Concurrency vs. Sequential Interleavings in 1-D Threshold Cellular Automata

Predrag Tosic*, Gul Agha
Open Systems Laboratory, 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, that is,* (logically) simultaneously. *We consider herewith the sequential version of* **CA***, or* **SCA***, and compare it 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. While the result is trivial if one considers a single computation on a chosen input, we find it both nontrivial, and having some important and far-reaching implications when applied to all possible inputs and, moreover, to the entire nontrivial classes of* **CA (SCA)***. We also share some thoughts on how to extend our results herein, and we try motivate the study of* genuinely asynchronous cellular automata.

1. Introduction and Motivation

Cellular automata (CA) were originally introduced as abstract mathematical models that can capture, at a high level, the behavior of biological systems capable of self-reproduction [15]. Subsequently, *CA* have been extensively studied in a great variety of application domains, but mostly in the context of physics and, more specifically, of studying complex (physical or biological) systems and their dynamics (e.g., [20-22]). However, *CA* can also be viewed as an abstraction of massively parallel computers (e.g, [7]). Herein, we study a particular simple yet nontrivial class of *CA* from a computer science perspective. This class are the *threshold cellular automata*. We pose (and partially answer) some fundamental questions regarding the nature of the *CA* parallelism, i.e., the concurrency of the classical *CA* computation; the analysis is done in the context of *threshold CA*.

Namely, it is well known that *CA* are an abstract computational model of *fine-grain parallelism* [7], in that the elementary operations executed at each node are rather simple and hence comparable to the basic operations performed by the computer hardware - yet, due to interaction and synergy among a (typically) great number of these nodes, many *CA* are capable of highly complex behaviors (i.e., computations). In a classical (parallel) *CA*, whether finite or infinite, all the nodes execute their operations *logically simultaneously*: in general, the state of node x_i at time step $t+1$ is some simple function of the states of node x_i itself, and a set of its pre-specified neighbors at time t.

We consider herewith the sequential version of *CA*, or *SCA*, and compare it with the classical, *parallel* (*concurrent*) *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. While the result is trivial if one considers a single *CA*, we find the result quite nontrivial, important and with some far-reaching implications when applied to the entire classes of *CA* and *SCA*. Hence, granularity of the basic *CA* operations, insofar as the ability to simulate their concurrent computation via appropriate nondeterministic sequential interleavings of these basic operations, turns out not to be quite *fine enough*, as we prove that no such analogue of the sequential interleaving semantics applied to concurrent programs of classical *CA* can capture even rather simplistic parallel *CA* computations. We also share some thoughts on how to extend the results presented herein, and, in particular, we try motivate the study of *genuinely asynchronous cellular automata*, where asynchrony applies not only to the local computations at individual nodes, but also to *communication* among different nodes (via "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 the case when, for instance, the individual nodes update one at the time, according to some random order. This is a kind of asynchrony found in the literature, e.g., in [10]. 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 §4).

Before dwelling into the issue of parallelism vs. arbitrary sequential interleavings applied to threshold cellular automata, we first clarify the terminology, and then introduce the relevant concepts through a simple programming exercise in §§1.1.

An important remark is that we use the terms *parallel* and *concurrent* as synonyms throughout the paper. This is perhaps not the most standard convention among the researchers of programming languages semantics and semantic models of concurrency (e.g., [17], [16]), but we are not alone in not making the distinction between the two notions (cf. discussion in [16]). Moreover, by a *parallel* (equivalently, *concurrent*) *computation* we shall mean actions of several processing units that are carried out *logically* (if not necessarily *physically*) *simultaneously*. In particular, when referring to parallel (or, equivalently, concurrent) computation, we do assume a perfect synchrony. This approach is primarily motivated by the nature of *CA* "hardware" and the way classical *CA* compute.

1.1. Capturing concurrency by nondeterministic sequential interleavings

While our own brains are massively parallel computing devices, we seem to think and function rather sequentially. Indeed, human mind's approach to problem solving is usually highly sequential. In particular, when designing an algorithm or writing a computer program that is inherently parallel, we prefer to be able to understand such an algorithm or program in the sequential terms. It is not surprising, therefore, that since the very beginning of the design of parallel algorithms and parallel computation formalisms, 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 [14].

When interpreting concurrency via interleaving semantics, a natural question arises: *Given a parallel computing model, can its concurrent 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?* The answer is *"Yes"*, For most theoreticians of parallel computing (that is, all the "believers" in interleaving semantics as contrasted with, e.g., proponents of *true concurrency*, an alternative model not discussed herewith), 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, it need not always be clear, how do we tell, given a parallel computation in the form of a set of concurrently executing instructions or processes, 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 purpose 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.

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

Sequentially, one *always* gets the same answer: $x = 2$. In parallel (that is, if the two assignment operations to the same variable x are done concurrently), 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" is 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 exists a choice of a *sequential interleaving* of the six machine instructions above that leads to *"parallel"* behavior (i.e., the one where, after the code is executed, $x = 1$); in fact, there are several such permutations of instructions. Thus, by refining granularity from a high-level language instructions down to machine instructions, we can certainly preserve the interleaving "semantics" of concurrency, as the high-level language "concurrent" computation can be perfectly well understood in terms of the sequential interleavings of computational operations at the level of assembly language instructions.

2. Cellular Automata and Types of Their Configurations

We introduce *classical 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 input signals coming from the outside. The machine is initially in some starting state; upon reading each input signal (a single binary symbol, in the standard *DFA* case), 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 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 the set of the possible values held in each node (typically, this set is $\{0, 1\}$), and we also identify this set of values with the set of the 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, 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, and, moreover, are actually strictly more powerful than classical Turing machines (e.g., [7]).

More formally, we follow [7] and define classical (that is, synchronous and concurrent) *CA* in two steps: by first defining the notion of a *cellular space*, and subsequently that of a *cellular automaton* defined over an appropriate cellular space.

Definition 1: *Cellular Space* Γ is an ordered pair (G, Q) where G is a regular graph (finite or infinite), 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.

Definition 2: *Cellular Automaton* A is an ordered triple (Γ, N, M) where:
- Γ is a *cellular space*;
- N is a *fundamental neighborhood*;
- 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 local transition rule δ specifies how each node updates, based on its current value and that of its neighbors in N. By composing local transition rules for all nodes together, we obtain *the global map* on the set of (global) configurations of a cellular automaton.

Assuming a large number of nodes, there is plenty of potential for parallelism in the *CA* hardware. 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). Roughly, the underlying cellular space corresponds to the *CA* "hardware", whereas the *CA* "software" or program is given by the local update rule δ. The global evolution (or, analogously, massively parallel computation) of a *CA* is then obtained by the composition of the effects of the local node update rule to each of the nodes.

We now change pace and introduce some terminology borrowed from physics that we find appropriate and useful for characterizing *all possible computations* of a parallel or sequential *CA*. To this end, a *(discrete) dynamical system* view of *CA* is helpful. A *phase space* of a dynamical system is a (finite or infinite) directed graph where the vertices are the *global configurations* (or *global states*) of the system, and directed edges correspond to possible 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. 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 3: 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, if once reached, will be revisited infinitely often with a fixed, finite 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 CC with period 1. Due to deterministic evolution, any configuration of a classical, parallel *CA* is either a FP, a proper CC, or a TC.

3. 1-D CA vs. SCA 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 their proofs, are given in this section; discussion of the implications and relevance of these results, as well as the possible generalizations and extensions, will follow in *Section* §4.

Herein, we compare and contrast the classical, concurrent *CA* with their sequential counterparts, *SCA*, 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 *CA* are also *totalistic* (see, e.g., [7] or [22]). We show the fundamental difference in the configuration spaces, and therefore possible computations, in case of the classical, concurrent automata on one, and the sequential threshold cellular automata, on the other hand: while the former can have temporal cycles (of length two), the computations of the latter, under some mild additional conditions whose sole purpose is to ensure *some form of convergence*, *always* converge to a fixed point.

We fix the following conventions and terminology. Throughout, only *Boolean CA* and *SCA* are considered; in particular, the set of possible states of any node is $\{0, 1\}$. The terms "monotone symmetric" and "symmetric (linear) threshold" functions/update rules/automata are used interchangeably; similarly, "(global) dynamics" and "computation", when applied to any kind of an automaton, 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*. Also, unless explicitly stated otherwise, *CA (SCA) with memory* are assumed, and the default cellular space is a two-way infinite line. Moreover, all the underlying cellular spaces throughout the next two subsections are (finite or infinite) lines or rings.[1] The terms "phase space" and "configuration space" are used synonymously, as well, and sometimes abridged to *PS* for brevity.

3.1. A simple motivating example

A *1-D cellular automaton of radius r* is a *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 (and, in case of *CA with memory*, on the current state of that node itself). In case of a *Boolean CA with memory*, therefore, each node's next state depends on $2r + 1$ input bits, while in the *memoryless case*, the local update rule is a function of $2r$ input bits. The string of nodes can be a finite line graph, a ring (corresponding to "circular boundary conditions"), a one-way infinite string, or, in the most common case one finds in the literature, the cellular space is a two-way infinite string.

We compare and contrast the qualitative properties of configurations spaces, and therefore dynamics or possible computations, of the classical, parallel *CA* versus the dynamics (computations) of *SCA*. Sequential cellular automata (*SCA*) and their generalizations to non-regular (finite) graphs have been already studied in the context of a formal theory of computer simulation (see, e.g., [2-6]).

There are plenty of simple, even trivial examples where not only are concrete computations of parallel *CA* from particular initial configurations different from the corresponding computations of the sequential *CA*, but actually the entire configuration spaces of the *parallel CA* and the corresponding *SCA* are structurally 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 Boolean *XOR* of the two inputs (viz., of the node's own current state, and that of its only neighbor). The two phase spaces are given in *Fig. 1*. In (b), since the corresponding automaton is *nondeterministic*, the numbers next to the transition arrows indicate which node, 1 or 2, is updating and thus causing the indicated global state transition from the current state.

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, the fixed point "sink" state, that is, in the language of *nonlinear dynamics*, a stable global attractor, will be reached.

In case of the sequential node updates, the configuration 00 is still a FP but, this time, it is not reachable from any other configuration. Moreover, there are two more (unstable) pseudo-fixed points, 01 and 10, and two temporal two-cycles. In particular, while all three states, 11, 10 and 01, are *transient states* in the parallel case, sequentially, each of them, for any "typical" sequence of node updates, is going to be revisited, either as a pseudo-FP or a cycle state. 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* pseudo-fixed point states and cycle states. The phase space capturing all possible sequential computation is given in *Fig. 1 (b)*.

Several observations are in order. First, overall, the sequential configuration space seems richer than its parallel counterpart. In particular, whereas, due to determinism, any FP state of a classical *CA* is necessarily a stable attractor or

[1] We have already generalized some of the results that follow to more general cellular spaces, but, for the reasons of conciseness and clarity of exposition, those results will not be discussed herein.

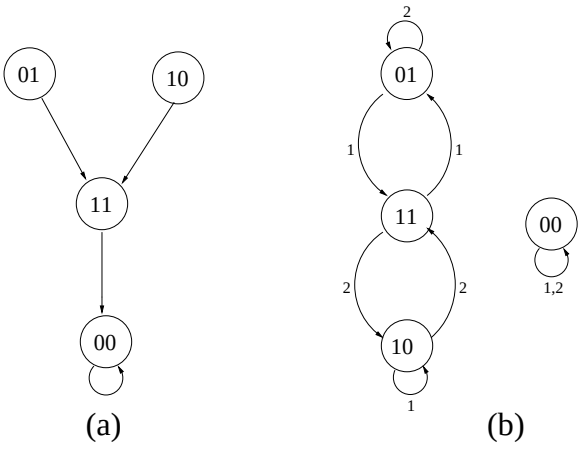

Figure 1. Configuration spaces for two-node (a) parallel and (b) sequential *XOR CA*, resp.

a "sink" (in the terminology of *complex dynamics*), and, in particular, once a FP configuration is reached, all the future iterations stay there, in case of different possible sequential computations on the same cellular space, pseudo-fixed points clearly need not be stable. Also, whereas the phase space of a parallel *CA* is cycle-free (if we do not count FPs as "proper" cycles), the phase space of the corresponding *SCA* has nontrivial temporal cycles. On the other hand, the union of all possible sequential computations ("interleavings") cannot fully capture the concurrent computation, either: e.g., consider *reachability* of the state 00.

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 parallel *CA* and sequential *SCA* are capable of, are quite comparable. Moreover, in cases when one is richer than the other, it seems reasonable that *SCA* will be capable of more interesting computations than parallel *CA*, given the nondeterminism due to all the possibilities for node update sequences.

This detailed discussion of a trivial example of the *(S)CA* phase spaces has the main purpose of motivating what is to follow: an entire class of *CA* and *SCA*, with the node update functions even simpler than *XOR*, yet for which, irrespective of the number of nodes, the boundary conditions and other details, it is the concurrent *CA* that are *provably* capable of computations that no corresponding (or similar, in a sense to be defined below) *SCA* are capable of.

3.2. On the existence of cycles in threshold CA and SCA

Now we consider the threshold automata in parallel and sequential settings.

Definition 4: A *threshold automaton (threshold CA)* is a (parallel or sequential) cellular automaton where δ is a (herein, Boolean-valued) *linear threshold function*.

Herein, we also assume δ to be a *symmetric function* of all of its inputs.

Due to the nature of the node update rules, cyclic behavior intuitively should not be expected in such 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: (i) the local update rules are the simplest nonlinear totalistic rules one can think of; (ii) given the rules, the cycles are not to be expected - yet they exist, and in case of classical, parallel *CA only*; and, related to that observation, (iii) it is, for this class of *(S)CA*, the parallel *CA* that exhibit the more interesting behavior than sequential *SCA*, and, in particular, while there is nothing (qualitatively) among the possible sequential computations that is not present in the parallel case, the classical parallel threshold *CA* are capable of "oscillatory/non-converging computations" - they may have nontrivial temporal cycles - that cannot be reproduced by any threshold *SCA*.

The results below hold for two-way infinite *1-D CA*, as well as for finite *CA* and *SCA* with sufficiently many nodes and circular boundary conditions (i.e., for *(S)CA* whose cellular spaces are finite rings).

Lemma 1:
(i) A 1-D classical (i.e., parallel) *CA* with $r = 1$ and the *MAJORITY* update rule has (finite) temporal cycles in the phase space (*PS*).

(ii) *1-D Sequential CA* with $r = 1$ and the *MAJORITY* update rule do not have any (finite) cycles in the phase space, *irrespective* of the sequential node update order ρ.

Remark: In case of infinite sequential *SCA* as in the Lemma above, a nontrivial cycle configuration does not exist even in the limit. We also note that ρ is an arbitrary sequence of an *SCA* nodes' indices, not necessarily a (finite or infinite) permutation.

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 010 is 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 is a fixed point. Similar analysis applies to 001 and 100. Hence, no local neighborhood $x_1x_2x_3$, once changed, can ever "come back". Therefore, there are no proper cycles in Sequential 1-D *CA* with $r = 1$ and δ = *MAJORITY*.

Part (ii) of the Lemma 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 sequential update order ρ, the phase space *PS(A)* of the automaton A is cycle-free.

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

Lemma 2:

(i) *1-D (parallel) CA* with $r = 2$ and with the *MAJORITY* node update rule have (finite) cycles in the phase space.

(ii) Any *1-D SCA* with *MAJORITY* node update rule, $r = 2$ and any sequential order on node updates has a cycle-free phase space.

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 *threshold automaton*) A such that A has (finite) cycles in the phase space.

Namely, given any $r \geq 1$, a (classical, concurrent) *CA* with $\delta =$ *MAJORITY* 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 *threshold CA (SCA)* defined on finite 1-D cellular spaces, provided that such automata have sufficiently many nodes and assuming circular boundary conditions (i.e., assuming Γ is a sufficiently big finite ring). Moreover, the result extends to finite and infinite *CA* in higher dimensions, as well; in particular, *2D rectangular grid CA* and *Hypercube CA* with $\delta =$ *MAJORITY* (or another nontrivial symmetric threshold update rule) 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) *threshold parallel CA* have temporal two-cycles.

It turns out that the two-cycles in the *PS* of concurrent *CA* with $\delta =$ *MAJORITY* 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 or infinite) regular Cayley graph as the underlying cellular space, the following general result holds [7], [8]:

Proposition 1: Let a classical *CA* $A = (\Gamma, N, T)$ be such that T is an elementary symmetric threshold local update rule applied to a finite cellular space Γ. Then for all configurations $C \in PS(A)$, there exists $t \geq 0$ such that $T^{t+2}(C) = T^t(C)$.

In particular, this result implies that, in case of any (finite) monotone symmetric automaton, for any starting configuration C_0, there are only two possible kinds of orbits: upon repeated iteration, after finitely many steps, the computation either converges to a fixed point configuration, or else one 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 at all, it will 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.[2]

To summarize, symmetric linear threshold 1-D *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* can never cycle; while for simplicity we have shown that this holds only for *SCA* with short-range interactions (small r), the result actually holds for (finite) *SCA* with arbitrary finite radii of interaction, $r \geq 1$. In particular, 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* is guaranteed to converge to a fixed point. Since this holds irrespective of the choice of the sequential update ordering (and, extending to infinite *SCA*, temporal cycles cannot be obtained even "in the limit", that is, via infinitely long computations, obtained

[2] Additionally, in order to ensure some sort of convergence of a given *SCA*, and, more generally, in order to ensure that, in some sense, *all* nodes get a chance to update their states, an appropriate condition that guarantees *fairness* may need to be specified. For finite *SCA*, one sufficient such condition is to impose a fixed upper bound on the number of sequential steps before any given node gets its "turn" to update. In infinite *SCA* case, the issue of fairness is nontrivial, and some form of *dove-tailing* of sequential individual node updates may need to be imposed; further discussion of this issue, however, is beyond our current scope.

by allowing arbitrary infinite sequences of individual node updates), we conclude that no choice of "sequential interleaving" can capture the concurrent computation. Consequently, the "interleaving semantics" of *SCA* fails to fully capture the concurrent behavior of classical *CA* even for this, simplest nonlinear class of totalistic *CA*, namely, the symmetric threshold cellular automata.

4 Discussion and Future Directions

The results in §3 show that, even for the very simplest (nonlinear) totalistic cellular automata, nondeterministic interleavings dramatically fail to capture concurrency. It is not surprising that one can find a (classical, 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 §3 clearly shows (see *Fig. 1* and the related discussion). However, we find it rather interesting that very profound differences (a possibility of looping vs. the guaranteed convergence to a fixed point configuration) can be observed in case of the simplest nonlinear 1-D parallel and sequential *CA* with symmetric 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 the possible computations of the entire class of the *(symmetric) threshold CA* update rules.

Moreover, the differences in parallel and sequential computations in case of the Boolean *XOR* local update rule can be largely ascribed to the properties of the *XOR* function. 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., in case of finite domains and converging computations, to always converge to a fixed point. This intuition about the monotone symmetric *SCA* is shown correct. It is actually, in a sense, (statistically) "almost correct" in case of the parallel *CA*, as well, in that the actual non-FP cycles can be shown to be very few, and without any incoming transients [19]. Thus, in this case, the very existence of the (rare) nontrivial temporal cycles can be ascribed directly to the assumption of *perfect synchrony* (i.e., effective simultaneity) of the parallel node updates.

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. Another obvious extension is to consider 2-D and other higher-dimensional threshold *CA*, as well as *CA* defined over regular Cayley graphs that are not simple Cartesian grids. It is also of interest to consider CA-like finite automata defined over *arbitrary* rather than only regular (finite) graphs. We already have some results along these two lines, but do not include them herein due to space constraints.

Another interesting extension is to consider classes of node update rules beyond the 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, 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* is r, then any change in the state of y can affect the state of x no sooner, but also and more importantly, *no later* than after about $\frac{k}{r}$ (parallel node update) computational steps. As the nodes all update "in sink", locally a classical *CA* is a perfectly synchronous concurrent system. However, globally, i.e., if one is interested in the interactions of nodes that are at a distance greater than r apart, then the classical *CA* and their various graph automata extensions are a class of models of *bounded asynchrony*.

We remark that two particular classes of graph automata defined over arbitrary (not necessarily regular, or Cayley) *finite* graphs, viz., sequential and synchronous dynamical systems (SDSs and SyDSs, respectively), and their various phase space properties, have been extensively studied; see, e.g., [3-6] 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 asynchrony in computation at different nodes (that is, beyond the arbitrary sequential node updates yet with respect to the global time) that has been studied since at least 1984 (e.g., [10], [11]).

We propose a broad study of the general phase space properties, and a qualitative comparison-and-contrast of the asynchronous *CA (ACA)* and the classical *CA* and *SCA*. Such a study could shed light on detecting computational behaviors that are solely due to asynchrony, that is, what can be viewed as an abstracted version of "network delays" in physically realistic (asynchronous) cellular automata. Communication asynchronous *CA*, i.e., various nondeterministic choices for a given *(A)CA* that are due to asyn-

chrony, can be shown to subsume all possible behaviors of classical and sequential *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 §3; but the question arises, exactly how much more expressive and powerful the former model really is than the latter.

5 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 concept to parallel *CA* and thus motivate the study of sequential cellular automata, *SCA*, and the comparison and contrast between *SCA* and classical, concurrent *CA*. Concretely, we show that, even in very simplistic cases, this sequential semantics fails to capture concurrency of classical *CA*. Hence, simple as they may be, the basic operations (local node updates) in classical *CA* cannot always be considered atomic. In particular, the fine-grain parallelism of *CA* turns out not to be quite fine enough when it comes to capturing concurrent execution via nondeterministic sequential interleavings of those basic operations. It then seems reasonable to consider a single local node update to be made of an ordered sequence of finer elementary operations: (1) fetching ("receiving"?) all the neighbors' values, (ii) updating one's own state according to the update rule δ, and (iii) making available ("sending"?) one's new state to the neighbors.

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

We also argue that various extensions of the basic *CA* model can 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.

Bibliography

[1] R. B. 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. & Comput., 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. & 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 1-3: 41-64, 2003

[5] C. Barrett, H. Mortveit, and C. Reidys, "Elements of a theory of computer simulation II: sequential dynamical systems", Applied Math. & Comput. 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", Appl. Math. & Comput. vol. 122(3), 2001

[7] M. Garzon, "Models of Massive Parallelism: Analysis of Cellular Automata and Neural Networks", Springer, 1995

[8] E. Goles, S. Martinez, "Neural networks: theory and applications", Kluwer, Amsterdam, 1990

[9] E. Goles, S. Martinez (eds.), "Cellular Automata and Complex Systems", Nonlinear Phenomena and Complex Systems series, Kluwer, Dordrecht, 1999

[10] T. E. Ingerson and R. L. Buvel, "Structure in asynchronous cellular automata", Physica D: Nonlinear Phenomena, vol. 10 (1-2), Jan. 1984

[11] S. A. Kauffman, "Emergent properties in random complex automata", ibid.

[12] R. Milner, "A Calculus of Communicating Systems", Lecture Notes Comp. Sci., Springer, Berlin, 1989

[13] R. Milner, "Calculi for synchrony and asynchrony", Theoretical Comp. Sci. 25, Elsevier, 1983

[14] R. Milner, "Communication and Concurrency", C. A. R. Hoare series ed., Prentice-Hall Int'l, 1989

[15] J. von Neumann, "Theory of Self-Reproducing Automata", edited and completed by A. W. Burks, Univ. of Illinois Press, Urbana, 1966

[16] J. C. Reynolds, "Theories of Programming Languages", Cambridge Univ. Press, 1998

[17] Ravi Sethi, "Programming Languages: Concepts & Constructs", 2nd ed., Addison-Wesley, 1996

[18] K. Sutner, "Computation theory of cellular automata", MFCS98 Satellite Workshop CA, Brno, Czech Rep., 1998

[19] P. Tosic, G. Agha, "Complete characterization of phase spaces of certain types of threshold cellular automata" (in preparation)

[20] S. Wolfram "Twenty problems in the theory of CA", Physica Scripta 9, 1985

[21] S. Wolfram (ed.), "Theory and applications of CA", World Scientifi c, Singapore, 1986

[22] S. Wolfram, "Cellular Automata and Complexity (collected papers)", Addison-Wesley, 1994