Cellular automata in noise, computing and self-organizing

Peter Gács

Boston University

Quantum Foundations workshop, August 2014

Goal: Outline some old results about reliable cellular automata.

Why relevant: There is no reference to quantum foundations or cosmology. But some thinkers about cosmology may doubt the possibility for a system of arbitrary “logical depth” to evolve at positive temperature.

The results: There is a cellular automaton that

- can perform an arbitrary computation while resisting random noise, provided its level is small (but constant). It continuously cleans away the consequences of faults, preventing their accumulation.
- can self-organize: do the above even if started from a very simple (essentially homogenous) initial condition.

Caveats

- This is not working in thermal equilibrium. But it can work in an environment like the earth’s surface.
- The CA is, though finite, still very complex (and so is the proof that it works).
- Such complex elementary units are unlikely to exist in the actual universe. But no physical law prohibits them.

History $\eta(x,t)$.

time

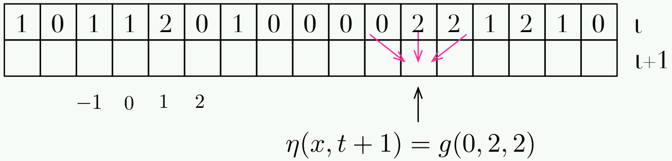
0	1	0	1	1	2	0	1	0	0	0	0	2	2	1	2	1	0
1	1	1	1	0	2	0	1	0	1	0	2	0	2	1	2	1	0
2	1	0	1	1	$\boxed{2}$	$\boxed{1}$	1	0	0	0	0	2	2	1	2	2	0
	0	0	0	1	0	0	2	1	0	2	0	1	2	0	1	1	1
			-1	0	1	2											

$$\eta(1,2) = 2, \eta(2,2) = 1, \ldots$$

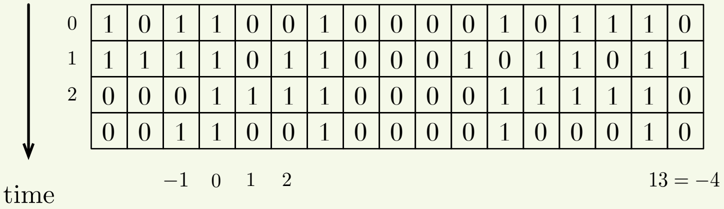
We say that history η is a trajectory of local transition function $g : \mathbb{S}^r \to \mathbb{S}$ if

$$\eta(x, t + 1) = g(\eta(\theta_1(x), t), \ldots, \eta(\theta_r(x), t)).$$

Example $\Lambda = \mathbb{Z}$, $N = \{-1, 0, 1\}$.



Here is a trajectory of Wolfram's rule 110 on $\mathbb{Z}/(17\,\mathbb{Z})$.



0	1	0	1	1	0	0	1	0	0	0	0	1	0	1	1	1	0
1	1	1	1	1	0	1	1	0	0	0	1	0	1	1	0	1	1
2	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1	0
	0	0	1	1	0	0	1	0	0	0	0	1	0	0	0	1	0
			-1	0	1	2											$13 = -4$

The rule says: "If your right neighbor is 1 and the neighborhood state is not 111 then your next state is 1, otherwise 0".

Computation

A cellular automaton **A** can be used as a computing device.

- The program P and the input X can be some strings written into the initial configuration $\xi = \xi_{(P,X)}$.
- The computation is a trajectory of **A** starting with ξ.
- The output is defined by some convention.

Perturbation

Let g be the transition of a deterministic CA $\mathbf{A}$.
A stochastic process $\eta(x,t)$ is a trajectory of an ε-perturbation of $\mathbf{A}$ if, with events

$$\mathcal{E}_{x,t} = \left\{\eta(x,t+1) \neq g(\eta(x-1,t),\eta(x,t),\eta(x+1,t))\right\},$$

for distinct space-time points $u_1,\ldots,u_k$ we have

$$\mathbb{P}(\mathcal{E}_{u_1} \wedge \mathcal{E}_{u_2} \wedge \cdots \wedge \mathcal{E}_{u_k}) \leqslant \varepsilon^k.$$

Sense of fault-tolerance

For simplicity, let us just want cell 0 to keep some initial information forever (with large probability). The simplest highly nontrivial result concerns just keeping a bit of information forever:

Theorem There is a one-dimensional deterministic cellular automaton $\mathbf{A}$ with some function F on the set of states, an ε, and initial configurations ξ_0, ξ_1 with the following property for both $b \in \{0, 1\}$. Let η be a trajectory of the ε-perturbation of $\mathbf{A}$. If $\eta(x, 0) = \xi_b(x)$ for all x then

$$\mathbb{P}\{ F(\eta(0, t)) \neq b \} \leqslant 1/3.$$

In 2 dimensions this is much easier to achieve (Toom's Rule), though not trivial.

- The 1-dimensional result contradicts some physicists' intuition that there is "no phase transition" in 1 dimension.
- Unlike Toom's rule, it does not rely on geometry, only on "pure organization". Its hierarchical structure can carry the much heavier burden of arbitrary computation as well.

Why difficult?

Suppose we start from a configuration of all 0's or all 1's, and want to remember, which one it was, in noise.

- Idea: some kind of local voting.
- In 1 dimension, seems hopeless: suppose we started from all 0's. Eventually, a large island of 1's appears.

  0000000000011111111111111111110000000000000

  A local voting-type (monotonic) rule cannot eliminate it (sufficiently fast): at a boundary, it does not know which side is the island side.

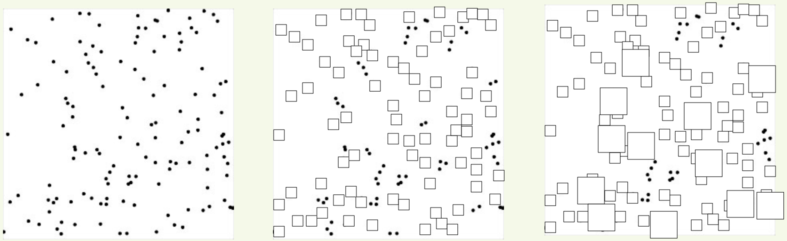
Finite versions

- If the infinite system is ergodic (eventually loses all information about its initial configuration), then the amount of time that the finite version can keep information (relaxation time) will stay bounded even if we increase the size of the space.
- For our fault-tolerant infinite automata, the relaxation time of the finite version is a growing exponential function of the size.

Noise

How do we deal with low-probability noise combinatorially? Low probability is not a combinatorial property, low frequency is.

- Consider first noise that has low frequency everywhere (noise of level 1).
- Then allow violations of this, but assume that those violations have even lower frequency (noise of level 2).
- And so on.
- After making this precise, one can prove that this classifies all faults arising in a low-probability noise.



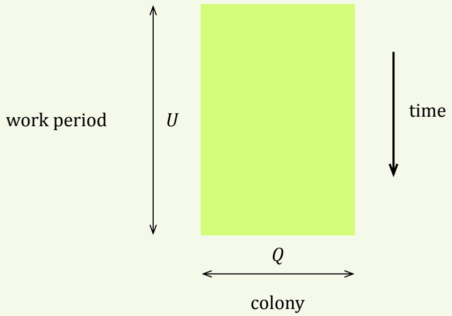
The noise is 2-sparse if there are no dots left in the last picture.

Resisting 1-sparse noise

- Suppose that individual bursts of faults are well-separated.
- To correct them, organize the cells into colonies. Each colony stores its information in redundant form, and performs it computation (interacting with neighbors) with some repetition.
- It is useful to view this structure as a “simulation”.

Block simulation

A block simulation uses a block code between two cellular automata with a special property: machine M^* is simulated step-for-step by another machine M.



- Each cell of M^* is represented by a colony of Q cells of M.
- Each step of M^* is simulated by a work period of U steps of M.

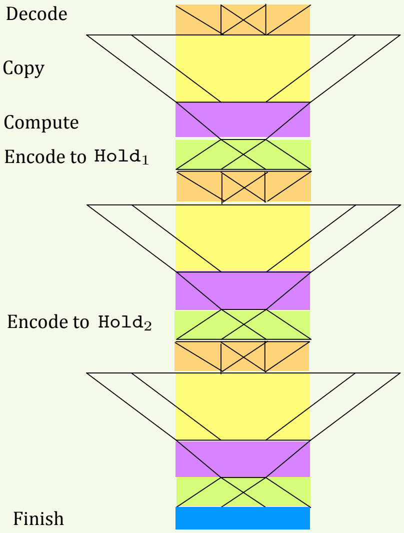
Fields

It is useful to view each state as a data record having several fields.
Example, with `Info`, `Addr`, `Age`, `Mail`, `Work` tracks:

`Info`	*ua*	*vw*	*ax*	*zf*	*yy*	
`Addr`	7	0	1	2	3	. . .
`Age`	41	41	41	41	41	
`Mail`	*b*	*a*	*r*	*z*	*x*	
`Work`	*k*	*m*	*l*	*s*	*m*	

Each cell's bits are shown as a vertical string subdivided into fields.

The simulation program seen in space-time



Decode Majority of the three repetitions.

Copy From neighbor colonies.

Compute Apply the simulated transition function g.

Encode Store 3 copies in $\texttt{Hold}_i$

Repeat the above, for $i = 1, 2, 3$

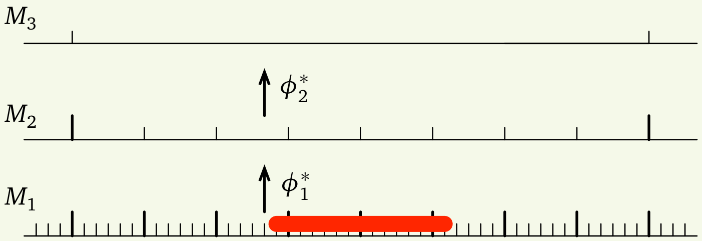
Finish $\texttt{Info} \leftarrow \mathrm{Maj}_{i=1}^{3} \texttt{Hold}_i$ locally.

Hierarchy

- Machine M_1 resists a 1-sparse set of faults: bursts of size $\beta\rho_1$ that were at a distance greater than ρ_2 from each other.
- Upgrade: now we want to resist a 2-sparse set. So, we may also have bursts of size $\beta\rho_2$, (at distances $> \rho_3$).
- Idea: Let M_2 be itself a simulation of some machine M_3, where M_2 resists a 2-sparse set! We could build M_2 from M_3 just as we built M_1 from M_2:

  $$M_1 \to M_2 \to M_3.$$

  It uses blocks of Q_2 cells of M_2, where $Q_1 Q_2 < \rho_3/3$.
- As we construct M_2 from M_3 and M_1 from M_2, the state set $\mathbb{S}_1$ should not grow.

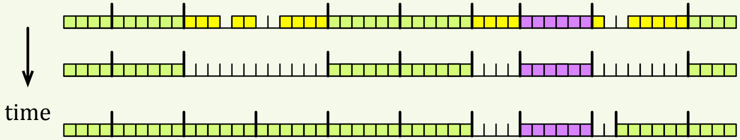


We hope that M_3 can deal with 2-sparse violations of 1-sparsity (red area above), since the cells of M_1 simulating it (via $\phi_2^*\phi_1^*$) are stretching over an area of size $\gg \rho_2$.
Indeed, the the extra redundancy in the second-level colonies deals with the information effects of the new faults, provided the faults leave the simulation on level 1 intact.

New problem: structure destruction

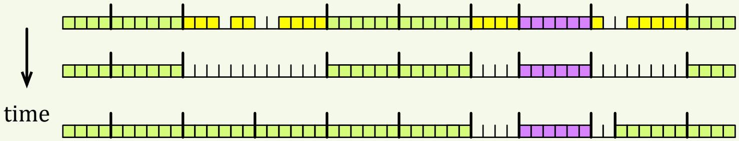
- Now faults can wipe out the structure of 3-4 consecutive colonies of M_1 (see red area again). In this case, it makes no sense to talk about M_2 simulating M_3, since those cells of M_2 are not even there (they would exist only in simulation by M_1).
- This new problem—that the M_2 cells may not exist—must still be solved in automaton M_1.

We propose two more rules.

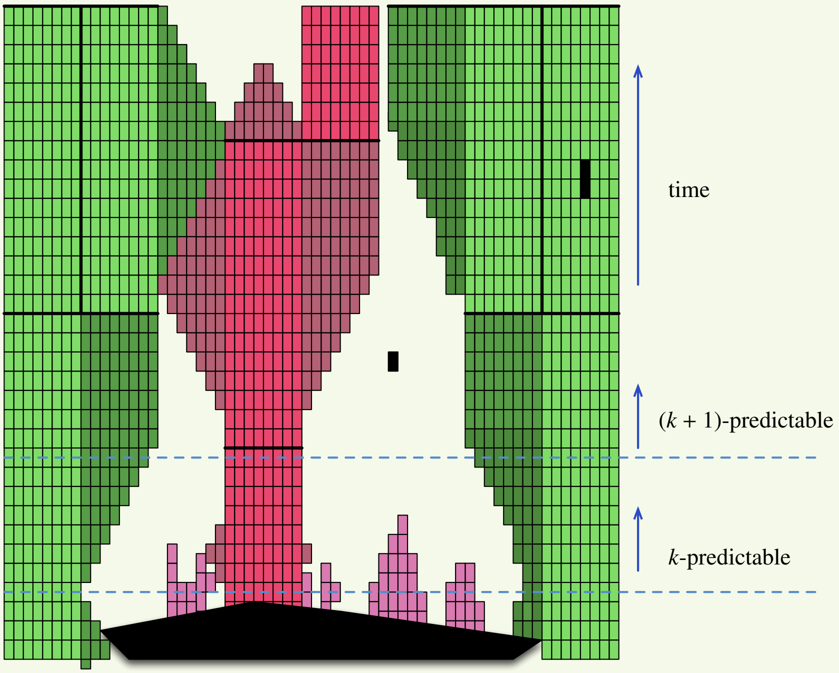


- Rule *Decay* kills a cell for which healing did not solve promptly its inconsistency with a neighbor within its own colony. Repeated application of this will wipe out unhealable partial colonies (yellow cells).
- Rule *Grow* lets a colony extend an arm of consistent cells into nearby vacuum. If new colony creation fails within a certain number of steps, the arm is erased.

New problem: faults can create whole bad colonies (for example, the purple colony above is misplaced). How to get rid of these?



Key idea: the bad colony should eliminate itself.
To reason about this, generalize the notion of history for cellular automata—in order that a misplaced colony of M_1 could also be viewed as simulating a (misplaced) cell of M_2.


time
$(k+1)$-predictable
k-predictable

Forced simulation

Automaton M_1 needs the following property:

Forced simulation As long as the local structure (the `Addr`, `Age` variables) is in order, a colony always carries out the program of simulating a cell of M_2.

A typical cellular automaton $\mathbf{A}_1$ simulating some other cellular automaton $\mathbf{A}_2$ would rely on some program of $\mathbf{A}_2$, written into each colony of $\mathbf{A}_1$. The simulation performed by machine M_1 must be, on the other hand, hard-wired: it should not rely on any written program, since that program could be corrupted.

Amplifiers

The above ideas allow to define sequence of generalized cellular automata and simulations:

$$M_1 \xrightarrow{\Phi_1} M_2 \xrightarrow{\Phi_2} M_3 \xrightarrow{\Phi_3} \cdots ,$$

called an amplifier.
Only M_1 is there physically!
The claim of the theorem follows easily from the basic properties of the amplifier.

- The details are tedious. . .

Computation

- The first theorem only claimed that we can build a CA remembering one bit. But it is easy to add a computing layer (field) to each cellular automaton M_k: this gives computations that are more reliable on higher levels.
- Even better: start the computation on the bottom level and lift it to higher levels only as it runs longer.

Self-organization

- The above construction assumes an initial configuration that already represents an infinite hierarchy of simulations. But it is also possible to start from a homogenous initial configuration.
- How do we break the symmetry? Using randomness. This creates some seeds from which colonies can grow, and from them supercolonies, and so on.
- How do we deal with growths that started from different seeds and collide?
  - If one of them is bigger, it overrides the other.
  - If they are equal, they toss a coin to decide who overrides.