns. "Predecessor and Permutation Existence Problems for Sequential Dynamical Systems", Los Alamos National Laboratory Report, LA-UR-01-668, 2001
5. C. Barrett, H. Hunt, M. Marathe, S. S. Ravi, D. Rosenkrantz, R. Stearns, and P. Tosic. "Gardens of Eden and Fixed Points in Sequential Dynamical Systems", *Discrete Math. and Theoretical Comp. Sci. (DMTCS),* Proc. AA DM-CCG, pp. 95-110, 2001
6. C. Barrett, H. B. Hunt III, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns. "Reachability problems for sequential dynamical systems with threshold functions", *Theoretical Computer Science (TCS),* vol. 295, issues 1-3, pp. 41-64, Feb. 2003
7. C. Barrett, H. Mortveit, and C. Reidys. "Elements of a theory of computer simulation II: sequential dynamical systems", *Applied Mathematics and Computation,* vol. 107 (2-3), 2000
8. C. Barrett, H. Mortveit, and C. Reidys. "Elements of a theory of computer simulation III: equivalence of sequential dynamical systems", *Appl. Math. and Comput.,* vol. 122 (3), 2001
9. C. Barrett and C. Reidys. "Elements of a theory of computer simulation I: sequential CA over random graphs", *Appl. Math. and Comput.,* vol. 98 (2-3), 1999
10. I. Czaja, R. J. van Glabbek, U. Goltz. "Interleaving semantics and action refinement with atomic choice", in "Advances in Petri Nets", (G. Rozenberg, ed.), LNCS 609, Springer-Verlag, 1992
11. Max Garzon. *"Models of Massive Parallelism: Analysis of Cellular Automata and Neural Networks",* Springer, 1995
12. R. J. van Glabbek, U. Goltz. "Equivalences and refinement", Proc. LITP Spring School Theoretical CS, La Roche-Posay, France (I. Guessarian, ed.), LNCS 469, Springer-Verlag 1990
13. E. Goles, F. Fogelman, D. Pellegrin. "Decreasing energy functions as a tool for studying threshold networks", *Discr. Appl. Math.,* vol. 12, pp. 261 - 277, 1985
14. E. Goles, S. Martinez. "Exponential transient classes of symmetric neural networks for synchronous and sequential updating", *Complex Systems,* vol. 3, pp. 589 - 597, 1989

15. E. Goles, S. Martinez. *"Neural and Automata Networks: Dynamical behavior and Applications",* Math. and Its Applications series (vol. 58), Kluwer, 1990
16. E. Goles, S. Martinez (eds.). *"Cellular Automata and Complex Systems",* Nonlinear Phenomena and Complex Systems series, Kluwer, 1999
17. H. Gutowitz (ed.). *"Cellular Automata: Theory and Experiment",* The MIT Press / North-Holland, 1991
18. C. A. R. Hoare. *"Communicating Sequential Processes",* Prentice Hall, 1985
19. T. E. Ingerson and R. L. Buvel. "Structure in asynchronous cellular automata", *Physica D: Nonlinear Phenomena,* Vol. 10, Issues 1-2, pp. 59 - 68, January 1984
20. S. A. Kauffman. "Emergent properties in random complex automata" (ibid.)
21. R. Milner. *"A Calculus of Communicating Systems",* Springer-Verlag Lecture Notes in Computer Science (LNCS), 1980
22. R. Milner. "Calculi for synchrony and asynchrony", *Theoretical Computer Science (TCS),* vol. 25, pp. 267 - 310, 1983
23. R. Milner. *"Communication and Concurrency",* Prentice-Hall, 1989
24. J. von Neumann. *"Theory of Self-Reproducing Automata",* edited and completed by A. W. Burks, Univ. of Illinois Press, Urbana, 1966
25. C. Nichitiu and E. Remila. "Simulations of Graph Automata" Proc. MFCS'98 Satellite Workshop on Cellular Automata, Brno, Czech Republic, Aug. 1998
26. P. Orponen. "On the computational complexity of discrete Hopfield nets", Proc. 20th Int'l Colloqium on Automata, Languages and Programming (ICALP '93), Springer LNCS series, vol. 700, pp. 215 - 226, 1993
27. P. Orponen. "Computational complexity of neural networks: a survey", *Nordic J. Comp.,* vol. 1 (1), pp. 94 - 110, 1994
28. P. Orponen. "Computing with truly asynchronous threshold logic networks", *TCS,* vol. 174 (1-2), pp. 123 - 136, 1997
29. J. C. Reynolds. *"Theories of Programming Languages",* Cambridge Univ. Press, 1998
30. C. Reidys. "On acyclic orientations and sequential dynamical systems", *Advances Appl. Math.,* vol. 27, 2001
31. R. Sethi. *"Programming Languages: Concepts and Constructs"* (2nd ed.), Addison-Wesley, 1996
32. K. Sutner. "Computation theory of cellular automata", MFCS98 Satellite Workshop on CA, Brno, Czech Rep., 1998
33. P. Tosic. "A Perspective on the Future of Massively Parallel Computing: Fine-Grain vs. Coarse-Grain Parallel Models", Proc. 1st ACM Conf. on Computing Frontiers (CF'04), Ischia, Italy, April 2004
34. P. Tosic, G. Agha. "Concurrency vs. Sequential Interleavings in 1-D Threshold Cellular Automata", APDCM Workshop within Int'l Parallel & Dist. Processing Symp. (IPDPS), Santa Fe, New Mexico, USA, April 2004 (in Proc. IEEE IPDPS '04)
35. P. Tosic, G. Agha. "Characterizing Configuration Spaces of Simple Threshold Cellular Automata", ACRI'04, Amsterdam, The Netherlands, Oct. 25-28, 2004; in Springer-Verlag LNCS series, vol. 3305, pp. 861 - 870
36. S. Wolfram. "Twenty problems in the theory of CA", *Physica Scripta,* vol. 9, 1985
37. S. Wolfram (ed.). *"Theory and applications of CA",* World Scientific, Singapore, 1986
38. S. Wolfram. *"Cellular Automata and Complexity (collected papers)",* Addison-Wesley, 1994
39. S. Wolfram. *"A New Kind of Science",* Wolfram Media, Inc., 2002