ON PARALLEL vs. SEQUENTIAL THRESHOLD CELLULAR AUTOMATA

PREDRAG TOSIC* and GUL AGHA

Open Systems Laboratory (http://osl.cs.uiuc.edu), Department of Computer Science, University of Illinois at Urbana-Champaign, 201 N. Goodwin, Urbana, IL 61801, USA
{*p-tosic, agha*}*@cs.uiuc.edu*

ABSTRACT

Cellular automata (CA) *are an abstract model of* fine-grain parallelism, *as the node update operations are rather simple, and therefore comparable to the basic operations of the computer hardware. In a classical* CA*, all the nodes execute their operations in parallel and* in perfect synchrony*. We consider herewith the sequential version of* CA*, called* SCA*, and compare these* SCA *with the classical,* parallel CA*. In particular, we show that there are* 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, the fine granularity of the basic* CA *operations and, therefore, the* fine-grain parallelism *of the classical, synchronous* CA*, insofar as the "interleaving semantics" is concerned, is* not fine enough*. We also share some thoughts on how to extend the results herein, and, in particular, we try to motivate the study of* genuinely asynchronous cellular automata*.*

Keywords: cellular automata, linear threshold automata, dynamical systems, concurrency, sequential interleaving semantics

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 [19]. 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., [11, 26, 27, 28, 29]). However, *CA* can also be viewed as an abstraction of massively parallel computers (e.g, [8]). 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 the *CA* parallelism, i.e., the perfect synchrony of the classical *CA* computation.

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,* whether finite or infinite, 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. While the result would be trivial if one considers a single *(S)CA* computation, we argue that the result is nontrivial and important when applied to *all possible inputs* (starting configurations) and, moreover, to the entire classes of *CA* and *SCA.* Hence, the granularity of the basic *CA* operations, insofar as the (im)possibility of simulating

*Contact author. Work phone: 217-244-1976. Fax: 217-333-9386.

their concurrent computation via appropriate sequential interleavings of these basic operations, turns out not to be quite *fine enough.* We also share some thoughts on how to extend the results presented herein, 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., 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 [14, 15]. 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) *(A)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 (see *Section 4*).

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 *Subsection 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 *Chapter §12* of [22], "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 [22]. In contrast, [20] 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". [20] 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 a *perfect synchrony.*

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., [7, 9, 13, 17, 18]).

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[a],

[a] That is, for all "believers" in the interleaving semantics of concurrency - as contrasted with, e.g., proponents of

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 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 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, 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 the interleavings that allow "no-ops" for some of the instructions) is guaranteed to capture the concurrent behavior, i.e., so that (2) holds. That is, sometimes *refining the granularity of operations* so that sequential interleavings can capture synchronous parallel behavior, becomes a *necessity.*

---

true concurrency, an alternative model not discussed herewith.

We address herein the (in)adequacy of the sequential interleavings semantics when applied to *CA* where the individual node updates[b] 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 (*Section 2*), to fail rather dramatically even in the context of the simplest (nonlinear) CA node update rules.

2. Cellular Automata and Types of Their Configurations

We introduce *CA* by first considering *(deterministic) Finite State Machines (FSMs)* such as *Deterministic Finite Automata (DFA)*. 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 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 *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 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 (for more, see, e.g., [8]).

We follow [8] 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 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.

[b] It is tacitly assumed here 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.*

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 what 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 (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[c]. 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 (even in a single parallel step). In particular, the notion of independence between parallelism and concurrency as defined in [22] 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 *FSM* - 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 [24]. 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 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 (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" to any "memory cell" (or 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 herein 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 labeling 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.

[c]See the discussion in *Section 1,* and, in particular, the definition of the relationship between concurrency and parallelism in reference [22].

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, 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* possible sequences of node updates. For that purpose, the following class of sequential 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 $s : \{1, 2, 3, ...\} \rightarrow L$, where L is the set of integer labels of the nodes in Γ.

We now change 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 system* view of *CA* is helpful. A *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 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* 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 (see *Subsection 3.1*).

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 *Def. 5*). In particular, in case of the *NICA* automata, a single configuration can simultaneously be a weak FP, a CC and a TC; see *Subsection 3.1* for a simple example.

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

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 section. Discussion of the implications and relevance of these results, as well as some possible generalizations and extensions, will follow in *Section 4*.

Herein, we 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, viz., 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., [8] or [28]). We show the fundamental difference in the configuration spaces, and therefore possible computations, between the parallel threshold 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 underlying cellular spaces Γ) they 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 classical *CA* based on perfect synchrony, we restrict the underlying cellular spaces to *one-dimensional* Γ. We formally define the class of *1-D (S)CA* of a finite radius below:

Definition 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 any 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 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 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, 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", 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 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. Synchronous Parallel CA vs. Sequential Interleavings CA: A Simple 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 *CA* on 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 the two inputs. The two phase spaces are given in *Fig. 1.*

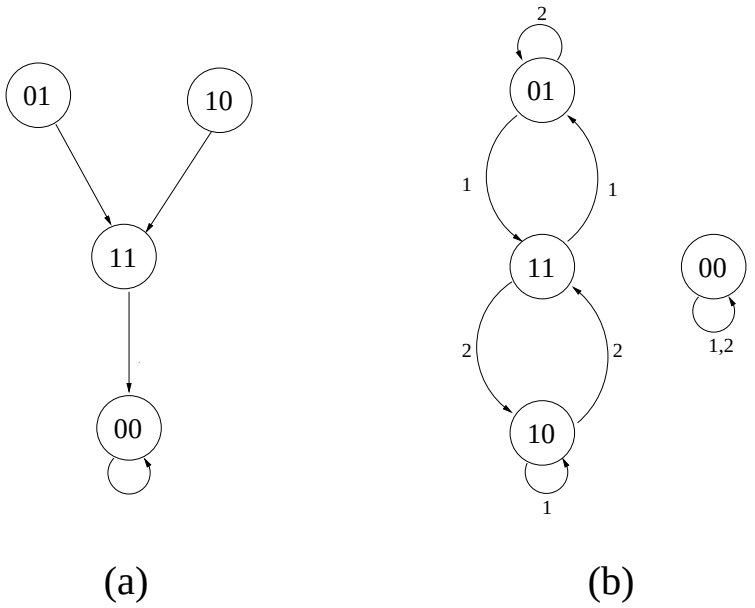

Figure 1:
Configuration spaces for two-node (a) parallel and (b) sequential cellular automata, respectively. Each node computes the logical XOR function of its own current state, and that of the other node. In (b), the integer labels next to the transition arrows indicate which node, 1 or 2, is updating and thus causing the indicated global state transition.

In the parallel case, the state 00 is the "sink", and the entire configuration space is as in *Fig. 1 (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.

In the case of sequential node updates, 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 any "typical" (infinite) sequence of node updates, is going to be revisited *infinitely often.* In fact, for some sequences 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 *Fig. 1 (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 *any* sequence of node updates[d].

Some observations are in order. First, overall, the configuration space of the *XOR NICA* is richer than the *PS* of its parallel counterpart. In particular, due to determinism, any FP state of a classical *CA* is necessarily a stable attractor or "sink". 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 cycles), the phase space of the corresponding *NICA* has nontrivial finite temporal cycles.

[d] In [25] 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 *GE* in discrete dynamical systems, the reader is referred to [3, 4].

On the other hand, the union of all possible sequential computations ("interleavings") cannot fully capture the concurrent computation, either: consider, for example, *reachability* of the state 00.

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 cycles in the parallel case, as well. Hence, for *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 trivial example of *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 *Subsection 3.2* and *Section 4*) *SCA* and, consequently, *NICA,* are capable of.

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

We shall now compare and contrast the classical, concurrent and perfectly synchronous *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 *symmetric linear threshold functions.* This will be done by studying the configuration space properties, that is, the possible computations, of the simple threshold automata in the parallel and sequential settings.

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

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[e] called *weights.*

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

Therefore, given an integer k, a *k-threshold function*, in general, is any function of the form as in *Def. 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.*

[e] In general, w_i 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., [3, 4, 26, 27].

Definition 9 A *simple* threshold *(S)CA* is an 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 1-D threshold *(S)CA* studied in the sequel will have the node update functions of the general form

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

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 the k-threshold *(S)CA* on a specified number of nodes in this case is just the 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.

Due to the nature of the node update rules, cyclic behavior intuitively should not be expected in such simple threshold 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 exist, and in the case of the classical, parallel (i.e., synchronous) *CA only*; and, related to that observation,
- it is, for this class of 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 two-way infinite *1-D (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 1-D parallel CA with $r = 1$, the MAJORITY update rule, and an even number of nodes, has finite temporal cycles in the phase space (PS); the same holds for two-way infinite 1-D MAJ CA.

(ii) 1-D Sequential CA with $r = 1$ and the MAJORITY update rule do not have any temporal cycles in the phase space, 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 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. Consider either an infinite *1-D 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.

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. 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 not fixed, $010 \rightarrow^{\star} \{000, 110, 011, 111\}$. However, $010 \notin \{000, 110, 011, 111\}$. By symmetry, similar analysis holds for 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.

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* of the input bits, such sequential *CA* still do not have any proper cycles.

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

Proof.

Since $r = 1$ and $2r + 1 = 3$, there are only *five* Monotone Symmetric Boolean (*MSB*) functions, or, 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 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 *MSB* 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 also hold for *1-D CA* with radius $r = 2$:

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

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

(ii) Any 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.

Proof.

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

The argument in (ii) is similar to that of *Lemma 1 (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 1 and 2, part (i), we have the following

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

Namely, given any $r \geq 1$, a (classical, concurrent) *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 *threshold CA* defined on finite 1-D cellular spaces, provided that such automata have sufficiently many nodes, that the number of nodes is appropriate (see [25] for more details), and assuming circular boundary conditions (i.e., assuming that Γ is a sufficiently big finite ring). 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 *2D Cartesian grids* and *Hypercubes* have two-cycles in their respective phase spaces.

More generally, for any underlying cellular space Γ 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 Γ 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 *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 [8, 10]:

Proposition 1 *Let a classical, 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* symmetric threshold automaton, 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 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 and synchronous dynamical systems; see, e.g., [3, 4, 5, 6]. 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 10 An infinite sequence $s : N \rightarrow L$ 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 $s(1), s(2), s(3), ...$

Now that we have defined what we mean by a *single step* of a sequential *CA,* as well as adopted some reasonable notion[f] of *fairness,* we can now state the following generalization of *Proposition 1* to both finite and infinite *1-D CA* and *1-D SCA:*

Proposition 2 *Let a parallel CA or a sequential SCA be defined over a finite or infinite 1-D cellular space (that is, a line or a ring), with a finite rule radius* $r \geq 1$. *Let this automaton's local update rule be an elementary symmetric threshold function. Let's also assume, in the sequential cases, that the fairness condition from* Def. 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* initial configuration[g] necessarily (and *quickly* [25]) converges to either a FP or a two-cycle [10]:

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* $F^{t+1}(C) = F^t(C)$.

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 (stable) fixed point.

To summarize, if the computation of a *SCA* starting from some configuration C 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 *Prop. 2,* two basic facts have to be established. One, convergence to finite temporal cycles of length three or higher is not possible. Indeed, *Prop. 1* establishes 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 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

[f] Our notion of fairness in *Def. 10* 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 *Prop. 2* in the main text.

[g] Also sometimes called *compact support;* see, e.g., [10]. 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.

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 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 [4] for a class of sequential graph automata called *Sequential Dynamical Systems (SDS),* 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 2* below and its proof are directly based on [4]:

Theorem 2 *The following dichotomy holds:*

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

(ii) Any 1-D SCA *with* any *monotone symmetric Boolean update rule* δ, *any finite* $r \geq 1$, *defined over any (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 $\{(1^r 0^r)\}^\omega$, $(0^r 1^r)^\omega$, with the obvious modification in case of finite *CA* with n nodes (for n even, and assuming circular boundary conditions).

Part (ii) (proof sketch): The proof of this interesting property is based on a slight modification of a similar result in [4] for a class of the sequential graph automata called *Sequential Dynamical Systems (SDS).* A *simple symmetric SDS* is an SDS 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, 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 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 this configuration) is always nonnegative, 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 (and, therefore, sequential steps), an equilibrium where no nodes can further flip is reached; this equilibrium will be a fixed point configuration. Due to space considerations, we do not provide all the details. Instead, we again refer the reader to [4] for a full, rigorous proof of the same property as in our claim herein, only in a slightly different setting - the difference being immaterial insofar as the validity of the claim and the applicability of the just outlined proof idea based on the decreasing configuration potentials are concerned.

[h] 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, however, for the sake of clarity and space constraints.

□

To summarize, symmetric 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 (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 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 the nodes in $\Gamma(A)$.

The 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"[i]). Hence, we conclude that no choice of a "sequential interleaving" can 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 *CA* even for this, simplest nonlinear class of totalistic *CA* update rules.

4. Discussion and Future Directions

The results in *Section 3* show that, even for the very simplest (nonlinear, nonaffine) totalistic cellular automata, sequential nondeterminism - that is, the union of all possible sequential interleavings - dramatically fails to capture concurrency. It is not surprising that one can find a concurrent *CA* such that no sequential *CA* with the same underlying cellular space and the same node update rule can reproduce identical or even "isomorphic" computation, as the example at the beginning of *Section 3* clearly shows (see *Fig. 1* and the related discussion). However, we find it rather interesting that very profound differences can be observed even in the case of extremely simple parallel and sequential *CA* - that is, the one-dimensional automata with small r and simple threshold functions as the node update rules - and that this profound difference does not apply merely to individual *(S)CA* and/or their particular computations, but to *all* possible computations of an entire, relatively broad class of the *CA* update rules.

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 *Subsection 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 cycles can be shown to be very few, and without any incoming transients [25]. 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 *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 (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

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

systems modeled by the threshold *CA* to those properties that are *statistically robust* (see, e.g., [1]). It can be readily argued in a rigorous, probabilistic sense that, again, the typical, statistically robust behavior of a typical threshold *CA* computation is a relatively short transient chain, followed by convergence to a stable fixed point. In particular, the non-fixed-point temporal cycles of the threshold *CA* not only utterly 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*.

After these digressions on the meaning and implications of our results on the 1-D threshold parallel and sequential *threshold CA,* we now 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. Another direction of future inquiry is to consider *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 [3, 4]. We have also recently obtained some general results, similar in spirit to those in *Section 3* herein, for the parallel and sequential threshold automata defined over arbitrary *bipartite graphs*: such graph automata, if the nodes are updated concurrently, also in general do contain nontrivial cycles even in case of the simplest node update rules such as *MAJORITY*, yet no cycles are possible if the nodes are updated sequentially and any monotone symmetric node update rule is used.

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, appears an interesting problem to consider.

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 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.

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., [3, 4, 6, 21] and references therein. It would be interesting, therefore, to consider *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 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., [14, 15]).

What are, then, such *genuinely asynchronous CA* 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 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

$$x_i^{t+1} = f(x_i^t, x_{i_1}^t, ..., x_{i_{2r}}^t) \tag{7}$$

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

$$x_i^{t+1} = f(x_i^t, x_{i_1}^{t_1}, ..., x_{i_{2r}}^{t_{2r}}) \tag{8}$$

We observe that t in *Eqn.* (7) 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.* (8), 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[j]? Yet, if we want to study *genuinely* asynchronous *CA* models (rather than arbitrary sequential models with global clocks), these changes in the definition seem unavoidable.

We point out that this, genuine (that is, communication) asynchrony in *CA* (see *Eqn.* (8)) 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 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 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 our 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 (proper, stable) 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 [3] on the FP invariance for sequential and synchronous (concurrent) graph automata are just special cases of this, more general result.

Theorem 3 *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 qualitative comparison-and-contrast of the asynchronous *CA* that we propose, and the classical *CA, SCA* and *NICA.* Such study would shed more light on those behaviors that are solely due to (our abstracted version of) network delays.

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

More generally, communication asynchronous *CA*, i.e., the various nondeterministic choices for a given 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 (Γ, N, M). In particular, the nondeterminism that arises from (unbounded) asynchrony subsumes the nondeterminism of a kind studied in *Section 3*; but the question arises, exactly how much more expressive the former model really is than the latter.

5. Summary and Conclusions

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 very simplistic cases, this sequential "interleaving semantics" of *NICA* fails to capture concurrency of the classical *CA*. One 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:

- Fetching all the neighbors' values ("Receiving"? "Reading shared variables"?)
- Updating one's own state according to the update rule δ (that is, performing the local computation)
- 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 these 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,* 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.

Acknowledgments: The work presented herein was supported by the *DARPA IPTO TASK Program*, contract number *F30602-00-2-0586*. Many thanks to Reza Ziaei (UIUC) for several useful discussions.

References

1. W. Ross Ashby, "Design for a Brain", Wiley, 1960
2. C. Barrett and C. Reidys, "Elements of a theory of computer simulation I: sequential CA over random graphs", Applied Math. and Computation, vol. 98 (2-3), 1999
3. 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. Proc. AA (DM-CCG), July 2001
4. 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", TCS vol. 295, issues 1-3, pp. 41-64, Feb. 2003
5. C. Barrett, H. Mortveit, and C. Reidys, "Elements of a theory of computer simulation II: sequential dynamical systems", Applied Math. and Computation, vol. 107 (2-3), 2000
6. C. Barrett, H. Mortveit, and C. Reidys, "Elements of a theory of computer simulation III: equivalence of sequential dynamical systems", Applied Math. and Computation, vol. 122 (3), 2001
7. 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
8. Max Garzon, "Models of Massive Parallelism: Analysis of Cellular Automata and Neural Networks", Springer, 1995
9. 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
10. E. Goles, S. Martinez, "Neural and Automata Networks: Dynamical behavior and Applications", Math. and Its Applications series (vol. 58), Kluwer, 1990
11. E. Goles, S. Martinez (eds.), "Cellular Automata and Complex Systems", Nonlinear Phenomena and Complex Systems series, Kluwer, 1999
12. Howard Gutowitz (ed.), "Cellular Automata: Theory and Experiment", The MIT Press / North-Holland, 1991
13. C. A. R. Hoare, "Communicating Sequential Processes", Prentice Hall, 1985
14. T. E. Ingerson and R. L. Buvel, "Structure in asynchronous cellular automata", Physica D: Nonlinear Phenomena, Volume 10, Issues 1-2, January 1984
15. S. A. Kauffman, "Emergent properties in random complex automata" (ibid.)
16. Robin Milner, "A Calculus of Communicating Systems", Springer-Verlag Lecture Notes in Computer Science (LNCS), 1980
17. Robin Milner, "Calculi for synchrony and asynchrony", Theoretical Computer Sci. 25, pp. 267-310, 1983
18. Robin Milner, "Communication and Concurrency", Prentice-Hall, 1989
19. John von Neumann, "Theory of Self-Reproducing Automata", edited and completed by A. W. Burks, Univ. of Illinois Press, Urbana, 1966
20. J. C. Reynolds, "Theories of Programming Languages", Cambridge Univ. Press, 1998
21. C. Reidys, "On acyclic orientations and sequential dynamical systems", Adv. Appl. Math. vol. 27, 2001
22. Ravi Sethi, "Programming Languages: Concepts & Constructs", 2nd ed., Addison-Wesley, 1996

23. K. Sutner, "Computation theory of cellular automata", MFCS98 Satellite Workshop on CA, Brno, Czech Rep., 1998
24. P. Tosic, "A Perspective on the Future of Massively Parallel Computing: Fine-Grain vs. Coarse-Grain Parallel Models", Proc. ACM Computing Frontiers '04, Ischia, Italy, April 2004
25. P. Tosic, G. Agha, "Characterizing Configuration Spaces of Simple Threshold Cellular Automata", ACRI'04, Amsterdam, The Netherlands, Oct. 25-28, 2004 (to appear in a Springer-Verlag LNCS volume)
26. S. Wolfram "Twenty problems in the theory of CA", Physica Scripta 9, 1985
27. S. Wolfram (ed.), "Theory and applications of CA", World Scientific, Singapore, 1986
28. Stephen Wolfram, "Cellular Automata and Complexity (collected papers)", Addison-Wesley, 1994
29. Stephen Wolfram, "A New Kind of Science", Wolfram Media, Inc., 2002