Reliably computing cellular automata

Peter Gács

Boston University

Goal: Outline some old, but still not generally digested results about reliable cellular automata.

The results: There is a cellular automaton that can perform arbitrary computation while resisting a the most natural kind of random noise, provided its volume is small.
Such an automaton can continuously clean away the consequences of faults, preventing their accumulation.

Emphasis: On hierarchical methods (of construction and proof), since this is where mainly my own contribution lies.

Idiosynchrasies: Though the results are simply stated, the methods lead into a somewhat special world with its own concepts—please, be patient.

I assume you know cellular automata, but I need some special terminology.

- Set of cells (sites): for example, $\Lambda = \mathbb{Z}^3$, or $\Lambda = \mathbb{Z}/m\mathbb{Z}$.
- Finite set $\mathbb{S}$ of (local) states.
- Configuration: any function $\xi : \Lambda \to \mathbb{S}$.

$$\Lambda = \mathbb{Z} \qquad\qquad \mathbb{S} = \{0, 1, 2\}$$

1	0	1	1	2	0	1	0	0	0	0	2	2	1	2	1	0
		-1	0	1	2											

$\xi(-1) = 1, \xi(0) = 1, \xi(1) = 2, \ldots$

History $\eta(x, t)$.

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											

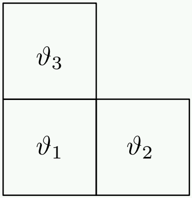
time

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

Neighborhood function: $N(x) = \{\theta_1(x), \ldots, \theta_r(x)\}$.
Normally $\Lambda = \mathbb{Z}^d$ and we have $\theta_i(x) = x + \theta_i(\mathbf{0})$.

Examples

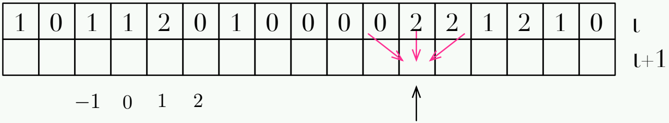
- von Neumann neighborhood: the 7 nearest neighbors (including itself) of a point, say, in the lattice $\mathbb{Z}^3$.
- Toom neighborhood: $\{\theta_1(\mathbf{0}), \theta_2(\mathbf{0}), \theta_3(\mathbf{0})\} = \{(0,0), (0,1), (1,0)\}$.



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



$$\eta(x, t + 1) = g(0, 2, 2)$$

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											$13=-4$

time

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

So, a (deterministic, synchronous) cellular automaton is given by these data:

$$\mathbf{A} = \mathrm{CA}(\Lambda, \mathbb{S}, \theta, g).$$

We will omit Λ, θ if $\Lambda = \mathbb{Z}$, and $N(\mathbf{0}) = \{-1, 0, 1\}$.

Example (The Toom Rule) $\Lambda = \mathbb{Z}^2$, $\mathbb{S} = \{0, 1\}$,

$$\begin{aligned} N(\mathbf{0}) &= \{(0,0), (0,1), (1,0)\}, \\ g(x,y,z) &= \mathrm{Maj}(x,y,z). \end{aligned}$$

The new state is the majority of the state of the cell itself, and of its northern and eastern neighbor.

A cellular automaton $\mathbf{A}$ 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 $\mathbf{A}$ starting with ξ.
- The output is defined by some convention.

Assume these conventions fixed somehow, this defines a (possibly partial) function $f_{\mathbf{A},P}(X)$ computed on cellular automaton $\mathbf{A}$ with program P.

A cellular automaton $\mathbf{A}$ is computationally universal if for every computable function $g(X)$ there is a program P with $f_{\mathbf{A},P}(X) = g(X)$.

Theorem There are computationally universal cellular automata.

For example, it is easy to turn any universal Turing machine into a one-dimensional computationally universal cellular automaton.

Cellular automata are particularly well-suited as models for computation in noise.

- Their space-time uniformity means we do not assume any complex hardware structure immune to errors (unlike von Neumann's fault tolerant circuits).
- Their parallelism provides power to combat noise that occurs all over space-time.

For simplicity of notation, assume 1 dimension, with $x-1, x, x+1$ the neighbors of site x.
Let E be a set of space-time points, and η a history of the cellular automaton $\mathbf{A}$ with transition function g. The pair (η, E) is called a perturbed trajectory of $\mathbf{A}$ with set of faults, or exceptions, or noise E if

$$\eta(x, t+1) = g(\eta(x-1, t), \eta(x, t), \eta(x+1, t))$$

for all $(x, t) \notin E$. The set of all possible noises is denoted by

$$\mathrm{Noises} = 2^{\Lambda \times \mathbb{Z}_+}.$$

The non-stochastic approach to error correction is interested in cellular automata whose perturbed trajectories behave well provided the set of faults obeys some reasonable restrictions.

The stochastic point of view does not impose restrictions directly on the set of faults, instead assumes that they come from some stochastic process, and the restrictions apply to the distribution of the process. Instead of a local transition function $g : \mathbb{S}^3 \to \mathbb{S}$, now a local transition probability matrix $W : \mathbb{S}^4 \to [0, 1]$ with

$$\sum_{s \in \mathbb{S}} W(s, r_{-1}, r_0, r_1) = 1.$$

A PCA is noisy if all of its local transition probabilities in matrix W are positive (no prohibited local transitions).

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

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

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

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

This is in some ways more restricted than a PCA (must be close to a deterministic automaton), and in some ways more general (complete homogeneity is not required).
For a while we focus on stochastic perturbation, but our solutions will relate the deterministic and the stochastic notions of perturbation to each other—through the notion of sparsity.

For simplicity, let us just want cell 0 to keep some initial information forever (with large probability). The simplest highly nontrivial result to be explained:

Theorem (Main) There is a one-dimensional deterministic cellular automaton with some partition of the set of states

$$\mathbb{S} = D_0 \cup D_1, \quad D_0 \cap D_1 = \emptyset$$

and initial configurations ξ_0, ξ_1 with the following property for both $b \in \{0, 1\}$. If $\eta(x, 0) = \xi_b(x) \in D_b$ for all x then

$$\mathbb{P}\{\, \eta(0, t) \notin D_b \,\} \leqslant 1/3.$$

The proof uses significantly some initial ideas of Kurdyumov.

Let us explore the significance of the main theorem for the theory of probabilistic cellular automata. Let $\eta(x, t)$ be a trajectory of a probabilistic cellular automaton $\mathbf{A}$. Let

$$\mu_t$$

be the probability distribution of the random history $\eta(\cdot, t)$. The time transition is described by a linear operator P:

$$\mu_{t+1} = P\mu_t.$$

A measure μ is invariant (an equilibrium measure) if $\mu = P\mu$. It is easy to show that invariant measures always exist.

A PCA with transition operator P is ergodic if

a. There is only one invariant measure ν.
b. For every initial configuration, the measures μ_t converge weakly to ν.

Weak convergence: convergence on all sets of the form

$$\{\, \zeta : \zeta(-n) = s_{-n}, \ldots, \zeta(n) = s_n \,\}.$$

Ergodicity says that, eventually, the automaton forgets everything about the initial configuration.

- A noisy cellular automaton on a finite space is always ergodic, as a finite Markov chain with all positive transition probabilities. So in what follows, we assume the space infinite (and return to the finite case later).
- It is easy to construct examples of ergodic cellular automata: just let the transition matrix $W(s, r_{-1}, r_0, r_1)$ be independent of r_{-1}, r_0, r_1.
- It is also easy to construct examples of non-ergodic ones: just take a deterministic automaton that never changes its state!
- The Main Theorem above gives a noisy non-ergodic infinite cellular automaton.

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

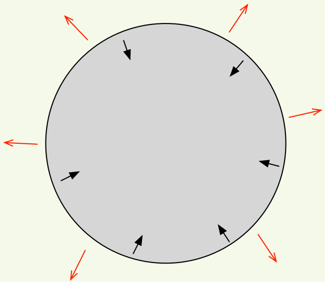
- Idea: some kind of local voting.
- In 1 dimension, seems hopeless: suppose we started from all 0's. Eventually, a large island of 1's appears.

  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. (Theorems of Gray.)

Symmetric voting in 2 dimensions?

- Voting in the 5-element symmetric neighborhood? In the absence of noise, will not decrease a large rectangle of 1's in a sea of 0's.
- Even in noise, any symmetric local voting (including the center) will decrease a large disk of radius r of 1's only with average speed $1/r$. If the noise is biased (say brings 1 with probability ε and 0 with probability 0), it increases the disk with constant average speed ε.

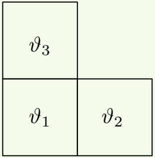


Result: increase with speed $\varepsilon - 1/r > 0$! Even if we started with all 0's, the 1's win out.

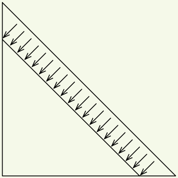
Assume that the result of local voting in the symmetric (von Neumann) neighborhood is changed to 1 with the same probability ε as to 0.

- The conjecture is that the system is nonergodic, but there is no proof.
- In continuous time, proved nonergodic for a special choice of transition rates: the ones making it the dynamic version of the Ising model of statistical physics.

Toom's voting rule is not symmetric: it uses the neighborhood (self, north, east).



In a sea of 0's it erases an arbitrary island of size L in L steps.



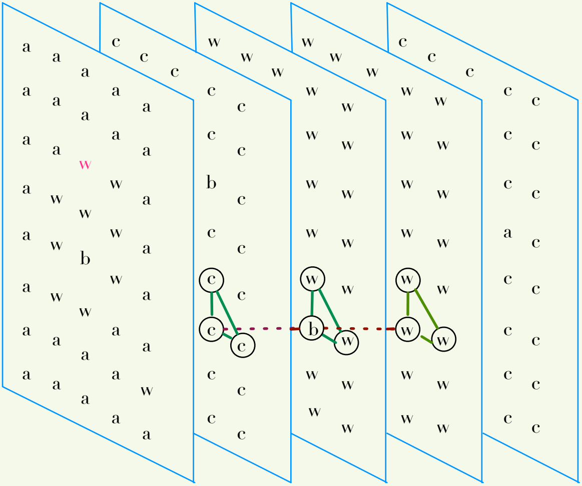
Any PCA obtained by ε-perturbation with small ε from this rule is nonergodic. The proof is not easy, we will give sketch.

The simplest known fault-tolerant computation model is the three-dimensional cellular automaton introduced in [Gács-Reif 88].

Definition (Toom-layering) Let $\mathbf{U}$ be an arbitrary 1-dimensional cellular automaton. We define its Toom-layering as a 3-dimensional automaton

$$\mathbf{U}'.$$

In its initial configuration, we slice the space into planes by the value of the first coordinate. Every cell with coordinates x, y, z will have the initial state of cell x of automaton $\mathbf{U}$.
The transition rule of $\mathbf{U}'$ is: Toom's rule within each plane, then the rule of $\mathbf{U}$ across the planes.



Transition rule of **U′**: Toom's rule within each plane, then the rule of **U** across the planes.

In what sense is this fault-tolerant?

Theorem (Reliable computation, infinite version) There is a noise bound ε with the following property. Let $\zeta(x, t)$ be a computation (history) of $\mathbf{U}$, and let $\eta(x, y, z, t)$ be a trajectory of an ε-perturbation of the Toom-layering $\mathbf{U}'$ with initial condition $\eta(x, y, z, 0) = \zeta(x, 0)$. Then for all $x, y, z \in \mathbb{Z}$, $t \in \mathbb{Z}_+$ we have

$$\mathbb{P}\{\, \eta(x, y, z, t) \neq \zeta(x, t) \,\} = O(\varepsilon).$$

As noted, a noisy finite PCA, say a noisy perturbed Toom rule on the space

$$\Lambda = \mathbb{Z}_L^2,$$

(a torus of size L) is always ergodic. What is the significance of the above results then?
For two initial configurations ξ_0, ξ_1 let $\eta_i(\boldsymbol{x}, t)$ be the process starting from ξ_i. We call (for simplicity) the relaxation time $R_\delta(L, \varepsilon)$ the smallest time t_0 such that for all $t \geqslant t_0$:

$$|\,\mathbb{P}\{\,\eta_1(\mathbf{0}, t) = 0\,\} - \mathbb{P}\{\,\eta_0(\mathbf{0}, t) = 0\,\}| < \delta.$$

Estimates the time for the information about index i to be erased from the value $\mathbb{P}\{\,\eta_i(\mathbf{0}, t) = 0\,\}$.

Proposition If the infinite system on $\mathbb{Z}^2$ is ergodic then the relaxation time $R_\delta(L, \varepsilon)$ has an upper bound $M(\delta, \varepsilon)$ independent of the size L.

Thus, ergodicity of the infinite version implies that increasing the size of the finite version will not increase its fault-tolerance significantly. On the other hand, there is a proof (by Berman-Simon) of Toom's theorem showing that the relaxation time of a perturbed Toom rule grows exponentially with the size L:

Theorem For a perturbed version of Toom's rule, and some $\delta = O(\varepsilon)$, $c(\delta, \varepsilon) > 0$ we have

$$R_\delta(L, \varepsilon) > 2^{cL}.$$

- In the finite version of the reliable computation theorem, the upper bound $O(\varepsilon)$ is replaced with

$$O(\varepsilon) + t \cdot 2^{-cL}$$

for some $c > 0$. This shows that we can compute exponentially long in a cellular automaton of size L.

- A user may want to know how to implement a computation with a given space need S and time need T, where the whole result is correctly decodable with probability, say, $1 - \delta$. In this case, our computing device should be a 3-dimensional torus

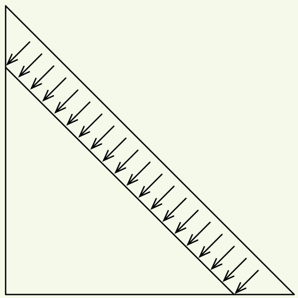
$$\mathbb{Z}_S \times \mathbb{Z}_L^2, \quad L = O(\log(ST/\delta)).$$

Along the computing dimension of length S we perform the computation, along the stabilizing dimensions of logarithmic length L, the Toom rule. The result in each position along the computing dimension is obtained at time T by majority vote along the stabilizing dimensions.

The proof of the reliability of the above computing automaton is almost the same as the proof of nonergodicity of Toom's rule (essentially, just carry an extra dimension in the notation), so we concentrate on Toom's Rule. The proof I choose is not the simplest, not even the strongest one (in terms of the relaxation time lower bound). But

- it is based on the simple intuition of the shrinking triangles,
- its hierarchical technique will be reused in the later lectures.

Recall the intuitive explanation for why Toom's rule works:



The noiseless rule shrinks a triangle surrounded by 0's. However,

- Our rule is now noisy.
- The outside now contains "litter", too.

Still, a simulation of the noisy Toom rule strongly supports the shrinking-triangle intuition.

We return to the problem of relating deterministic and stochastic perturbations to each other.
How to 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.

Define the distance of two points (in, say 3 dim):

$$|\boldsymbol{x} - \boldsymbol{y}| = \max(|y_1 - x_1|, |y_2 - x_2|, |y_3 - x_3|).$$

Ball (actually, a cube):

$$B(\boldsymbol{x}, r) = \{\, y : |\boldsymbol{x} - \boldsymbol{y}| < r \,\}.$$

Let E be a set of space-time points. A point of E is (r, R)-isolated if

$$d(x, E \setminus B(x, r)) > R,$$

that is each point of E is either closer than r to x or farther than R. Let

$$D(E, r, R)$$

denote the set of points of E that are not (r, R)-isolated.

We will permanently fix a sequence $1 = \rho_1 < \rho_2 < \cdots$, and some $\beta \geqslant 8$.

- Let $E^{(1)} = E$. The sets of $E^{(2)}, E^{(3)}, \ldots$ are obtained by deleting first the $(\beta\rho_1, \rho_2)$-isolated points, then the $(\beta\rho_2, \rho_3)$-isolated points, and so on:

$$E^{(k+1)} = D(E^{(k)}, \beta\rho_k, \rho_{k+1}).$$

  We call $E^{(k)}$ the k-noise of E.
- Set E is (r, R)-sparse if $D(E, r, R) = \emptyset$: it consists of "bursts" of size r farther than R.
  It is k-sparse if $E^{(k+1)} = \emptyset$.
  It is sparse if $\bigcap_k E^{(k)} = \emptyset$.

The following observation is key.

Proposition (Sparsity Bound) Assume $\log\beta \leqslant \log\frac{\rho_{k+1}}{\rho_k} \ll 1.5^k$. Then for small enough ε, the following holds for all $k \geqslant 1$. Assume that each point is in the random space-time set $\mathcal{E}$ independently from all the others, with probability $\leqslant \varepsilon$. Then for each point x and each k,

$$\mathbb{P}\{\, B(x,\rho_k) \cap \mathcal{E}^{(k)} \neq \emptyset \,\} < 2\varepsilon \cdot 2^{-1.5^k}.$$

(Case $k = 1$, and $B(x,\rho_1)$ (a single point): this gives $< \varepsilon$ as expected.)
The probability that the noise in $B(x,\rho_k)$ is not k-sparse is decreasing doubly exponentially with k.
This suggests hiearchical (multiscale) proof: on "level" k, deal just with $(\beta\rho_k, \rho_{k+1})$-isolated faults.

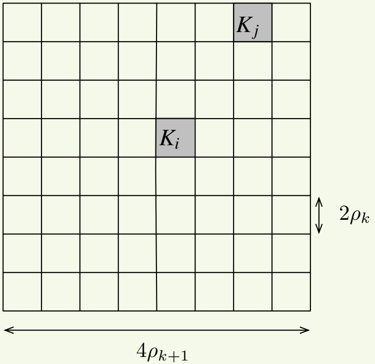
One can see that the event $B(x, \rho_k) \cap \mathcal{E}^{(k)} \neq \emptyset$ depends at most on $\mathcal{E} \cap B(x, 3\rho_k)$.
By induction, proving

$$\mathbb{P}\{\, B(x, \rho_{k+1}) \cap \mathcal{E}^{(k+1)} \neq \emptyset \,\} < 2\varepsilon \cdot 2^{-1.5^{k+1}}.$$

Let $p_k = 2\varepsilon \cdot 2^{-1.5^k}$. Suppose $y \in \mathcal{E}^{(k+1)} \cap B(x, \rho_{k+1})$. Then there is a point

$$z \in B(y, \rho_{k+1}) \cap \mathcal{E}^{(k)} \setminus B(y, \beta\rho_k).$$

Consider a standard partition of the (three-dimensional) space-time into balls (cubes) $K_p = c_p + [-\rho_k, \rho_k)^3$ with centers $c_1, c_2, \dots$. The balls K_i, K_j containing y and z respectively intersect $B(x, 2\rho_{k+1})$. The triple-size balls $K_i' = c_i + [-3\rho_k, 3\rho_k)$ and K_j' are disjoint, since $d(y, z) > \beta\rho_k$ by assumption.



$\mathcal{E}^{(k)}$ must intersect two balls (cubes) K_i, K_j of size $2\rho_k$ separated by at least $4\rho_k$, of $B(x, 2\rho_{k+1})$.

By inductive assumption, the event $\mathcal{F}_i$ that K_i intersects $\mathcal{E}_k$ has probability bound p_k. It is independent of the event $\mathcal{F}_j$, since these events depend only on the triple size disjoint balls K'_i and K'_j. The probability that both of these events hold is at most p_k^2. The number of possible cubes K_p intersecting $B(x, 2\rho_{k+1})$ is at most $C_k := ((2\rho_{k+1}/\rho_k) + 2)^3$, so the number of possible pairs is at most $C_k^2/2$, bounding the probability of our event by

$$\begin{aligned} C_k^2 p_k^2/2 &= 2C_k^2 \varepsilon^2 2^{-1.5^{k+1}} \cdot 2^{-0.5\cdot 1.5^k} \\ &= 2\varepsilon 2^{-1.5^{k+1}} \cdot \varepsilon C_k^2 2^{-0.5\cdot 1.5^k}. \end{aligned}$$

Our assumptions imply that the last factor is $\leqslant 1$ when ε is small.

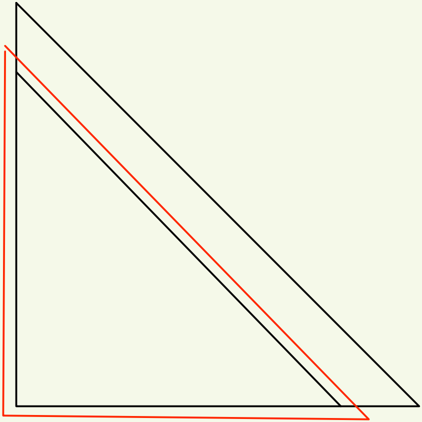
Noise is a concept in space-time, damage is the corresponding concept in space $\Lambda = \mathbb{Z}^2$. We can talk about the k-damage $D^{(k)}$ of a set $D \subseteq \Lambda$, using the same sequence ρ_k, but possibly a larger parameter β.
In studying the Toom rule starting from all 0's, let
$D(t) = \{ x : \eta(x, t) = 1 \}$.

A triangle of size $s = c - a - b > 0$ is a set of the plane given as follows:

$$T(a, b, c) = \{ (x, y) : x \geqslant a, y \geqslant b, x + y \leqslant c \}.$$

when we said the triangle shrinks, we meant that it is replaced with $T(a, b, c - 1)$. It disappears when c becomes smaller than $a + b$.

In noise, triangles do not shrink quite as before. But, as will be shown, they still shrink.



Let $v_k = \sum_{i=1}^{k-1} \frac{c_0}{i^2}$ where $\sum_{k>0} \frac{c_0}{k^2} < 1/2$. Let $z \in \Lambda$ an arbitrary point, $B(d) = B(z, d)$, $L > \Delta > 0$.

Conditions:

- a $\frac{\rho_{k+1}}{\rho_k} \gg k^2$.
- b $D^{(k+1)}(t_0 - \Delta) \cap B(L) \subseteq T(a, b, c)$.
- c The pair (η, E) is a perturbed trajectory of the Toom rule with a set of faults E such that $E^{(k+1)}$ does not intersect $B(L) \times [t_0 - \Delta, t_0]$.

Proposition Under these conditions, if $\Delta > \rho_{k+1}$ then

$$B(L - \Delta) \cap D^{(k+1)}(t_0) \subseteq T(a - v_k\Delta, b - v_k\Delta, c - (1 - v_k)\Delta).$$

Without noise this would be $T(a, b, c - \Delta)$. Size shrinks by $\Delta(1 - 3v_k)$ instead of by Δ.

Corollary Suppose, with $\Delta = 4\rho_{k+1}$:

- (a) $D^{(k+1)}(t_0 - \Delta)$ does not intersect $B(L)$.
- (b) We have a perturbed trajectory (η, E) where $E^{(k)}$ does not intersect $B(L) \times [t_0 - \Delta, t_0]$.

Then $B(L - \Delta) \cap D^{(k)}(t_0) = \emptyset$.

Indeed, $D^{(k)}(t_0 - \Delta) \cap B(L)$ is enclosed in balls of the form $B(x, \beta\rho_k)$, separated by distances of ρ_{k+1}. The balls are contained in triangles of size $2\beta\rho_k$. All these will be "erased" in time $4\beta\rho_{k+1}$, according to the Proposition.

Suppose that a trajectory of an ε-perturbation of the Toom rule started from all 0's. Let (x, y, t_0) be a space-time point, $t_k = t_0 - 4\beta(\rho_1 + \cdots + \rho_k)$, and consider the ball $B_k = B((x, y), 2\rho_k)$.

- Let $\mathcal{G}_k$ be the event that $D^{(k+1)}$ does not intersect B_k at time t_k. True for a sufficiently large k, since there were no 1's at time 0.
- Let $\mathcal{F}_k$ be the event that $E^{(k)}$ does not intersect $[t_{k+1}, t_k] \times B_{k+1}$. The Sparsity Bound gives a constant C_1 with $\mathbb{P}(\bigcap_k \mathcal{F}_k) > 1 - C_1\varepsilon$.

By the Corollary, $\mathcal{G}_{k+1} \wedge \mathcal{F}_k \Rightarrow \mathcal{G}_k$. Assuming that $\mathcal{F}_k$ holds for all k:

$$\mathcal{G}_k \overset{\mathcal{F}_{k-1}}{\Longrightarrow} \mathcal{G}_{k-1} \overset{\mathcal{F}_{k-2}}{\Longrightarrow} \cdots \overset{\mathcal{F}_0}{\Longrightarrow} \mathcal{G}_0,$$

hence $\eta(x, y, t) = 0$.

Proof idea

The Proposition is proved by induction.

- Inductive assumption gives shrinking with velocity $1 - 3v_{k-1}$ when the k-noise is also missing, instead of just the $(k+1)$-noise.
- The k-noise brings in some blocks of size ρ_k, separated from each other by ρ_{k+1}.
  Thus, the "relative frequency" of violations of $(k-1)$-sparsity is about $\frac{\rho_{k+1}}{\rho_k} = \frac{1}{Ck^2}$, from which $v_k - v_{k-1}$ is obtained.

7 Summary of previous lectures

1. (Larry) Probabilistic cellular automata. Formulating the main result in context, and some ideas (fields).
2. (Peter) The sparsity method. Its application to proving the nonergodicity of the perturbed two-dimensional Toom rule.
3. (Larry) Reliable simulation in 1 dimension in 1-sparse noise: colonies, the most important fields.
4. (Peter, today) Elaborating the simulation component of Larry's last lecture. New problems caused by non-1 sparse faults. Forced self-simulation.

The simplicity of the 2(3)-dimensional solution seems to be an anomaly. Desirable:

Fewer dimensions There are thermodynamic reasons to argue that a 3-dimensional fault-tolerant cellular automaton is not realizable physically in 3-dimensional space. Indeed, in physical systems the error-correcting operations (as any irreversible operations) generate heat, which needs an extra dimension to conduct (or, as in the case of the Earth's surface, radiate) out.

Continuous time The Toom-layering construction relies on discrete time in an essential way, but synchronizing over unlimited distances is physically unrealistic.

Less redundancy Repetition is not a very economical way to introduce redundancy.

From now on, we will work with 1-dimensional cellular automata.

Let us show how to correct an (r, R)-sparse set of faults: that is faults that come in small bursts (size r, separated from each other by distance R).

We will only need that R is some large constant times greater than r.

We must store information, in order not to lose it to noise, with redundancy: using extra space.
Let X be a set whose elements are the possible values of our information, and Y some other set. The pair of mappings

$$\phi_* : X \to Y, \quad \phi^* : Y \to X$$

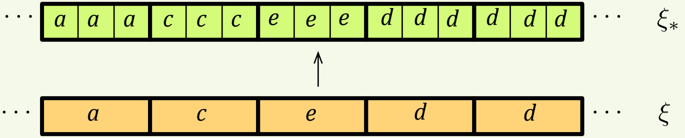
is called a code if it satisfies the identity $\phi^*(\phi_*(x)) = x$. The encoding function is ϕ_*, the decoding function is ϕ^*. Example:

$$\phi_*(x) = (x, x, x), \quad \phi^*(x, y, z) = \mathrm{Maj}(x, y, z).$$

Error correction can be seen as the process of decoding and then encoding again.

Block codes

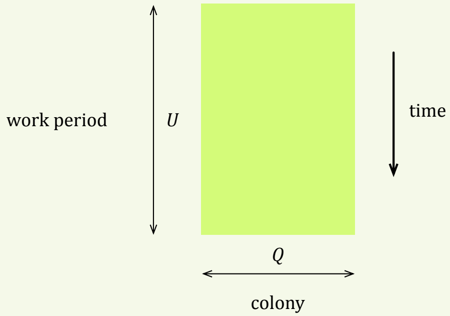
- Code (ϕ_*, ϕ^*) is called a block code with block size Q if the values of ϕ_* are words of length Q: that is $X = \mathbb{A}$, $Y \subseteq \mathbb{A}_*^Q$ for some finite alphabets $\mathbb{A}, \mathbb{A}_*$. The above example (repetition and majority decoding) is a block code.
- A block code can be extended to configurations $\xi(x)$ over the infinite space $\Lambda = \mathbb{Z}$. Example:



Decoding can also be extended (though this extension is not automatically shift-invariant).

Block simulation

A block simulation uses a block code between two cellular automata with a special property: machine $M = \mathrm{CA}(\mathbb{S}, g)$ is simulated step-for-step by another machine $M_* = \mathrm{CA}(\mathbb{S}_*, g_*)$.



- 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_*.

See precise definition of simulation later.

Let

- $\zeta_t = \zeta(\cdot, t)$ = the content of the original computation at time t.
- η_{tU} = the representation of ζ_t (with possible errors) by η at time tU.

How to perform the step $\eta_{tU} \to \eta_{(t+1)U}$?
In the Toom-layering construction, $U = 1$, and we could just work directly on η_t, step-for-step, since

- the code was very simple (repetition)
- the extra dimension allowed direct access to each bit of the code.

But in general, it is not clear how to work directly on the encoded information.

Pedestrian way

$$\eta_{tU} \to \text{ decode } \to \zeta_t \to \text{ compute } \to \zeta_{t+1} \to \text{ encode } \to \eta_{(t+1)U}.$$

During this whole process, (even if the “compute” part is trivial) the information seems vulnerable to error.

To control damage, let us structure information functionally even within individual cells. At any one time, work only on part of the information, protecting thereby the rest from error propagation.

- View the cell states as binary strings: $\mathbb{S} = \{0,1\}^m$, where m is called the capacity of the cell.
- Let $1 \leqslant f_1 < f_2 < \cdots < f_k \leqslant m$, $F = \{f_1, \dots, f_k\}$. For an arbitrary cell state $s = (s_1, \dots, s_m)$ we write

$$s.F = (s_{f_1}, \dots, s_{f_k}).$$

  We call F a field, and $s.F$ a field of s. Notation borrowed from the programming languages Pascal, C, and so on.

- Typically, different fields are disjoint intervals of bits. Viewing each cell as a computer processor, view fields as its program's variables in local memory.
- If $(\xi(x) : x \in \Lambda)$ is a configuration, then for each field F the values $\xi(x).F$ form a track

$$(\xi(x).F : x \in \Lambda).$$

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*	

Here each cell's bits are assumed to form a vertical string.

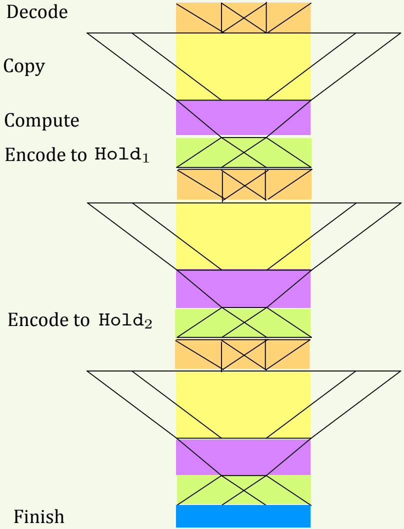
As an aside, let us give an equivalent formulation of the Main Theorem (now we assume probabilistic faults, not 1-sparse ones) in terms of fields:

Theorem There is a one-dimensional deterministic cellular automaton with some field `Rider` and initial configurations ξ_0, ξ_1 with the following property for both $b \in \{0, 1\}$. If $\eta(x, 0) = \xi_b(x)$ then

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

Program outline

- Keep the encoded state of the simulated fault-free computation in a track called `Info`.
- While decoding, computing, encoding, don't change `Info`: use other tracks: say `Mail` for moving information around, `Work` for auxiliary computations.
- Perform the decode-compute-encode process 3 times. In iteration $i = 1, 2, 3$, store the result in track $\mathtt{Hold}_i$.
- Replace `Info` with $\mathrm{Maj}(\mathtt{Hold}_1, \mathtt{Hold}_2, \mathtt{Hold}_3)$ in a single, last step, in each cell simultaneously.
- To organize all this, use a field `Age` as a program counter, and a field `Addr` to show the relative place of each cell within its group. Then each cell, as long as its `Addr` and `Age` are correct, will always know its task in the current program step.



Decode Majority of the three repetitions.

Copy From neighbor colonies.

Compute Apply the simulated transition function g.

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

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

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

Put $2r$ steps of idling (doing nothing) between all these stages.

- If the value of the fields

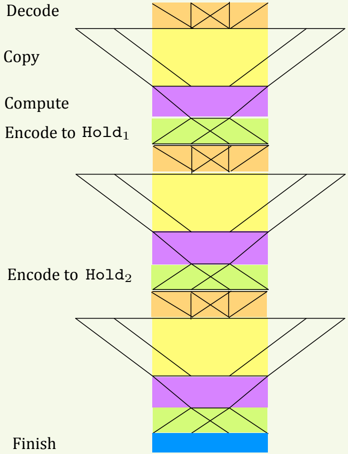
$$\texttt{Addr} \in \{0, \ldots, Q-1\}, \quad \texttt{Age} \in \{0, \ldots, U-1\}.$$

  is not corrupted by faults then each cell knows its position within its colony and the step of the program it needs to execute: so it will know what to do with the information in the rest of the fields.
- The transition table is convenient to describe by a series of rules. In the example below, $\texttt{Mail}^{-1}$ denotes the $\texttt{Mail}$ field of the left neighbor.

Rule 1: Example rule

if $t_1 \leqslant \texttt{Age} < t_2$ **and** $\texttt{Addr} < Q/2$ **then**
    $\texttt{Mail} \leftarrow \texttt{Mail}^{-1}$

All our rules are just such conditional assignments.



Assume $R > U, 3Q$, so that at most one burst of faults (of size r) can affect one colony work period.

Lemma (Nice Faults) The theorem holds in the case where the faults cannot corrupt the `Addr` and `Age` fields.

Indeed, due to the idling steps, each fault affects at most one stage i (or the step Finish). Two effects that matter:

- The output of the stage in track $\mathtt{Hold}_i$. Corrected in Finish.
- Other fields of the cells where the burst occurred. Corrected in Decode of the next work period.

Fortunately, this can be done locally. I give an example solution, based on simple common-sense ideas.

- Define a notion of live and dead cells (say by a field $\mathtt{Live} \in \{0, 1\}$).
- Live cells x, y are consistent if

  $$\mathtt{Age}(y) = \mathtt{Age}(x), \quad \mathtt{Addr}(y) \equiv \mathtt{Addr}(x) + (y - x) \pmod{Q}.$$

  A domain is a maximal interval of consistent cells.
- (*Purge* rule): Kill cells whose domain is smaller than the burst size r.
- (*Heal* rule): If a dead cell has one live neighbor with sufficient "backing", or two such consistent live neighbors, revive it consistently with them.

We will implement this.

Call the left depth $\mathrm{depth}_{-1}(x)$ of a cell x the distance to the first cell on the left that is inconsistent with it if this distance is $\leqslant \mathrm{MaxDepth} = r$; otherwise, MaxDepth. Right depth is defined accordingly. So, $\mathrm{depth}_{-1}(x) > 1$ means x is consistent with its left neighbor.
The fields $\mathtt{Depth}_{-1}$, $\mathtt{Depth}_1$ try to keep track of the depth.
pfor is parallel **for**, doing all in a single step:

Rule 2: *Purge*

pfor $j \in \{-1, 1\}$ **do**
  if $\mathtt{Live} = 1$ **and** $\mathrm{depth}_j = 1$ **then**
    if $\mathrm{depth}_{-j} < \mathrm{MaxDepth}$ **then** $\mathtt{Live} \leftarrow 0$
    else $\mathtt{Depth}_j \leftarrow 1$
  else
    $\mathtt{Depth}_j \leftarrow \mathrm{MaxDepth} \wedge (\mathtt{Depth}_j^j + 1)$

Reviving the dead cells:

Rule 3: *Heal*

pfor $j \in \{-1, 1\}$ **do**

if $\mathtt{Live} = 0$ **and** $\mathtt{Live}^j = 1$ **and** $\mathtt{Depth}^j_j = \text{MaxDepth}$ **and** ($\mathtt{Live}^{-j} = 0$ **or** the two neighbors are consistent with each other) **then**

$\mathtt{Live} \leftarrow 1$

$\mathtt{Age} \leftarrow \mathtt{Age}^j$

$\mathtt{Addr} \leftarrow \mathtt{Addr}^j - j \bmod Q$

Assume that the rules *Purge*, *Heal* have been added.

Lemma (Structure Restoration) After every burst, the affected area becomes consistent with its neighborhood again in $3r$ steps. The `Info` field is not affected anywhere outside the burst.

- The fact that the `Info` field is not affected follows directly from the design.
- The proof of restoration of consistency takes some argument, since the behavior is not completely monotonic: *Heal* may revive cells that *Purge* will kill later. But it is not a difficult argument.

The Structure Restoration lemma allows the application of the argument of the Nice Faults lemma, even if the faults are not nice, just 1-sparse. This finishes the proof of the theorem.

Proposition (Sparse error correction) Let $M = \mathrm{CA}(\mathbb{S}, g)$ be an arbitrary cellular automaton. For any r there is a new machine $M_* = \mathrm{CA}(\mathbb{S}_*, g_*)$ and:

- $|\mathbb{S}_*|$ depending somehow (see discussion) on $r, |\mathbb{S}|$,
- R, Q, U depending linearly on r, and somehow on $|\mathbb{S}|$,
- Block code (ϕ_*, ϕ^*) with block size Q,

such that the following holds. Let
$\zeta(x, t)$ = trajectory of M with $\zeta(\cdot, 0) = \xi$,
(η, E) = a perturbed trajectory of M_* with $\eta(\cdot, 0) = \phi_*(\xi)$,
where the set of faults E is (r, R)-sparse.
Then for all t:

$$\zeta(\cdot, t) = \phi^*(\eta(\cdot, tU)),$$

the original computation is decoded from perturbed trajectory η.

How do the colony size Q and the number of simulating states $|\mathbb{S}_*|$ depend on $|\mathbb{S}|$?
Same question: how does the simulation compute the simulated transition function g?
Two possibilities:

- The computation is essentially just one step, since the `Work` field of a simulating cell can contain a whole state of the simulated machine. But in this case the set of simulating states $\mathbb{S}_*$ is larger than set of simulated states $\mathbb{S}$. Does not scale up.
- The transition function g is computed on a `Work` track of constant size independent of $\mathbb{S}$. A program of the transition function is written onto the `Work` track before the computation, and the simulating computation computes this function somehow using the program.
  This will be our solution, but what is a program, and what does "computation" mean?

Simplest kind of program just a lookup table. For the transition function $g : \mathbb{S}^3 \to \mathbb{S}$, it contains

$$|\mathbb{S}|^3$$

elements, so has a length $|\mathbb{S}|^3 \log_2 |\mathbb{S}|$, written in binary. Then we need essentially $Q \geqslant |\mathbb{S}|^3$.
But then the number of states $|\mathbb{S}_*|$ of the simulating machine (as we designed it, with addresses), is at least $|\mathbb{S}|^3$, so again the simulator is bigger than what it simulates. Does not scale up.

Better

- Define a programming language for transition functions.
- Simulate only cellular automata that have a short program.

Best No program, just compute the transition function "somehow". Our solution will amount to this, but the "somehow" needs explanation!

1. (Larry) Probabilistic cellular automata. Formulating the main result in context, and some ideas (fields).
2. (Peter) The sparsity method. Its application to proving the nonergodicity of the perturbed two-dimensional Toom rule.
3. (Larry) Reliable simulation in 1 dimension in 1-sparse noise: colonies, the most important fields.
4. (Peter) Elaborating the simulation component of Larry's last lecture. New problems caused by non-1 sparse faults. Forced self-simulation.
5. (Larry) The main tools to deal with larger-scale noise beyond forced self-simulation: Decay and Growth.
6. (Peter, today) Generalized cellular automata and simulation. Amplifiers. Pulling it all together.

Let $1 = \rho_1 < \rho_2 < \dots$ and an appropriate $\beta \geqslant 8$ be given, as in the definition of sparsity. Setting $r = \beta\rho_1$, $R = \rho_2$, we constructed a 1-dimensional cellular automaton that computes reliably in the presence of $(\beta\rho_1, \rho_2)$-sparse, that is 1-sparse noise. We built it as

$$M_* = \mathrm{CA}(\mathbb{S}_*, g_*) = M_1 = \mathrm{CA}(\mathbb{S}_1, g_1)$$

simulating an "arbitrary"

$$M = \mathrm{CA}(\mathbb{S}, g) = M_2 = \mathrm{CA}(\mathbb{S}_2, g_2).$$

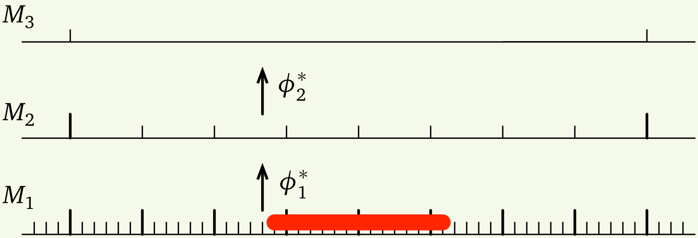
The simulation encoded each cell of M_2 into a colony (block of size $Q_1 = Q < \rho_2/3$) of M_1 via a code

$$\phi = (\phi_*, \phi^*).$$

- 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 (via a code ϕ_2), 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:

  $$M_3 \xrightarrow{\phi_2} M_2 \xrightarrow{\phi_1} M_1.$$

  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.

We hope to build a sequence of machines $M_1, M_2, \dots$ where M_k simulates M_{k+1} via code $\phi_k = (\phi_{k*}, \phi_k^*)$, in the following sense.

Goal Suppose that (η, E) is a perturbed trajectory of M_1, and

$$\eta^1 = \eta, \quad \eta^{k+1} = \phi_k^* \eta^k, \qquad\qquad k = 1, 2, \dots.$$

Then $(\eta^k, E^{(k)})$ is a perturbed trajectory of M_k for all k.

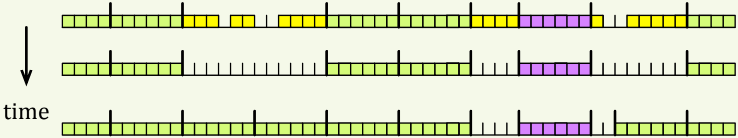
In other words, $\eta^{(k)}$ only disobeys the transition function g_k in $E^{(k)}$, that is in areas of really big noise.

We have not proved this yet, even for $k = 1$. Indeed, our 1-sparse simulation assumed $E^{(2)} = \emptyset$, but now we need the result for arbitrary E.

Let us see what new problems arise.

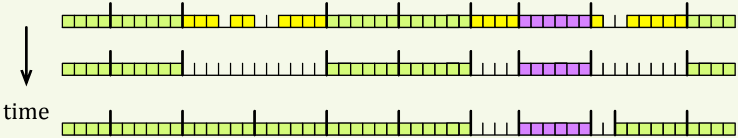
- Nice faults do not change `Addr` and `Age`. Even if the faults were not nice, in our construction the `Addr` and `Age` values were corrected by the rules *Purge*, *Heal*.
- Now faults can wipe out the structure of 3-4 consecutive colonies of M_1 (see red area again). In this case, it makes no sense 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.



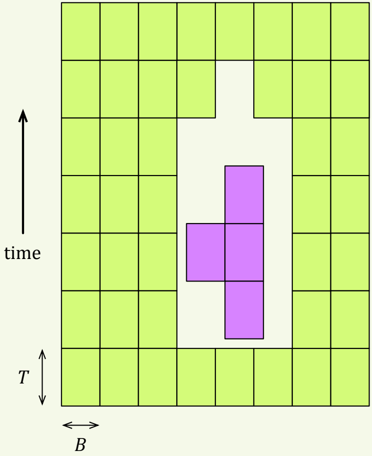
- Rule *Decay* kills a cell for which healing did not solve promptly its inconsistency with a neighbor within its own colony. Repeated application of this will wipe out unhealable partial colonies (yellow cells).
- Rule *Grow* lets a colony extend an arm of consistent cells into nearby vacuum. If new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



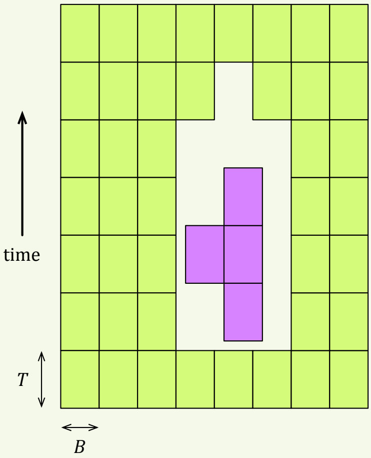
Key idea: the bad colony should eliminate itself.

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.



Generalize histories of M_2 to answer the question: what do perturbed histories of M_1 simulate?
History: Cells have disjoint bodies of length size B (not necessarily adjacent) and disjoint dwell-periods of length T. The dwell-period is what a cell spends in a state between switchings.

Using cell body sizes (and dwell period sizes) allows a colony of Q cells of size B to simulate a cell of size QB occupying the same space.



Questions

- Is there a transition function for the case of neighbor (closer than B) cells that are non-adjacent?
  No, now the "transition function" only imposes some conditions on the trajectory (η, E).
- Do non-adjacent neighbor cells communicate?
  Yes (though not strictly necessary).

We rely on the following mechanism to eliminate bad colonies.

- In case a neighbor colony does not exist (in a consistent way), the program should still proceed. A cell x simulated by a bad colony performs transition g_2, with a program similar to g_1: it also has a *Purge* rule. This *Purge* will kill x (since it is in a small island).
- When the simulated cell x of M_2 dies then all elements of the colony representing it should die. The computational part of the simulation will take care of this: In repetition $i = 1, 2, 3$, a $\mathtt{Doomed}_i$ track records in every cell whether the colony should die—then a last majority vote does the killing.

Automaton M_1 needs the following property:

Forced simulation As long as the structure (`Addr`, `Age`) variables are 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.
- Below, we will show how to pass on the forced simulation property also to the simulated machine M_2.

- The whole transition function can be specified as a sequence of rules like *Purge*, *Heal* above—that is of the type

  if condition on fields of self or neighbors **then**
  assignment to some field
  else
  ...

  The sequence can be written as a single string (say, of bits) R.
- There is a couple of extra primitives in the rules. First, they have access to a parameter k, to define the transition function

$$g_{R,k}(a, b, c)$$

  of automaton M_k.

The other important new primitive is a special instruction

$$\mathit{Write\text{-}rules\text{-}bit}\ .$$

When called, it makes the assignment $\texttt{Work} \leftarrow R(\texttt{Index})$, where $\texttt{Index}$ is a certain field whose value is interpreted as a number. This is the key to self-simulation: the program has access to its own bits.

Let us fix some computationally universal cellular automaton $\mathbf{U}$. By convention, program P and input X produce in it an output $f_{\mathbf{U},P}(X)$. Since the structure of all rules is very simple, they can be read and interpreted by $\mathbf{U}$ in reasonable time:

Theorem There is a constant string called Interpr with the property that for all positive integers k, strings R, A, B, C where R is a sequence of rules, and bit strings $A, B, C \in \mathbb{S}_k$:

$$f_{\mathbf{U},\mathrm{Interpr}}(R, 0^k, A, B, C) = g_{R,k}(A, B, C).$$

The computation on $\mathbf{U}$ takes time $O(|R| \cdot |A|)$.

The proof parses and implements the rules. Implementing the *Write-rules-bit* instruction is natural: Machine $\mathbf{U}$ determines the number i represented by the simulated `Index` field, looks up $R(i)$ in R, and writes it into the simulated `Work` field.

Why is there no circularity in these definitions?

- The instruction *Write-rules-bit* is written literally in R in the appropriate place: the string R is not part of the rules (that is of itself...).
- On the other hand, machine **U** has explicit access to the string R as one of the arguments.

In the earlier outline, there was a step saying: "apply the simulated transition function g". We give more detail now, to implement forced simulation:

- Onto track `Work`, write:
  - String Interpr.
  - String R representing the set of rules. To do this: for `Index` running from 1 to $|R|$, execute the instruction *Write-rules-bit* and move right.
  - 0^{k+1} (with the help of parameter k).
  - Strings A, B, C copied from the three neighbor colonies representing the simulated cell states.
- Simulate the universal automaton $\mathbf{U}$ on track `Work`: it computes $g_{R,k+1}(A, B, C) = f_{\mathbf{U},\mathrm{Interpr}}(R, 0^{k+1}, A, B, C)$.

This achieves the forced simulation: the correct sequence R of rules will be used even if the corresponding part of the workspace was completely corrupted by noise before the start of the work period.

- On level 1, the transition function $g_{R,1}(a, b, c)$ is defined completely when the rule string R is given. It has the forced simulation property by definition, and string R is "hard-wired" into it in the following way:

$$g_{R,1}(a, b, c).\mathtt{Work} = R(b.\mathtt{Index})$$

  whenever $b.\mathtt{Index}$ represents a number between 1 and $|R|$, and $b.\mathtt{Age}$ satisfies the condition under which the instruction *Write-rules-bit* is called in the rules (written in R).
- The forced simulation property of the simulated transition function $g_{R,k+1}(A, B, C)$ is achieved by the above defined computation step—which relies on the forced simulation property of $g_{R,k}(a, b, c)$.

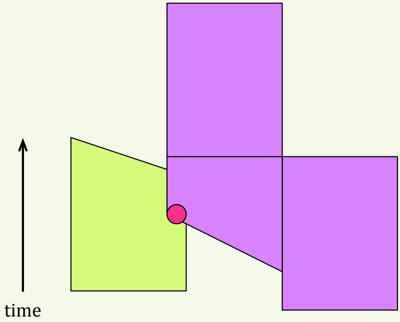
With more rules killing and reviving cells:

$$\mathit{Purge}\,,\ \mathit{Heal}\,,\ \mathit{Decay}\,,\ \mathit{Grow}\,,$$

it becomes harder to make sure that they do not conflict with each other. Most of these potential conflicts are solved as follows:

- *Purge* and *Heal* are fast, but are restricted to a small range.
- *Decay* is slow.
- *Grow* is slow and is acting only in part of the work period.

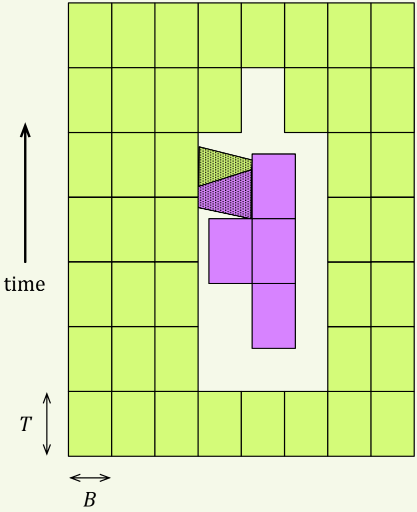
Delicate situation that might allow a single burst to affect events on the colony level:



A bad colony's growth, as it completes the creation of a neighbor, bites into a good colony (due to a burst).

Two possible solutions.

- If there is communication among non-adjacent neighbors, *Heal* is made stronger than *Grow*: it kills neighboring growth cells that are in its way.
- If there is no such communication, there is a less natural solution: let *Grow* work in a zigzag way: say, $2c$ steps forward, c steps back for some constant c. This gives the good colony a chance to heal after a possible faulty bite.



To communicate, non-adjacent colonies extend communicating arms. If the work periods intersect substantially, the arms will live long enough to carry information. In borderline cases, there is nondeterminism about which work-period of your neighbor colony you will communicate with. See the definition of trajectories below.

The cell body size B and the cell dwell period lenght T become part of the definition of a generalized cellular automaton

$$\mathrm{CA}(\mathbb{S}, g, B, T).$$

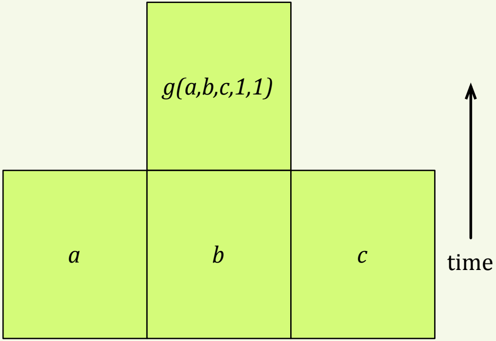
In the transition function $g(a, b, c, L, R)$, the bit L says whether the left neighbor is adjacent and aligned; R says the same about the right neighbor.

A history consists of

- A set $\mathcal{R}$ of starting points $\{(x_i, t_i)\}_{i=1}^{\infty}$ of disjoint space-time rectangles $[x_i, x_i + B) \times (t_i, t_i + T]$.
- A map $\eta : \mathcal{R} \to \mathbb{S}$ assigning a state to each rectangle. We say $\eta(x, t)$ is live if $(x, t) \in \mathcal{R}$, and vacant otherwise.

The set of histories and configurations of machine M is denoted by $\mathrm{Histories}(M)$ and $\mathrm{Configs}(M)$ respectively.

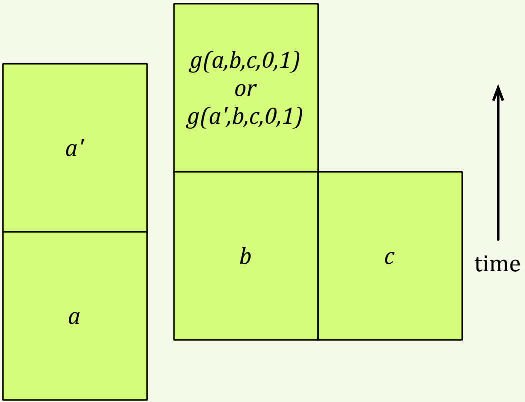
We now have to say when a history η along with exception set E is a trajectory (η, E) of $\mathrm{CA}(\mathbb{S}, g, B, T)$. In all conditions below, assume there is no exception point (fault) near (x, t):
$[x - 5B, x + 5B) \times (t - 5T, t] \cap E = \emptyset$.



Simplest case: if $(x - B, t), (x, t), (x + B, t), (x, t + T) \in \mathcal{R}$ then

$$\eta(x, t + T) = g(\eta(x - B, t), \eta(x, t), \eta(x + B, t), 1, 1).$$

There are corresponding (typically non-deterministic) conditions for neighbors that are not adjacent and aligned.



Examples:

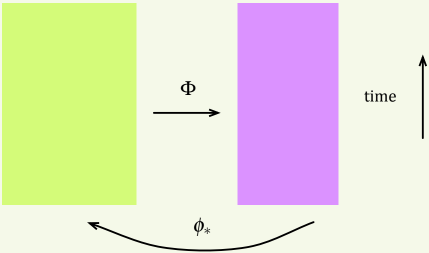
- (a) Suppose $(x, t), (x + B, t), (x, t + T) \in \mathcal{R}$, and there is no left adjacent-aligned neighbor. Then still $\eta(x, t + T) = g(r, \eta(x, t), \eta(x + B, t), 0, 1)$ for some $r \in \mathbb{S}$, but the condition does not specify r.
- (b) Suppose that in addition to the above, there is a left non-adjacent-aligned neighbor $B \leqslant B' < 2B$ and a $t - T < t' \leqslant t$ with $(x - B', t'), (x - B', t' + T) \in \mathcal{R}$.
  Then the output value is based on one of the two possible left inputs, we do not specify which:

$$\begin{aligned} \eta(x, t + T) = g(r, \eta(x, t), \eta(x + B, t), 0, 1), \;\text{ where} \\ r \in \{\eta(x - B, t'), \eta(x - B', t' + T)\}. \end{aligned}$$

Definition (Simulation) The pair of mappings (Φ, ϕ^*)

$$\Phi : \text{Histories}(M_1) \times \text{Noises} \to \text{Histories}(M_2) \times \text{Noises},$$
$$\phi_* : \text{Configs}(M_2) \to \text{Configs}(M_1)$$

is a simulation of machine M_2 by machine M_1 if for every $\xi \in \text{Configs}(M_2)$, for every trajectory of η of M_1 with $\eta(\cdot, 0) = \phi_*(\xi)$, the value $(\eta^*, E^*) = \Phi(\eta, E)$ is a trajectory of M_2.



Example In a block simulation with decoding function ϕ^*, and empty exception sets, $\Phi(\eta^1, \emptyset) = (\eta^2, \emptyset)$ where we obtain $\eta^2(\cdot, t)$ by decoding η^1 at time tU:

$$\eta^2(\cdot, t) = \phi^*\Big(\eta^1(\cdot, tU)\Big).$$

- Recall the definition of k-noise. We fix a sequence of scales $\rho_1 < \rho_2 < \cdots$: the details are not important now. If E is a space-time set then we denoted by $E^{(k)}$ its k-noise.
- Our goal is to define a sequence of generalized cellular automata and simulations:

$$M_1 \stackrel{\Phi_1}{\to} M_2 \stackrel{\Phi_2}{\to} M_3 \stackrel{\Phi_3}{\to} \cdots .$$

This object will be called an amplifier. If $(\eta^1, E^{(1)})$ is an initial history then denote $(\eta^{(k+1)}, E^{(k+1)}) = \Phi_k(\eta^k, E^{(k)})$.

- The cellular automaton M_1 is an ordinary, non-generalized one, with trajectory $(\eta^1, E^{(1)})$.
- We define the amplifier's action on the exception set E independently of η: if $(\eta^*, E^*) = \Phi_k(\eta, E)$ then $E^* = D(E, \beta\rho_k, \rho_{k+1})$, hence

$$(E^{(k)})^* = E^{(k+1)}.$$

The simulation Φ_k will be associated with a block code

$$(\phi_{k*}, \phi_k^*),$$

with block size Q_k, and a block simulation with work period size U_k.
The cell body size and dwell period size of cellular automaton M_{k+1} satisfy $B_{k+1} = Q_k B_k$, $T_{k+1} = T_k U_k$.
Eventually we want to make implications not only from lower levels to higher levels, but also from higher levels to lower ones.
Then a simple result (found, say, in a 1 bit field `Rider`) need not be decoded.

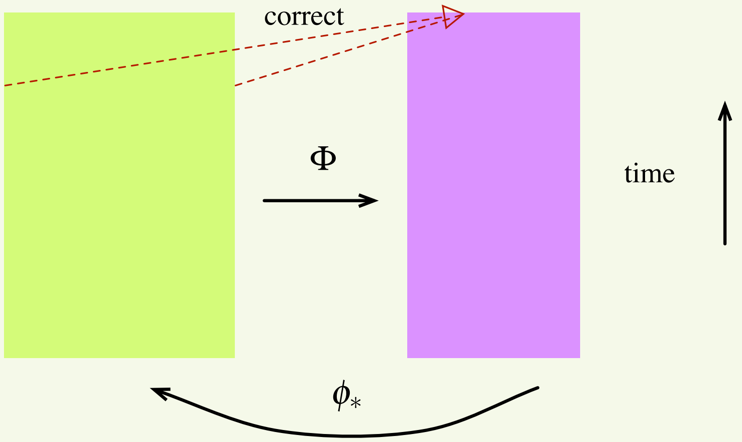
Definition Suppose that each cellular automaton M^k has a distinguished field `Rider`. The amplifier has the error-correction (trickle-down) property in field `Rider` if the following holds for each k.

- (a) For any symbol s, the `Rider` track of the encoded string $\phi_{*k}(s)$ only depends on the field $s.\mathtt{Rider}$.
- (b) Let $\eta^{k+1}(x, t)$ be live, and let $(x', t') = (x + aB_k, t + uT_k)$ for some $0 \leqslant a \leqslant Q_k,\ 0 \leqslant u < U_k$. If

$$\big(\,(x', t') + [-5B_k, 5B_k) \times (-5T_k, 0]\,\big) \cap E^{(k)} = \emptyset$$

then $\eta^k(x', t').\mathtt{Rider} = \phi_{*k}(\eta^{k+1}(x, t))(a).\mathtt{Rider}$.

So in the absence of recent k-noise near (x', t') its field `Rider` is "correct" on level k: as if obtained by decoding into $\eta^{k+1}.\mathtt{Rider}$ and encoding again into $\eta^k.\mathtt{Rider}$.


correct
Φ
time
ϕ_*

Lemma (Main) For a wide range of choices of the parameters, there is an amplifier with the error-correcting property for some field `Rider`.

The amplifier is essentially given by just the program of the simulation g_1 of M_2 by M_1, along with the code ϕ_1.
This program will satisfy the error-correcting property automatically: the last step of the computation replaces information track of the colony with the encoding of the new value of the cell represented by it. The `Rider` track is a subtrack included in this.

- Assume, say, that the field `Rider` has the same size on all levels, and in the encoding ϕ_{*k} the value `Rider` is just repeated in the `Rider` field of each member cell of the colony.
- Assume that we start with, say, $\eta(x, 0).\mathtt{Rider} = 1$ for all x.
- We need to construct an initial configuration with the property

$$M_1 \overset{\phi_{1*}}{\leftarrow} M_2 \overset{\phi_{2*}}{\leftarrow} M_3 \overset{\phi_{3*}}{\leftarrow} \cdots$$

- Trick: make first and last symbols of the string $\phi_{*k}(s)$ depend only on the symbol $s.\mathtt{Rider}$.
  Then every finite part of the infinite initial configuration is determined already by a finite number of hierarchy levels.

- For space-time point (x, t) let $(x_0, t_0) = (x, t)$,

$$x_k = x - (x \bmod B_k),$$
$$t_k = t - (t \bmod T_k).$$

  There is a K with $t_K = 0$.
- For $b \in \{0, 1\}$, assume that $\eta(x, 0).\mathtt{Rider} = b$ for all x.
- $\mathcal{G}_k(b) =$ the event $\eta^k(x_k, t_k).\mathtt{Rider} = b$. $\mathcal{G}_K(b)$ holds.
- $\mathcal{F}_k =$ the event

$$((x, t) + [-T_{k+1}, 0] \times [-B_{k+1}, B_{k+1}]) \cap E^{(k)} = \emptyset.$$

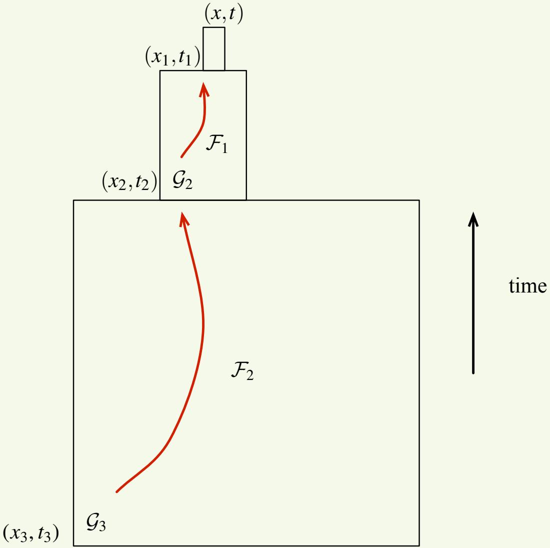
Then $\mathcal{G}_{k+1}(b) \wedge \mathcal{F}_k \Rightarrow \mathcal{G}_k(b)$ by the error correction property.

The Sparsity Bound gives $\mathbb{P}(\bigcap_k \mathcal{F}_k) > 1 - O(\varepsilon)$. Assuming that $\mathcal{F}_k$ holds for all k:

$$\mathcal{G}_K(b) \overset{\mathcal{F}_{K-1}}{\Longrightarrow} \mathcal{G}_{K-1}(b) \overset{\mathcal{F}_{K-2}}{\Longrightarrow} \cdots \overset{\mathcal{F}_0}{\Longrightarrow} \mathcal{G}_0(b),$$

hence $\eta(x,t).\texttt{Rider} = b$, so we derived the desired

$$\mathbb{P}\{\, \eta(x,t).\texttt{Rider} = b \,\} \geqslant 1 - O(\varepsilon).$$



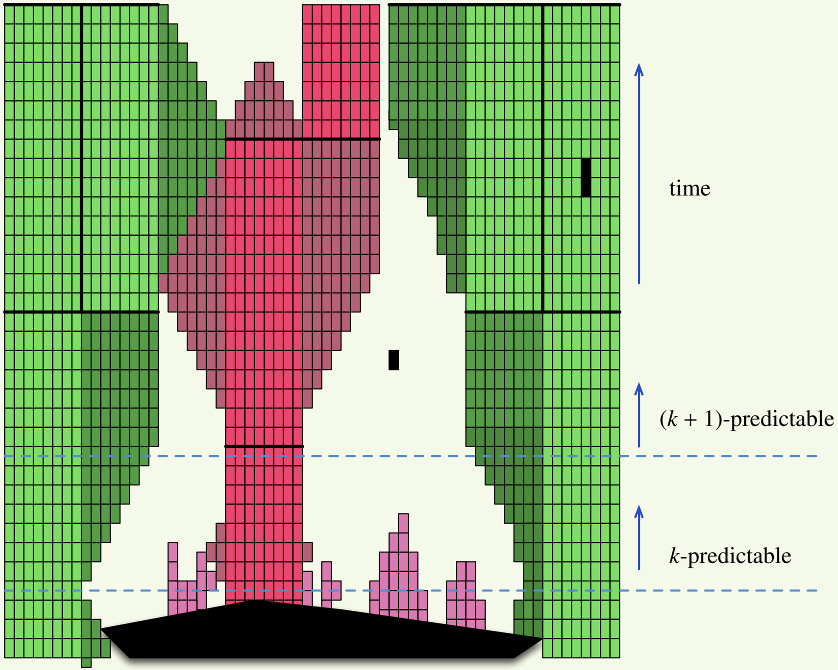
(x,t)
(x_1,t_1)
$\mathcal{F}_1$
(x_2,t_2)
$\mathcal{G}_2$
$\mathcal{F}_2$
time
$\mathcal{G}_3$
(x_3,t_3)

16 Some proof details

- Need to show that in the absence of $(k+1)$-noise, the $(k+1)$-cells decoded from existing colonies in the k-trajectory behave as they have to in a $(k+1)$-trajectory.
- Most difficult part is starting from a complete mess. The current absence of $(k+1)$-noise still allows arbitrary noise in the near past.

Stages of recovery (not strictly separated in time) and reasoning about them:

1. By induction, since we are in a k-trajectory, we can start reasoning about the history in terms of k-cells, very soon (say, in time $3T_k$) after any big noise.
2. Partial colonies are eliminated via the Decay rule.
3. The Grow rule creates a placeholder for an adjacent big cell when needed.
4. In full colonies, the program simulates $(k+1)$-cells very soon (say, within time $3T_{k+1}$).
5. In all this, a single k-burst (within the space-time window considered)
    - does not change much outside full colonies (due to Purge),
    - is corrected inside full colonies (due to Purge and Heal).


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

Opening a gap Decay opens a gap, unbridgeable by Heal, unless Heal succeeds promptly.

Widening gap Such a gap will widen, until reaching a colony boundary.

Path A live cell that is free (not in the shadow of a burst) starts a traceback path of predecessors or consistent neighbors. Such a path will pass around any burst of faults (this uses Purge).

Finding a full colony in the past A traceback path must lead to a full colony, otherwise it would encounter a widening gap (impossible).

Follow the colony's development forward to see that every free live cell belongs to an (extended) colony.

- The details are tedious…