

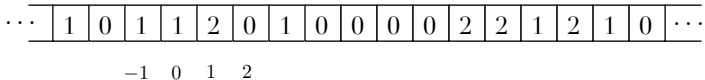
Reliably computing cellular automata

Peter Gács

Boston University

- Elementary parts: **cells**, or **sites**. Set of cells: for example, $\mathbb{C} = \mathbb{Z}^3$, or $\mathbb{C} = \mathbb{Z}/m\mathbb{Z}$ (**periodic boundary conditions**).
- Finite set \mathbb{S} of (local) **states**.
- (Space-) **configuration**: any function $\xi : \mathbb{C} \rightarrow \mathbb{S}$.

$$\mathbb{C} = \mathbb{Z} \quad \mathbb{S} = \{0, 1, 2\}$$



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

Space-time configuration $\eta(x, t)$.

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	2	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										

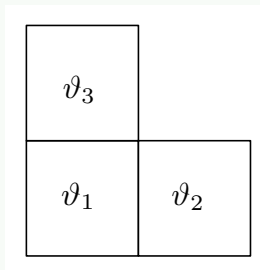
$$\eta(1, 2) = 2, \eta(2, 2) = 1, \dots$$

Neighborhood function: $N(x) = \{\vartheta_1(x), \dots, \vartheta_r(x)\}$.

Normally $\mathbb{C} = \mathbb{Z}^d$ and we have $\vartheta_i(x) = x + \vartheta_i(\mathbf{0})$.

Examples

- **von Neumann** neighborhood: the 7 nearest neighbors (including itself) of a point, say, in the lattice \mathbb{Z}^3 .
- **Toom** neighborhood:
 $(\vartheta_1(\mathbf{0}), \vartheta_2(\mathbf{0}), \vartheta_3(\mathbf{0})) = ((0, 0), (0, 1), (1, 0))$.



In discrete time, we say η is a **trajectory** of **local transition function** $g : \mathbb{S}^r \rightarrow \mathbb{S}$ if

$$\eta(x, t+1) = g(\eta(\vartheta_1(x), t), \dots, \eta(\vartheta_r(x), t)).$$

Example

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

1	0	1	1	2	0	1	0	0	0	0	2	2	1	2	1	0	t
																	$t+1$

-1 0 1 2

$$\uparrow$$

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

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

$$\mathbf{A} = \text{CA}(\mathbb{C}, \mathbb{S}, r, \vartheta, g).$$

Example (The Toom Rule)

$$\mathbb{C} = \mathbb{Z}^2, \mathbb{S} = \{0, 1\},$$

$$N(\mathbf{0}) = ((0, 0), (0, 1), (1, 0)),$$

$$g(x, y, z) = \text{Maj}(x, y, z).$$

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

$\text{Maj}(x, y, z)$ can be extended to the case of larger alphabets: when no symbol is in majority, let the result be y .

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 \mathbf{A} starting with ξ .
- Though a cellular automaton never halts, we can designate a **halting state** whose occurrence in cell $\mathbf{0}$ signals the end.
- Some convention can then find the **output** at the halting time.

Assume these conventions fixed now and for all.

This allows to define the (possibly partial) function $f_{\mathbf{A},P}(X)$ computable on cellular automaton \mathbf{A} with program P .

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

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.

Fault tolerance

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.
- Their parallelism provides power to combat noise that occurs all over space-time.

From now on, our model of computation for the purposes of fault-tolerance is a **probabilistic cellular automaton**.

Probabilistic cellular automata

For simplicity of notation, we assume that the automaton is in 1 dimension, with $x-1, x, x+1$ used as the neighbors of cell x . Now instead of a local transition function $g : \mathbb{S}^3 \rightarrow \mathbb{S}$, we have **local transition probability matrix** $W : \mathbb{S}^4 \rightarrow [0, 1]$ with

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

A **stochastic process** $\eta(x, t)$ is a **trajectory** for matrix $W(\cdot)$ if for any space-time configuration $\zeta(x, t)$, and sites $x_1 < \dots < x_n$:

$$\begin{aligned} & \mathbf{P}\{\eta(x_i, t) = \zeta(x_i, t), i = 1, \dots, n \mid \eta(\cdot, t-1) = \zeta(\cdot, t-1)\} \\ &= \prod_{i=1}^n W(\zeta(x_i, t), \zeta(x_i-1, t-1), \zeta(x_i, t-1), \zeta(x_i+1, t-1)). \end{aligned}$$

Thus, the local transitions occur independently at different sites, according to the transition matrix. In general, our probabilistic automaton is defined as

$$\mathbf{A} = \text{CA}(\mathbb{C}, \mathbb{S}, r, \vartheta, W).$$

(we just replaced the function g with the transition matrix W).

- A probabilistic cellular automaton is **noisy** if all of its transition probabilities are positive (no prohibited local transitions).
- Our probabilistic cellular automaton of main interest is an **ε -perturbation** of a deterministic automaton g :

$$W(s, r_{-1}, r_0, r_1) \leq 1 - \varepsilon \text{ if } s = g(r_{-1}, r_0, r_1).$$

The other values are arbitrary, but in the noisy case, they are **all positive**.

Let $\eta(x, t)$ be a trajectory of a probabilistic cellular automaton \mathbf{A} .
Let

$$\mu_t$$

be the probability distribution of the random space-time configuration $\eta(\cdot, t)$. Then 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 there are always invariant measures.

A PCA with transition operator P is **ergodic** if

- There is only one invariant measure ν .
- For every space-time trajectory, the measures μ_t converge **weakly** to the unique invariant measure ν .

Weak convergence: convergence on all sets of the form

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

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

Nonergodic noisy cellular automata

- A noisy cellular automaton on a finite space is always ergodic. 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!
- But it was a challenge to construct a **noisy non-ergodic** 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.

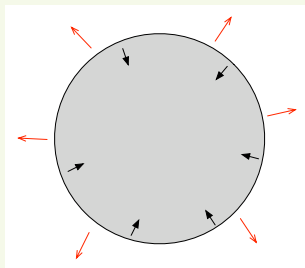
- Idea: some kind of **local voting**.
- In 1 dimension, hopeless: suppose we started from all 0's. Eventually, a large island of 1's appears.

00000000000111111111111111111100000000000000

A local voting-type rule cannot eliminate it (sufficiently fast) since it cannot figure out whether the island side is the 1's or the 0's. (Theorems of **Gray**.)

Symmetric voting in 2 dimensions?

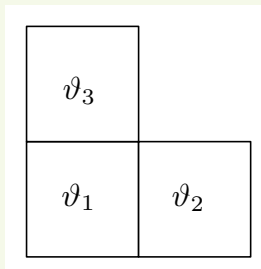
- Voting in the 5-element von symmetric neighborhood?
Will not decrease a large rectangle of 1's in a sea of 0's (in the absence of noise).
- Any symmetric local voting (including the center) will **decrease** a large disk of radius r of 1's only with a speed $1/r$. If the noise is **biased**, favors the appearance of 1's, it **increases** the disk with a constant speed ε .
Result: increase with speed $\varepsilon - 1/r > 0!$ Even if we started with all 0's, the 1's win out.



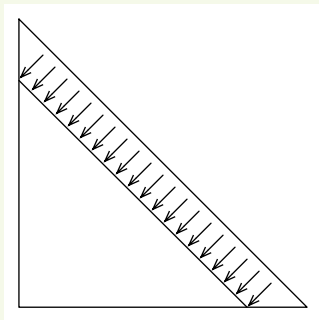
Symmetric voting, unbiased noise

- 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:



The proof of non-ergodicity of (any ε -perturbation with small ε) of this rule is not easy, it will be sketched later today.

Application to reliable computation

Layering

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

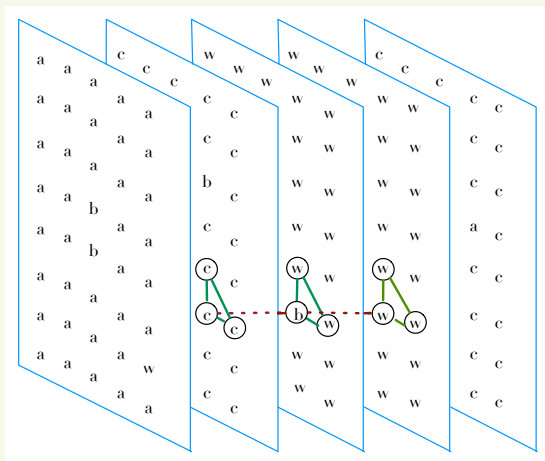
Definition (Toom-layering)

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? Consider a random process $\eta(u, t)$ (discrete t) that follows the transition rule \mathbf{U}' only approximately: at each space-time point (u, t) , the transition rule is applied except with some probability $< \varepsilon$, a **fault** occurs, when $\eta(u, t)$ becomes something else. We assume that faults occur **independently** of each other.

Theorem (Reliable computation, infinite version)

There is a constant c with the following property. Let $\zeta(x, t)$ be a computation (space-time configuration) of \mathbf{U} , and let $\eta(x, y, z, t)$ be a space-time configuration of the Toom-layering \mathbf{U}' with noise bound ε , such that for all x, y, z we have $\eta(x, y, z, 0) = \zeta(x, 0)$. Then for all $x, y, z \in \mathbb{Z}$, $t \in \mathbb{Z}_+$ we have

$$\mathbf{P} [\eta(x, y, z, t) \neq \zeta(x, t)] \leq c\varepsilon.$$

Finite versions

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

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

(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 $\eta_i(x, t)$ be the process starting from ξ_i . We call (for simplicity) the **relaxation time** $R_\varepsilon(L)$ the smallest time t_0 such that for all $t \geq t_0$:

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

This shows the time it takes for the information about index i of the initial configuration ξ_i to be erased from $\mathbf{P}\{\eta_i(\mathbf{0}, t) = 0\}$.

Proposition

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

(The proof should be easy.) Thus, if the infinite system is ergodic then there is no use building a larger cellular automaton, this will not help us in fighting the noise.

On the other hand, there is a proof (by [Berman-Simon](#)) of Toom's theorem showing that the relaxation time grows *exponentially* with the size L :

Theorem (Toom)

For a perturbed version of Toom's rule, there is a constant $c(\varepsilon) > 0$ with $R_\varepsilon(L) > 2^{c(\varepsilon)L}$.

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

$$c\varepsilon + t \cdot 2^{-dL}$$

for some $d > 0$.

- 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, δ . In this case, our computing device should be a **sausage-like** torus

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

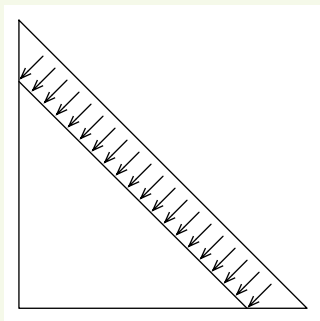
Along the **computing** dimension of length S we perform the computation, along the **stabilizing** dimensions of length L , the Toom rule. The result in each position along the computing dimension is obtained at time T by majority vote along the stabilizing dimensions.

The proof of the reliability of the Toom-Reif-Gács 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.

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 (noise of level 1).
- Then allow violations of this, but assume that those violations have low frequency (noise of level 2).
- And so on.

With defining “low frequency” here judiciously, hopefully this classification covers all cases.

More precisely, let us introduce a sequence $1 = \rho_1 < \rho_2 < \dots$ of distances:

$$\rho_{n+1} = Cn^2 \rho_n,$$

where C will be chosen appropriately. By the **distance** of two points $\mathbf{x} = (x_1, x_2, x_3)$ and $\mathbf{y} = (y_1, y_2, y_3)$ we mean

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

(l_∞ distance). Let

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

be an open ball (it is actually a cube). The distance $d(S_1, S_2)$ of two sets is defined as usual.

Let $E \subseteq \mathbb{Z}^3$ be a set of space-time points. The k -noise $E^{(k)}$ of E (and with it, the $> k$ -noise $E^{(>k)} = E \setminus E^{(k)}$) is defined recursively as follows.

- $E^{(0)} = \emptyset$.
- For $k > 1$, $E^{(k)}$ consists of points of $E^{(k-1)}$, and in addition the set of those points $x \in E^{(>k-1)}$ for which

$$d(B(x, \rho_k), E^{(>k-1)} \setminus B(x, \rho_k)) > \rho_{k+1}.$$

So, for each $k > 0$, $E^{(k)} \setminus E^{(k-1)}$ can be covered by balls $B(x_i, \rho_k)$, at a distance at least ρ_{k+1} from each other.

- The set E is k -sparse if $E = E^{(k)}$. It is k -sparse on set A if $E \cap A$ is k -sparse. In particular it is 1-sparse if it consists of points at a distance at least ρ_2 from each other.

The following observation is key.

Proposition (Sparsity Bound)

For small enough ε , the following holds for all $k \geq 0$. Assume that $\mathcal{E} \in \mathbb{Z}^3$ is a random set, where each point x is in \mathcal{E} independently from all the others, with probability ε . Then for each point x and each k ,

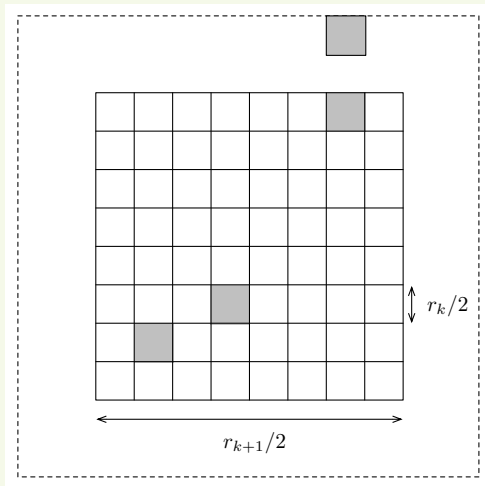
$$\mathbf{P}\{B(x, \rho_{k+1}) \cap \mathcal{E}^{(>k)} \neq \emptyset\} < 2\varepsilon \cdot 2^{-(1.5)^k}.$$

The probability that the noise in $B(x, \rho_{k+1})$ is not k -sparse is decreasing doubly exponentially with k .

(Even for $k = 0$, for the probability of any fault at all in $B(x, \rho_1)$ (which consists of a single point), this gives $< \varepsilon$.)

Proof sketch

Instead a similar bound for $B(x, \rho_{k+1}/4)$, a cube of size $\rho_{k+1}/2$.



$E^{(>k)}$ must intersect two subcubes $B(x_i, \rho_k/4)$, $B(x_j, \rho_k/4)$ separated by at least $\rho_k/4$, of the cube $B(x, \rho_{k+1}/4 + \rho_k)$.

Let $\beta = 1.5$. The probability for each i, j is by induction bounded by $2\varepsilon \cdot 2^{-\beta^k}$. Because of the **separation** of the cubes, these probabilities multiply for each pair i, j :

$$4\varepsilon^2 \cdot 2^{-2\beta^k} = 2\varepsilon \cdot 2^{-(2-\beta)\beta^k} \cdot 2\varepsilon \cdot 2^{-\beta^{k+1}}.$$

Each i has at $< (2\rho_{k+1}/\rho_k)^3$ possibilities in the cube. The total number of pairs i, j is bounded by

$$(2\rho_{k+1}/\rho_k)^6 = 4C^6 k^{12},$$

so the total probability is bounded by

$$4C^6 k^{12} \cdot 2\varepsilon \cdot 2^{-(2-\beta)\beta^k} \cdot 2\varepsilon \cdot 2^{-\beta^{k+1}} < 2\varepsilon \cdot 2^{-\beta^{k+1}}$$

for all k , if ε is small.

Noise is a concept in space-time, **damage** is the corresponding concept in space $\mathbb{C} = \mathbb{Z}^2$. We can thus talk about the k -damage D_k of a set $D \subseteq \mathbb{C}$. We say that D is **k -sparse** if $D_k = D$. We say it is k -sparse on set A if $D \cap A$ is k -sparse.

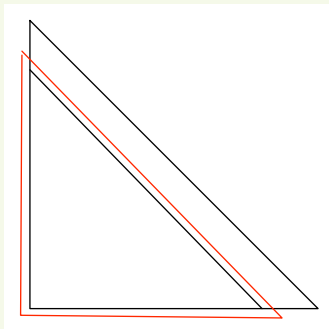
Shrinking triangle in noise

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

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

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 $z \in \mathbb{C}$ an arbitrary point, $B(d) = B(z, d)$, $\Delta > \delta > 0$. **Conditions:**

- a At time $t_0 - \delta$, the set of 1's is k -sparse on $B(\Delta) \setminus T(a, b, c)$.
- b The set of faults is k -sparse on $B(\Delta) \times [t_0 - \delta, t_0]$.

Proposition

Under these conditions, if $\delta > \rho_{k+1}$ then at time t_0 , the set of 1's is k -sparse on

$$B(\Delta - \delta) \setminus T(a - v_k \delta, b - v_k \delta, c - (1 - v_k) \delta).$$

Without noise and damage, this would be $B(\Delta - \delta) \setminus T(a, b, c - \delta)$. Size shrinks by $\delta(1 - 3v_k)$ instead of by δ .

Corollary

Suppose now, with $\delta = 4\rho_{k+1}$:

- a At time $t_0 - \delta$, the set of 1's is $(k + 1)$ -sparse on $B(\Delta)$.
- b The set of faults is k -sparse on $B(\Delta) \times [t_0 - \delta, t_0]$.

Then at time t_0 , the set of 1's is k -sparse on $B(\Delta - \delta)$.

Indeed, the violations of k -sparseness at time $t_0 - \delta$ are enclosed in balls of the form $B(x, \rho_{k+1})$, separated by distances of ρ_{k+2} . The balls are contained in triangles of size $2\rho_{k+1}$. All these will be “erased” in time $4\rho_{k+1}$, according to the Proposition.

Application to the Toom theorem

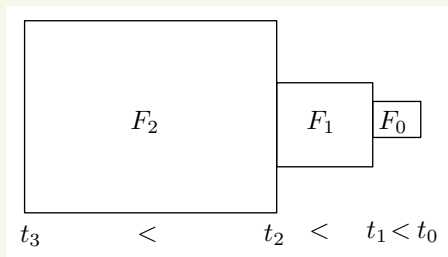
Suppose that the evolution of an ε -perturbation of the Toom rule started from all 0's. Let (x, y, t_0) be a space-time point, $t_k = t_0 - 4(\rho_1 + \dots + \rho_k)$, and let us now define the ball $B_k = B((x, y), 2\rho_k)$.

- Let G_k be the event that the set of 1's is k -sparse on B_k at time t_k . True for a sufficiently large k , since there were no 1's at time 0.
- Let F_k be the event that the set of faults is k -sparse in $[t_{k+1}, t_k] \times B_{k+1}$. The Sparsity Bound gives a constant C_1 with $\mathbf{P}(\bigcap_k F_k) > 1 - C_1 \varepsilon$.

By the Corollary, G_{k+1} and F_k implies G_k . Assuming that F_k holds for all k :

$$G_k \xrightarrow{F_{k-1}} G_{k-1} \xrightarrow{F_{k-2}} \cdots \xrightarrow{F_0} G_0,$$

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



The Proposition is proved by induction.

- Inductive assumption gives shrinking with velocity $1 - 3v_{k-1}$ when the noise and the initial damage are $(k - 1)$ -sparse, instead of k -sparse.
- k -damage brings in some blocks of size ρ_k , separated from each other by ρ_{k+1} . Same for k -noise.

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.

The detailed proof just implements these ideas.

This is the second one of a series of three lectures on reliable cellular automata. The first lecture constructed a reliably computing 3-dimensional cellular automaton. The style of proof chosen in that lecture is not essential for the theorem proved there, but is essential for what follows now. Please recall the definitions of k -noise, k -sparse set of faults in space-time and the corresponding notions of damage, and k -sparse damage in space.

Reliable computation in 3-dimensional cellular automata happens to have a very simple solution. The simplicity of this solution seems to be an anomaly.

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 the most economical way to introduce redundancy.

From now on, we will always work with 1-dimensional cellular automata, of the form

$$M = \text{CA}(\mathbb{C}, \mathbb{S}, r, \vartheta, g) = \text{CA}(\mathbb{Z}, \mathbb{S}, 3, \{-1, 0, 1\}, g) =: \text{CA}(\mathbb{S}, g),$$

that is we only indicate the set of states and the transition function, since \mathbb{C}, r, ϑ are fixed.

- The difficulty of the problem suggests to seek solution first just for a 1-sparse set of errors. Recall from the previous talk: a set E of errors is 1-sparse if it consists of points at some large distance ρ_2 from each other.
- However, in this case (say in 1 dimension): one might be able to **cheat** by massing a lot of information into a cell and then doing local majority votes.

- The cheating solution is useless since it **does not scale up** for dealing with errors that are not- 1-sparse: it crams too much information into a cell of ζ_* .
- Let us generalize 1-sparsity slightly, in order to prevent cheating. In the definition of k -noise and k -sparsity, allow $\rho_1 > 1$, calling it $2\rho_1$ the **burst size**. The new property will still be called 1-sparsity: in case of misunderstanding, it is called (ρ_1, ρ_2) -**sparsity**. Equivalently, such a sparse set can be covered by ρ_1 -balls that are at distance at least ρ_2 from each other.
- In the cheating trick applied to this case, the number of states of the fault-tolerant cellular automaton is an **exponential function** of ρ_1 . We are looking for solutions with a number of states that grows only **polynomially** in ρ_1 .

Let us treat encoding-decoding more explicitly than in the layering case.

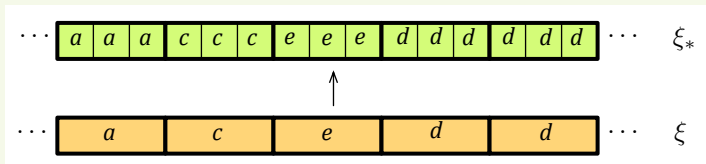
We must store information, in order not to lose it to noise, with **redundancy**: using extra space. The introduction of redundancy is generally called **coding**. Let X be a space whose elements are the possible values of our information, and Y some other space. The pair of mappings

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

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

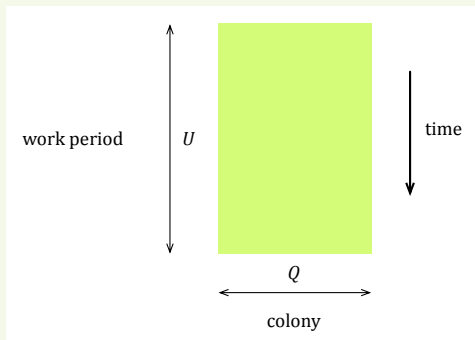
$$\phi_*(x) = (x, x, x), \quad \phi^*(x, y, z) = \text{Maj}(x, y, z).$$

- Code (ϕ_*, ϕ^*) is called a **block code** with block size Q if $X = \mathbb{A}$, $Y \subseteq \mathbb{A}_*^Q$ 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 $\mathbb{C} = \mathbb{Z}$. Example:



Decoding can also be extended (This extension is not automatically shift-invariant, see later.)

In a block simulation, we simulate the original machine $M = CA(\mathbb{S}, g)$ step-for-step by a new machine $M_* = CA(\mathbb{S}_*, g_*)$.



- One cell x of M will be represented by Q cells of M_* called a **colony**.
- One step of M will be simulated by U steps of M_* called a **work period**.

Theorem (Sparse error correction)

Let $M(\mathbb{S}, g)$ be an arbitrary cellular automaton. For any ρ_1 there is a new machine $M_* = CA(\mathbb{S}_*, g_*)$ with $|\mathbb{S}_*|$ depending polynomially on ρ_1 , $|\mathbb{S}|$, further ρ_2, Q, U depending linearly on ρ_1 , a block code (ϕ_*, ϕ^*) with block size Q , such that the following holds. Let $\zeta(x, t)$ be a trajectory of M with initial configuration ξ . Let $\eta(x, t)$ be an arbitrary space-time configuration of M_* , initial configuration $\phi_*(\xi)$, in which the set of faults is (ρ_1, ρ_2) -sparse. Then for all x, t we have

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

Note: The state space \mathbb{S}_* need not be larger than \mathbb{S} , since only Q cells of M_* must be able to hold a cell of M . We return to this important issue in the third lecture.

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).

How to perform the step $\eta_{tU} \rightarrow \eta_{(t+1)U}$?

In the Toom-layering construction, $U = 1$, and we could just work directly on η_t , step-for-step, for

- 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

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

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

Step toward solution: **structure information functionally** even within individual cells. At any one time, work only on part of the information, protecting the rest from error **propagation**.

- Set of cell states $\mathcal{S} = \{0, 1\}^m$, where m is called the **capacity** of the cell.
- Let $1 \leq f_1 < f_2 < \dots < f_k \leq m$, $F = \{f_1, \dots, f_k\}$. 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** . Typically, F is an interval of bits in the bit string s . This notation is borrowed from the programming languages Pascal, C, and so on.

- Typically, our fields are disjoint intervals of bits. Viewing each cell as a computer processor, fields are its program's **variables** in local memory.
- If $(\xi(x) : x \in \mathbb{C})$ of a cellular automaton, is a configuration, then for each field F the values $\xi(x).F$ can be grouped into a **track**

$$(\xi(x).F : x \in \mathbb{C}).$$

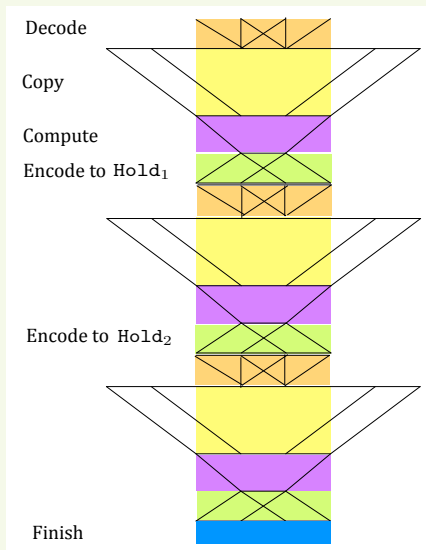
Example, with Info, Addr, Age, Mail tracks:

Info	<i>ua</i>	<i>vw</i>	<i>ax</i>	<i>zf</i>	<i>yy</i>	...
Addr	7	0	1	2	3	
Age	41	41	41	41	41	
Mail	<i>b</i>	<i>a</i>	<i>r</i>	<i>z</i>	<i>x</i>	

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 like `Mail`, `Work`.
- Perform the decode-compute-encode process 3 times. In iteration $i = 1, 2, 3$, store the result in track `Holdi`.
- Replace `Info` with `Maj(Hold1, Hold2, Hold3)` in a single step.
- 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.

The run in space-time



Decode Majority of three copies.

Copy From neighbor colonies.

Compute Apply a step of g .

Encode Store 3 copies in Hold _{i}

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

Finish $\text{Info} \leftarrow \text{Maj}_{i=1}^3 \text{Hold}_i$

Put also $2\rho_1$ steps of **idling**
(doing nothing) between all
these stages.

- If the value of the fields

$$\text{Addr} \in \{0, \dots, Q - 1\}, \quad \text{Age} \in \{0, \dots, U - 1\}.$$

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 **program** is just a convenient description of a transition table. It is best described by a series of **rules** like this below, where Mail^{-1} denotes the Mail field of the left neighbor.

Rule 8.1: Example rule

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

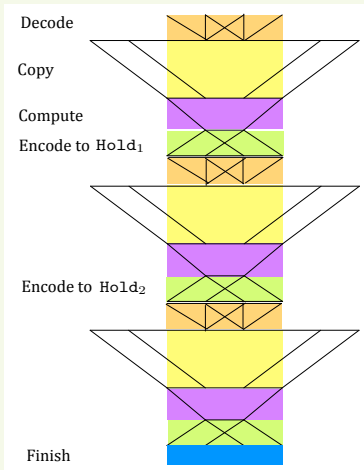
Assume $\rho_2 > U, 3Q$, so that at most one burst of faults (of size $2\rho_1$) can affect one colony work period.

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 Hold_i . Corrected in Finish.
- Other fields of the cells where the burst occurred. Corrected in Decode.



Correcting Addr and Age

Fortunately, this can be done **locally**. Anybody in the audience could come up with some rules to do it; I give an example solution.

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

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

A **domain** is a maximal interval of consistent cells.

- **Kill** cells whose domain is smaller than $2\rho_1$.
- If a dead cell has one live neighbor or two consistent live neighbors, **revive** it consistently with them.

We will implement this.

Call the **left depth** $\text{depth}_{-1}(x)$ of a cell x the distance to the first cell on the left that is inconsistent with it if this distance is $\leq \text{MaxDepth} = 2\rho_1$; otherwise, MaxDepth . **Right depth** is defined accordingly. So, $\text{depth}_{-1}(x) > 1$ means x is consistent with its left neighbor.

The fields $\text{Depth}_{-1}, \text{Depth}_1$ try to keep track of the depth.
pfor is parallel **for**.

Rule 8.2: *Purge*

```

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

Reviving the dead cells:

Rule 8.3: *Heal*

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

if $\text{Live} = 0$ **and** $\text{Live}^j = 1$ **and** $\text{Depth}_j^j = \text{MaxDepth}$

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

$\text{Live} \leftarrow 1$

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

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

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 $6\rho_1$ 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 Theorem is now proved easily. Having the Structure Restoration lemma allows the application of the argument of the Nice Faults lemma, even though the faults are not nice, (just 1-sparse).

This is the third one of a series of three lectures on reliable cellular automata.

The first lecture constructed a reliably computing 3-dimensional cellular automaton, and introduced a technique of proof that will be used now. Please recall the definitions of k -noise, k -sparse set of faults in space-time and the corresponding notions of damage, and k -sparse damage in space.

The second lecture constructed a 1-dimensional cellular automaton $M_1(\mathbb{S}_1, g_1)$ that computes reliably in the presence of 1-sparse noise. More exactly, the noise was allowed to be (ρ_1, ρ_2) -sparse: bursts of size $\leq \rho_1$, at a distance at least ρ_2 from each other.

We built it as

$$M_1(\mathbb{S}_1, g_1) = M_*(\mathbb{S}_*, g_*) \text{ simulating arbitrary } M_2(\mathbb{S}_2, g_2) = M(\mathbb{S}, g).$$

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

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

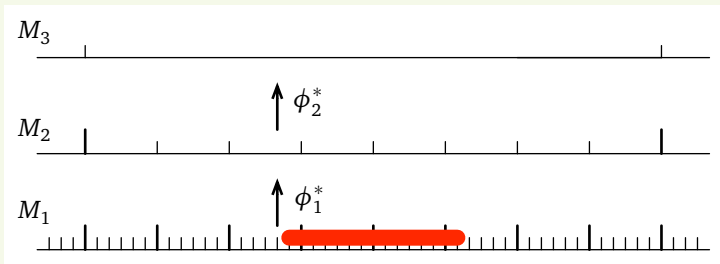
Upgrading to 2-sparse

- Machine M_1 resists a (ρ_1, ρ_2) -sparse set of faults: bursts of size $2\rho_1$ that were at a distance greater than ρ_2 from each other. We also called this a **1-sparse** set.
- Upgrade**: now we want to resist a 2-sparse set. So, in addition, we may have bursts of size $2\rho_2$, at a distance greater than ρ_3 from each other.
- Idea**: Let M_2 be itself a simulation of some machine M_3 (via a code ϕ_2), where M_2 resists a (ρ_2, ρ_3) -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 keep growing** as we add more levels. We will return to this problem later.

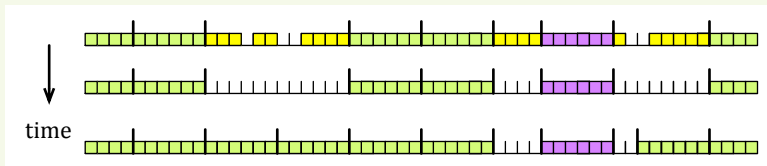


We hope that M_3 can deal with 2-sparse violations (red area above) of 1-sparsity, since the cells of M_1 simulating it (via $\phi_2^* \phi_1^*$) are stretching over an area of size $\gg \rho_2$

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.

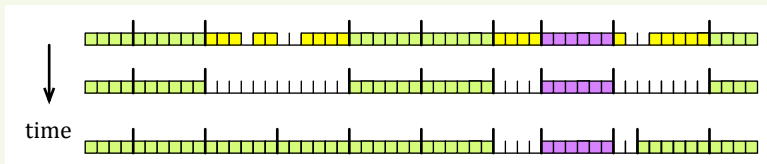
- Recall: in the simulation of last time, we called **nice faults** the ones that do not change Addr and Age. We showed that even if the faults are not nice, the Addr and Age values can still be corrected by the pair of rules *Purge*, *Heal*.
- Now the structure destruction can be more serious: faults can wipe out 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).
- Since the M_2 cells in question do not exist, we must solve this problem still in the automaton M_1 .

Two more rules.



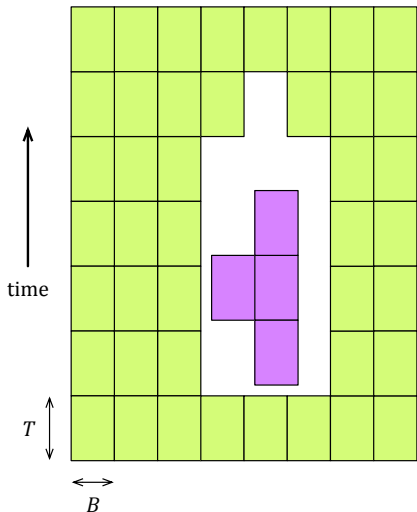
- Rule *Decay* kills a cell that has been inconsistent too long with a neighbor within its own colony. To wipe out unhealable partial colonies (yellow cells).
- Rule *Grow* lets a colony extend an arm of consistent cells into a nearby dead area. If a new colony could not be created, the arm decays again.

Unfortunately, the faults can also create **bad colonies** (for example, the purple colony above is misplaced). How to get rid of these?



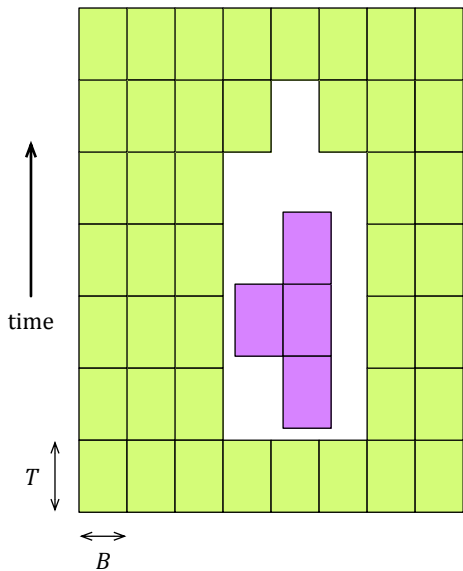
The bad colony can only be eliminated **by itself**. We said it simulates a cell of M_2 . But the presence of these bad colonies shows the need to **generalize** cellular automata: for example, the cell of M_2 simulated by the above bad colony has no adjacent neighbors.

Generalized cellular automata



We generalize M_2 to be able to answer the question: **what sort of object does M_1 simulate?**

Space-time configuration: 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 before switching.



Questions

- Is there a transition function for the case of neighbor (that is closer than B) cells that are non-adjacent?
- If not, what is a trajectory?
- Do non-adjacent neighbor cells communicate?

For a moment, assume these questions are answered.

We rely on the following to delete 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 g_2 , which has 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 must take care of this.

Forced simulation

Our automaton M_1 had 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 .

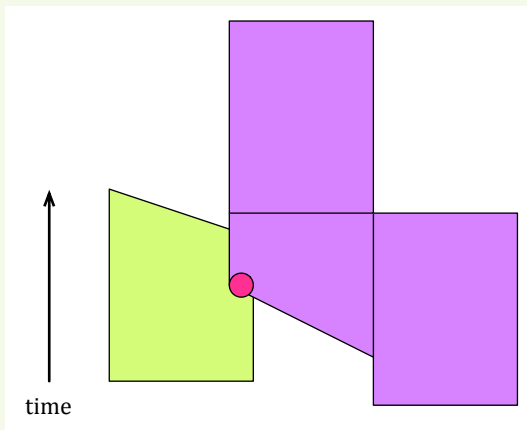
- A typical universal cellular automaton A_1 simulating some other cellular automaton A_2 would rely on the **program** of A_2 , written into each colony of A_1 . The the simulation performed machine M_1 is, on the other hand, **hard-wired**.
- It is not trivial to give the forced simaton property also to the simulated M_2 , but can be done, for example, by techniques related to the Kleene fixpoint theorem (recursion theorem).

With more rules killing and reviving cells:

Purge, Heal, Decay, Grow,

it becomes harder to make sure that they do not conflict with each other. Most of these potential conflicts are solved by making *Decay, Grow* much slower than *Purge, Heal*.

But here is a delicate situation that would allow a single burst to affect events on the colony level:

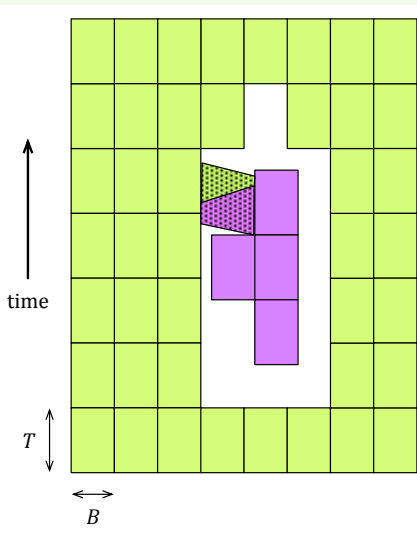


A bad colony's growth, as it completes the creation of a neighbor, **bites** into a good colony (due to a burst).

Two possible solutions.

- If there is communication among non-adjacent neighbors, *Heal* is made **stronger** than *Grow*: its creation kills neighboring growth cells that are in the way, if needed.
- 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.

Communication



Communication is not difficult to implement (in principle): colonies extend communicating arms for a long enough time to meet, provided that their work periods intersect substantially. In borderline cases, however, there is some unavoidable **nondeterminism** about which work-period of your neighbor colony you will communicate with. See the definition of trajectories below.

We make the cell body size B and the cell dwell period length T part of the definition of a generalized cellular automaton

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

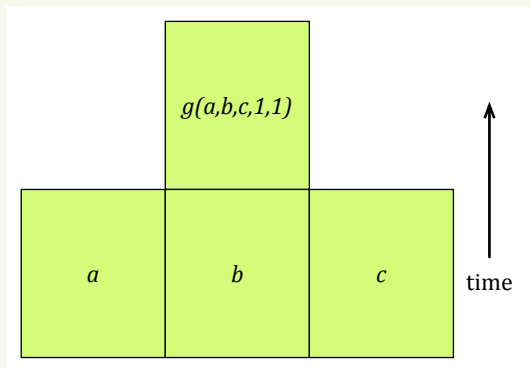
In the transition function $g(a, b, c, L, R)$, the bit L says whether the left neighbor is adjacent and aligned; R says the same about the right neighbor.

We denote by V the space-time $\mathcal{C} \times [0, \text{infty})$ by V . A **space-time configuration** consists of

- A set $\mathcal{R} \subseteq V$ of space-time points $\{(x_i, t_i)\}_{i=1}^{\infty}$ such that the rectangles $[x_i, x_i + B) \times [t_i, t_i + T)$ are disjoint.
- A mapping $\eta : \mathcal{R} \rightarrow \mathbb{S}$ assigning a state to each rectangle.
- A space-time set D of points called **exceptions**.

We will just denote it by (η, E) . The set of space-time configurations and space configurations of machine M is denoted by $\text{Evol}(M)$ and $\text{Config}(M)$ respectively.

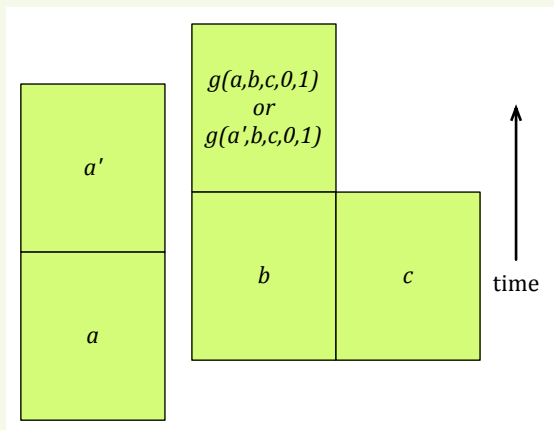
We now have to say when a space-time configuration is a **trajectory** of $\text{CA}(\mathbb{S}, g, B, T)$. In all conditions below, assume there is no exception point near (x, t) : $[x - 5B, x + 5B) \times (t - 5T, t] \cap D = \emptyset$.



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:

- and Suppose $(x, t), (x + B, t), (x, t + T) \in \mathcal{R}$, but instead of a left adjacent-aligned neighbor there is a $B \leq B' < 2B$ and a $t - T' < t' \leq t$ with $(x - B', t'), (x - B', t' + T) \in \mathcal{R} \setminus E$. Then the output value is one of the two based on the two possible left inputs, we do not specify which:

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

- Suppose $(x, t), (x + B, t), (x, t + T) \in \mathcal{R}$, and there is no left adjacent-aligned neighbor (but the other condition might not hold). Then still $\eta(x, t + T) = g(r, \eta(x, t), \eta(x + B, t), 0, 1)$ for some $r \in \mathbb{S}$.

Definition (Simulation)

Let M_1, M_2 be two generalized cellular automata. Let

$$\Phi : \text{Evol}(M_1) \rightarrow \text{Evol}(M_2), \quad \phi_* : \text{Config}(M_2) \rightarrow \text{Config}(M_1)$$

be mappings from the set of all space-time configurations of M_1 to those of M_2 , and from the set of space-configurations of M_2 to those of M_1 . We call it a **simulation** of machine M_2 by machine M_1 if for every $\xi \in \text{Config}(M_2)$, for every trajectory of M_1 starting from $\phi_*(\xi)$, the space-time configuration $\Phi(\eta)$ is a trajectory of M_2 .

In our block simulations 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 :

$$\eta_2(\cdot, t) = \phi^*(\eta_1(\cdot, tU)).$$

- Our goal is to define a sequence of generalized cellular automata and simulations:

$$M_1 \xrightarrow{\Phi_1} M_2 \xrightarrow{\Phi_2} M_3 \xrightarrow{\Phi_3} \dots$$

This object will be called an **amplifier**. If (η_1, D_1) is an initial space-time configuration then denote $(\eta_{k+1}, D_k) = \Phi_k(\eta_k, D_k)$.

- Recall the definition of k -noise. We fix a sequence of scales $\rho_1 < \rho_2 < \dots$: the details are not important now. If E is a space-time set then we denoted by $E^{(k)}$ its k -noise. By definition we have the identity

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

- The cellular automaton M_1 will be an ordinary, non-generalized one. η_1 will be a trajectory of it, with set of faults D_1 .
- We define the amplifier's action on the exception set D independently of η : if $(\eta^*, D^*) = \Phi_k(\eta, D)$ then $D^* = D^{(>k)}$, that is the part of D after deleting its k -noise. Then

$$D_k = D_1^{(>k-1)},$$

that is the exception set on level k is what remains from the faults after deleting the $(k - 1)$ -noise.

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. We will say that the amplifier has the **error-correction property** if the following holds.

Let $\eta_{k+1}(x, t)$ be live, and let $0 \leq a \leq Q_k$ and $0 \leq u < U_k$. Assume

$$\begin{aligned} [x - 5B_{k+1}, x + 5B_{k+1}) \times (t - 5T_{k+1}, t] \cap D_{k+1} &= \emptyset, \\ [x + (a - 5)B_k, x + (a + 5)B_k) \times (t + (u - 5)T_k, t] \cap D_k &= \emptyset. \end{aligned}$$

Then $\eta_k(x + aB_k, t + uT_k)$. Info = $\phi_{*k}(\eta_{k+1}(x, t))$. Info.

Here is a very informal statement of the main element of the proof.

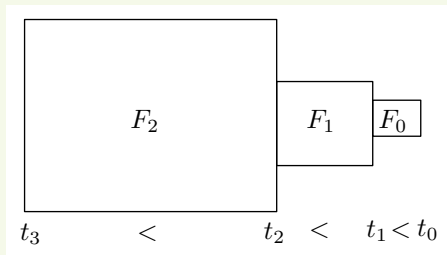
Lemma (Main)

For a wide range of choices of the parameters and the original cellular automaton ζ to be simulated, there is an amplifier with the error-correcting property.

The amplifier is essentially given by just the program that we wrote for the simulation g_1 of M_2 by M_1 , along with the code ϕ_1 . Except that now we must pretend that there can be neighbors that are not aligned or adjacent, and make choices for them.

- This lemma plays a role analogous to the Shrinking Triangle lemma in the case of the proof for the Toom Rule. But its proof is much harder.
The reason is that just as there, we must show that some large “mess” disappears. But this mess will not squeeze itself from outside, it must disappear by itself.
- The first, big task is to show that within a time, say, $3T_{k+1}$, only we will find only complete colonies. The main tool for this is the *Decay* rule, which kills the partial colonies; still, the details are tedious.

The final step of the proof depends on the exact theorem to be proven. For simplicity, assume that we only want to remember a bit of information, and so our code is the repetition code. Then we use the error-correction property for a reasoning similar to the one in Toom's Rule:



- The initial configuration has an infinite hierarchical structure. It takes a little argument to see that there is no problem in constructing it: indeed, it is not immediately obvious how to define the initial configuration given by

$$M_1 \xleftarrow{\phi_{1*}} M_2 \xleftarrow{\phi_{2*}} M_3 \xleftarrow{\phi_{3*}} \dots$$

- It is also possible to make an amplifier **self-organizing**. I cannot give the details now, but (for example in case the computation has only a constant-size input) it allows starting from a homogenous initial configuration.