2001-04-01

Reliable cellular automata with self-organization

RELIABLE CELLULAR AUTOMATA WITH SELF-ORGANIZATION

PETER GÁCS

Abstract. In a probabilistic cellular automaton in which all local transitions have positive probability, the problem of keeping a bit of information indefinitely is nontrivial, even in an infinite automaton. Still, there is a solution in 2 dimensions, and this solution can be used to construct a simple 3-dimensional discrete-time universal fault-tolerant cellular automaton. This technique does not help much to solve the following problems: remembering a bit of information in 1 dimension; computing in dimensions lower than 3; computing in any dimension with non-synchronized transitions.

Our more complex technique organizes the cells in blocks that perform a reliable simulation of a second (generalized) cellular automaton. The cells of the latter automaton are also organized in blocks, simulating even more reliably a third automaton, etc. Since all this (a possibly infinite hierarchy) is organized in "software", it must be under repair all the time from damage caused by errors. A large part of the problem is essentially self-stabilization recovering from a mess of arbitrary size and content. The present paper constructs an asynchronous one-dimensional fault-tolerant cellular automaton, with the further feature of "self-organization". The latter means that unless a large amount of input information must be given, the initial configuration can be chosen homogeneous.

Date: February 1, 2008.

1991 *Mathematics Subject Classification.* 60K35, 68Q80, 82C22, 37B15.

Key words and phrases. probabilistic cellular automata, interacting particle systems, renormalization, ergodicity, reliability, fault-tolerance, error-correction, simulation, hierarchy, self-organization.

Partially supported by NSF grant CCR-920484.

Contents

1. Introduction

A *cellular automaton* is a homogenous array of identical, locally communicating finite-state automata. The model is also called *interacting particle system*. Fault-tolerant computation and information storage in cellular automata is a natural and challenging mathematical problem but there are also some arguments indicating an eventual practical significance of the subject since there are advantages in uniform structure for parallel computers.

Fault-tolerant cellular automata (FCA) belong to the larger category of reliable computing devices built from unreliable components, in which the error probability of the individual components is not required to decrease as the size of the device increases. In such a model it is essential that the faults are assumed to be transient: they change the local state but not the local transition function.

A fault-tolerant computer of this kind must use massive parallelism. Indeed, information stored anywhere during computation is subject to decay and therefore must be actively maintained. It does not help to run two computers simultaneously, comparing their results periodically since faults will occur in both of them between comparisons with high probability. The self-correction mechanism must be built into each part of the computer. In cellular automata, it must be a property of the transition function of the cells.

Due to the homogeneity of cellular automata, since large groups of errors can destroy large parts of any kind of structure, "self-stabilization" techniques are needed in conjunction with traditional error-correction.

1.1. Historical remarks.

The problem of reliable computation with unreliable components was addressed in [29] in the context of Boolean circuits. Von Neumann's solution, as well as its improved versions in [9] and [23], rely on high connectivity and non-uniform constructs. The best currently known result of this type is in [25] where redundancy has been substantially decreased for the case of computations whose computing time is larger than the storage requirement.

Of particular interest to us are those probabilistic cellular automata in which all local transition probabilities are positive (let us call such automata *noisy*), since such an automaton is obtained by way of "perturbation" from a deterministic cellular automaton. The automaton may have e.g. two distinguished initial configurations: say ξ_0 in which all cells have state 0 and ξ_1 in which all have state 1 (there may be other states besides 0 and 1). Let $p_i(x,t)$ be the probability that, starting from initial configuration ξ_i, the state of cell x at time t is i. If $p_i(x,t)$ is bigger than, say, $2/3$ for all x,t then we can say that the automaton remembers the initial configuration forever.

Informally speaking, a probabilistic cellular automaton is called *mixing* if it eventually forgets all information about its initial configuration. Finite noisy cellular automata are always mixing. In the example above, one can define the "relaxation time" as the time by which the probability decreases below $2/3$. If an infinite automaton is mixing then the relaxation time of the corresponding finite automaton is bounded independently of size. A minimal requirement of fault-tolerance is therefore that the infinite automaton be non-mixing.

The difficulty in constructing non-mixing noisy one-dimensional cellular automata is that eventually large blocks of errors which we might call islands will randomly occur. We can try to design a transition function that (except for a small error probability) attempts to decrease these islands. It is a natural idea that the function should replace the state of each cell, at each transition time, with the majority of the cell states in some neighborhood. However, majority voting among the five nearest neighbors (including the cell itself) seems to lead to a mixing transition function, even in two dimensions, if the "failure" probabilities are not symmetric with respect to the interchange of 0's and 1's, and has not been proved to be non-mixing even in the symmetric case. Perturbations of the one-dimensional majority voting function were actually shown to be mixing in [16] and [17].

Non-mixing noisy cellular automata for dimensions 2 and higher were constructed in [27]. These automata are also *non-ergodic*: an apparently stronger property (see formal definition later). All our examples of non-mixing automata will also be non-ergodic. The paper [14] applies Toom's work [27] to design a simple three-dimensional fault-tolerant cellular automaton that simulates arbitrary one-dimensional arrays. The original proof was simplified and adapted to strengthen these results in [5].

Remark 1.1. A three-dimensional fault-tolerant cellular automaton cannot be built to arbitrary size in the physical space. Indeed, there will be an (inherently irreversible) error-correcting operation on the average in every constant number of steps in each cell. This will produce a steady flow of heat from each cell that needs therefore a separate escape route for each cell. ♢

A simple one-dimensional deterministic cellular automaton eliminating finite islands in the absence of failures was defined in [13] (see also [8]). It is now known (see [22]) that perturbation (at least, in a strongly biased way) makes this automaton mixing.

1.2. **Hierarchical constructions.** The limited geometrical possibilities in one dimension suggest that only some non-local organization can cope with the task of eliminating finite islands. Indeed, imagine a large island of 1's in the 1-dimensional ocean of 0's. Without additional information, cells at the left end of this island will not be able to decide locally whether to move the boundary to the right or to the left. This information must come from some global organization that, given the fixed size of the cells, is expected to be hierarchical. The "cellular automaton" in [28] gives such a hierarchical organization. It indeed can hold a bit of information indefinitely. However, the transition function is not uniform either in space or time: the hierarchy is "hardwired" into the way the transition function changes.

The paper [10] constructs a non-ergodic one-dimensional cellular automaton working in discrete time, using some ideas from the very informal paper [19] of Georgii Kurdyumov. Surprisingly, it seems even today that in one dimension, the keeping of a bit of information requires all the organization needed for general fault-tolerant computation. The paper [11] constructs a two-dimensional fault-tolerant cellular automaton. In the two-dimensional work, the space requirement of the reliable implementation of a computation is only a constant times greater than that of the original version. (The time requirement increases by a logarithmic factor.)

In both papers, the cells are organized in blocks that perform a fault-tolerant simulation of a second, generalized cellular automaton. The cells of the latter automaton are also organized in blocks, simulating even more reliably a third generalized automaton, etc. In all these papers (including the present one), since all this organization is in "software", i.e. it is encoded into the states of the cells, it must be under repair all the time from breakdown caused by errors. In the two-dimensional case, Toom's transition function simplifies the repairs.

1.2.1. *Asynchrony.* In the three-dimensional fault-tolerant cellular automaton of [14], the components must work in discrete time and switch simultaneously to their next state. This requirement is unrealistic for arbitrarily large arrays. A more natural model for asynchronous probabilistic cellular automata is that of a continuous-time Markov process. This is a much stronger assumption than allowing an adversary scheduler but it still leaves a lot of technical problems to be solved. Informally it allows cells to choose whether to update at the present time independently of the choice their neighbors make.

The paper [5] gives a simple method to implement arbitrary computations on asynchronous machines with otherwise perfectly reliable components. A two-dimensional asynchronous fault-tolerant cellular automaton was constructed in [30]. Experiments combining this technique with the error-correction mechanism of [14] were made, among others, in [2].

The present paper constructs a one-dimensional asynchronous fault-tolerant cellular automaton, thus completing the refutation of the so-called Positive Rates Conjecture in [20].

1.2.2. *Self-organization.* Most hierarchical constructions, including ours, start from a complex, hierarchical initial configuration (in case of an infinite system, and infinite hierarchy). The present paper presents some results which avoid this. E.g., when the computation's goal is to remember a constant amount of information, (as in the refutation of the positive rates conjecture) then we will give a transition function that performs this task even if each cell of the initial configuration has the same state. We call this "self-organization" since the hierarchical organization will still emerge during the computation.

1.2.3. *Proof method simplification.* Several methods have emerged that help managing the complexity of a large construction but the following two are the most important.

- A number of "interface" concepts is introduced (generalized simulation, generalized cellular automaton) helping to separate the levels of the infinite hierarchy, and making it possible to speak meaningfully of a single pair of adjacent levels.
- Though the construction is large, its problems are presented one at a time. E.g. the messiest part of the self-stabilization is the so-called Attribution Lemma, showing how after a while all cells can be attributed to some large organized group (colony), and thus no debris is in the way of the creation of new colonies. This lemma relies mainly on the Purge and Decay rules, and will be proved before introducing many other major rules. Other parts of the construction that are not possible to ignore are used only through "interface conditions" (specifications).

We believe that the new result and the new method of presentation will serve as a firm basis for other new results. An example of a problem likely to yield to the new framework is the *growth rate of the relaxation time* as a function of the size of a finite cellular automaton. At present, the relaxation time of all known cellular automata either seems to be bounded (ergodic case) or grows exponentially. We believe that our constructions will yield examples for other, intermediate growth rates.

1.2.4. *Overview of the paper.*

- Sections 2, 3, 4 are an informal discussion of the main ideas of the construction, along with some formal definitions, e.g.
  - block codes, colonies, hierarchical codes;
  - *abstract media*, which are a generalization of cellular automata;
  - *simulations*;
  - *amplifiers*: a sequence of media with simulations between each and the next one;
- Section 5 formulates the main theorems for discrete time.

  It also explains the main technical problems of the construction and the ways to solve them:
  - correction of structural damage by destruction followed by rebuilding from the neighbors;
  - a "hard-wired" program;
  - "legalization" of all locally consistent structures;
- Section 6 defines *media*, a specialization of abstract media with the needed stochastic structure. Along with media, we will define *canonical simulations*, whose form guarantees that they are simulations between media. We will give the basic examples of media with the basic simulations between them.

  The section also defines variable-period media and formulates the main theorems for continuous time.

- Section 7 develops some simple simulations, to be used either directly or as a paradigm. The example transition function defined here will correct any set of errors in which no two errors occur close to each other.

  We also develop the language used for defining our transition function in the rest of the paper.
- A class of media for which nontrivial fault-tolerant simulations exist will be defined in Section 8. In these media, called "robust media", cells are not necessarily adjacent to each other. The transition function can erase as well as create cells.

  There is a set of "bad" states. The set of space-time points where bad values occur is called the "damage". The Restoration Property requires that at any point of a trajectory, damage occurs (or persists) only with small probability (ε). The Computation Property requires that the trajectory obey the transition function in the absence of damage.

  It is possible to tell in advance how the damage will be defined in a space-time configuration η^* of a medium M_2 simulated by some space-time configuration η of a medium M_1. Damage is said to occur at a certain point (x,t) of η^* if within a certain space-time rectangle in the past of (x,t), the damage of η cannot be covered by a small rectangle of a certain size. This is saying, essentially, that damage occurs at least "twice" in η. The Restoration Property for η with ε will then guarantee that the damage in η^* also satisfies a restoration property with with $\approx \varepsilon^2$.
- Section 9 introduces all notions for the formulation of the main lemma. First we define the kind of amplifiers to be built and a set of parameters called the *amplifier frame*. The main lemma, called the Amplifier Lemma, says that amplifiers exist for many different sorts of amplifier frame. The rest of the section applies the main lemma to the proof of the main theorems.
- Section 10 defines self-organizing amplifiers, formulates the lemma about their existence and applies it to the proof of the existence of a self-organizing non-ergodic cellular automaton.
- Section 11 gives an overview of an amplifier. As indicated above, the restoration property will be satisfied automatically. In order to satisfy the computation property, the general framework of the program will be similar to the outline in Section 7. However, besides the single-error fault-tolerance property achieved there, it will also have a *self-stabilization* property. This means that a short time after the occurrence of arbitrary damage, the configuration enables us to interpret it in terms of colonies. (In practice, pieces of incomplete colonies will eliminate themselves.) In the absence of damage, therefore, the colony structure will recover from the effects of earlier damage, i.e. predictability in the simulated configuration is restored.
- Section 12 gives the rules for killing, creation and purge. We prove the basic lemmas about space-time paths connecting live cells.
- Section 13 defines the decay rule and shows that a large gap will eat up a whole colony.
- Section 14 proves the Attribution Lemma that traces back each non-germ cell to a full colony. This lemma expresses the "self-stabilization" property mentioned above. The proof starts with Subsection 14.1 showing that if a gap will not be healed promptly then it grows.
- Section 15 proves the Healing Lemma, showing how the effect of a small amount of damage will be corrected. Due to the need to restore some local clock values consistently with the neighbors, the healing rule is rather elaborate.
- Section 16 introduces and uses the error-correcting computation rules not dependent on communication with neighbor colonies.
- Section 17 introduces and applies the communication rules needed to prove the Computation Property in simulation. These are rather elaborate, due to the need to communicate with not completely reliable neighbor colonies asynchronously.
- Section 18 defines the rules for germs and shows that these make our amplifier self-organizing.

The above constructions will be carried out for the case when the cells work asynchronously (with variable time between switchings). This does not introduce any insurmountable difficulty but makes life harder at several steps: more care is needed in the updating and correction of the counter field of a cell, and in the communication between neighbor colonies. The analysis in the proof also becomes more involved.

Acknowledgement. I am very thankful to Robert Solovay for reading parts of the paper and finding important errors. Larry Gray revealed his identity as a referee and gave generously of his time to detailed discussions: the paper became much more readable (yes!) as a result.

2. Cellular automata

In the introductory sections, we confine ourselves to one-dimensional infinite cellular automata.

Notation. Let $\mathbb{R}$ be the set of real numbers, and $\mathbb{Z}_m$ the set of remainders modulo m. For $m = \infty$, this is the set $\mathbb{Z}$ of integers. We introduce a non-standard notation for intervals on the real line. Closed intervals are denoted as before: $[a, b] = \{\, x : a \le x \le b \,\}$. But open and half-closed intervals are denoted as follows:

$$[a+, b] = \{\, x : a < x \le b \,\},$$
$$[a, b-] = \{\, x : a \le x < b \,\},$$
$$[a+, b-] = \{\, x : a < x < b \,\}.$$

The advantage of this notation is that the pair (x, y) will not be confused with the open interval traditionally denoted (x, y) and that the text editor program will not complain about unbalanced parentheses. We will use the same notation for intervals of integers: the context will make it clear, whether $[a, b]$ or $[a, b] \cap \mathbb{Z}$ is understood. Given a set A of space or space-time and a real number c, we write

$$cA = \{\, cv : v \in A \,\}.$$

If the reader wonders why lists of assertions are sometimes denoted by (a), (b), ... and sometimes by (1), (2), ..., here is the convention I have tried to keep to. If I list properties that all hold or are required (conjunction) then the items are labeled with (a),(b), ... while if the list is a list of several possible cases (disjunction) then the items are labeled with (1), (2), ...

Maxima and minima will sometimes be denoted by $\vee$ and $\wedge$. We will write log for $\log_2$.

2.1. Deterministic cellular automata.

Let us give here the most frequently used definition of cellular automata. Later, we will use a certain generalization. The set $\mathbf{C}$ of *sites* has the form $\mathbb{Z}_m$ for finite or infinite m. This will mean that in the finite case, we take *periodic boundary conditions*. In a space-time vector (x, t), we will always write the space coordinate first. For a space-time set E, we will denote its space- and time projections by

$$\pi_{\mathrm{s}} E, \pi_{\mathrm{t}} E \tag{2.1}$$

respectively. We will have a finite set $\mathbb{S}$ of states, the potential states of each site. A *(space) configuration* is a function

$$\xi(x)$$

for $x \in \mathbf{C}$. Here, $\xi(x)$ is the state of site x.

The time of work of our cellular automata will be the interval $[0, \infty-]$. Our *space-time* is given by

$$\mathbf{V} = \mathbf{C} \times [0, \infty-].$$

A *space-time configuration* is a space-time function $\eta(x,t)$ which for each t defines a space configuration. If in a space-time configuration η we have $\eta(x,v) = s_2$ and $\eta(x,t) = s_1 \neq s_2$ for all $t < v$ sufficiently close to v then we can say that there was a *switch* from state s_1 to state s_2 at time v. For ordinary discrete-time cellular automata, we allow only space-time configurations in which all switching times are natural numbers $0, 1, 2, \ldots$. The time 0 is considered a switching time. If there is an ε such that $\eta(c,t)$ is constant for $a - \varepsilon < t < a$ then this constant value will be denoted by

$$\eta(c, a-). \tag{2.2}$$

The *subconfiguration* $\xi(D')$ of a configuration ξ defined on $D \supset D'$ is the restriction of ξ to D'. Sometimes, we write

$$\eta(V)$$

for the sub-configuration over the space-time set V.

A *deterministic cellular automaton*

$$CA(Tr, \mathbf{C}).$$

is determined by a transition function $Tr : \mathbb{S}^3 \to \mathbb{S}$ and the set $\mathbf{C}$ of sites. We will omit $\mathbf{C}$ from the notation when it is obvious from the context. A space-time configuration η is a *trajectory* of this automaton if

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

holds for all x, t with $t > 0$. For a space-time configuration η let us write

$$\overline{Tr}(\eta, x, t) = Tr(\eta(x-1,t), \eta(x,t), \eta(x+1,t)). \tag{2.3}$$

Given a configuration ξ over the space $\mathbf{C}$ and a transition function, there is a unique trajectory η with the given transition function and the initial configuration $\eta(\cdot, 0) = \xi$.

2.2. **Fields of a local state.** The space-time configuration of a deterministic cellular automaton can be viewed as a "computation". Moreover, every imaginable computation can be performed by an appropriately chosen cellular automaton function. This is not the place to explain the meaning of this statement if it is not clear to the reader. But it becomes maybe clearer if we point out that a better known model of computation, the Turing machine, can be considered a special cellular automaton.

Let us deal, from now on, only with cellular automata in which the set $\mathbb{S}$ of local states consists of binary strings of some fixed length $\|\mathbb{S}\|$ called the *capacity* of the sites. Thus, if the automaton has 16 possible states then its states can be considered binary strings of length 4. If $\|\mathbb{S}\| > 1$ then the information represented by the state can be broken up naturally into parts. It will greatly help reasoning about a transition rule if it assigns different functions to some of these parts; a typical "computation" would indeed do so. Subsets of the set $\{0, \ldots, \|\mathbb{S}\| - 1\}$ will be called *fields*. Some of these subsets will have special names. Let

$$All = \{0, \ldots, \|\mathbb{S}\| - 1\}.$$

If $s = (s(i) : i \in All))$ is a bit string and $F = \{i_1, \ldots, i_k\}$ is a field with $i_j < i_{j+1}$ then we will write

$$s.F = (s(i_1), \ldots, s(i_k))$$

for the bit string that is called *field* F of the state.

Example 2.1. If the capacity is 12 we could subdivide the interval $[0, 11]$ into subintervals of lengths 2,2,1,1,2,4 respectively and call these fields the input, output, mail coming from left, mail coming from right, memory and workspace. We can denote these as *Input*, *Output*, $Mail_j$ $(j = -1, 1)$, *Work*

and *Memory*. If s is a state then $s.Input$ denotes the first two bits of s, $s.Mail_1$ means the sixth bit of s, etc. ♢

Remark 2.2. Treating these fields differently means we may impose some useful restrictions on the transition function. We might require the following, calling $Mail_{-1}$ the "right-directed mail field":

> The information in $Mail_{-1}$ moves always to the right. More precisely, in a trajectory η, the only part of the state $\eta(x,t)$ that depends on the state $\eta(x-B,t-T)$ of the left neighbor is the right-directed mail field $\eta(x,t).Mail_{-1}$. This field, on the other hand, depends only on the right-directed mail field of the left neighbor and the workspace field $\eta(x,t-T).Work$. The memory depends only on the workspace.

Confining ourselves to computations that are structured in a similar way make reasoning about them in the presence of faults much easier. Indeed, in such a scheme, the effects of a fault can propagate only through the mail fields and can affect the memory field only if the workspace field's state allows it. ♢

Fields are generally either disjoint or contained in each other. When we join e.g. the input fields of the different sites we can speak about the input *track*, like a track of some magnetic tape.

2.2.1. *Bandwidth and separability.* Here we are going to introduce some of the fields used later in the construction. It is possible to skip this part without loss of understanding and to refer to it later as necessary. However, it may show to computer scientists how the communication model of cellular automata can be made more realistic by introducing a new parameter w and some restrictions on transition functions. The restrictions do not limit information-processing capability but

- limit the amount of information exchanged in each transition;
- limit the amount of change made in a local state in each transition;
- restrict the fields that depend on the neighbors immediately;

A transition function with such structure can be simulated in such a way that only a small part of the memory is used for information processing, most is used for storage. For a fixed *bandwidth* w, we will use the following disjoint fields, all with size w:

$$Inbuf = [0, w-1], \qquad Outbuf, \qquad Pointer. \tag{2.4}$$

Let

$$Buf = Inbuf \cup Outbuf, \qquad Memory = All \setminus Inbuf.$$

Note that the fields denoted by these names are not disjoint. For a transition function $Tr : \mathbb{S}^3 \to \mathbb{S}$, for an integer $\lceil \log \|\mathbb{S}\| \rceil \le w \le \|\mathbb{S}\|$, we say that Tr is *separable with bandwidth* w if there is a function

$$Tr^{(w)} : \{0,1\}^{7w} \to \{0,1\}^{5w} \tag{2.5}$$

determining Tr in the following way. For states r_{-1}, r_0, r_1, let $a = r_0.Pointer$, (a binary number), then with

$$p = Tr^{(w)}(r_{-1}.Buf, r_0.(Buf \cup [a, a+w-1]), r_1.Buf), \tag{2.6}$$

we define

$$Tr(r_{-1}, r_0, r_1).(Buf \cup Pointer) = p.[0, 3w-1]. \tag{2.7}$$

The value $Tr(r_{-1}, r_0, r_1)$ can differ from r_0 only in $Buf \cup Pointer$ and in field $[n, n+w-1]$ where $n = p.[4w, 5w-1]$ (interpreted as an integer in binary notation) and then

$$Tr(r_{-1}, r_0, r_1).[n, n+w-1] = p.[3w, 4w-1].$$

It is required that only *Inbuf* depends on the neighbors directly:

$$Tr(r_{-1}, r_0, r_1).Memory = Tr(Vac, r_0, Vac).Memory, \tag{2.8}$$

where *Vac* is a certain distinguished state. Let

$$legal^{Tr}(u, v) = \begin{cases} 1 & \text{if } v.Memory = Tr(Vac, u, Vac).Memory, \\ 0 & \text{otherwise.} \end{cases}$$

Thus, sites whose transition function is separable with bandwidth w communicate with each other via their field *Buf* of size $2w$. The transition function also depends on $r_0.[a, a + w - 1]$ where $a = r_0.Pointer$. It updates $r_0.(Buf \cup Pointer)$ and $r_0.[n + w - 1]$ where n is also determined by $Tr^{(w)}$.

Remark 2.3. These definitions turn each site into a small "random access machine". ♢

When designing a cellular automaton with small bandwidth for theoretical purposes, the guiding principle is to put almost everything into *Buf*, so that we do not have to worry about communication with neighbors. All the fields having to do with computation and administration will be small. The only exception is the bulk of the information that the cell is supposed to store. But even here, the redundant part of the information used for error-correcting purposes will be small and can be in *Buf*.

2.3. **Probabilistic cellular automata.** A *random space-time configuration* is a pair (μ, η) where μ is a probability measure over some measurable space $(\Omega, \mathcal{A})$ together with a measurable function $\eta(x, t, \omega)$ which is a space-time configuration for all $\omega \in \Omega$. We will generally omit ω from the arguments of η. When we omit the mention of μ we will use Prob to denote it. If it does not lead to confusion, for some property of the form $\{\, \eta \in R \,\}$, the quantity $\mu\{\, \omega : \eta(\cdot, \cdot, \omega) \in R \,\}$ will be written as usual, as

$$\mu\{\, \eta \in R \,\}.$$

We will denote the expected value of f with respect to μ by

$$\mathbf{E}_\mu f$$

where we will omit μ when it is clear from the context. A function $f(\eta)$ with values 0, 1 (i.e. an indicator function) and measurable in $\mathcal{A}$ will be called an *event function* over $\mathcal{A}$. Let W be any subset of space-time that is the union of some rectangles. Then

$$\mathcal{A}(W)$$

denotes the σ-algebra generated by events of the form

$$\{\, \eta(x, t) = s \text{ for } t_1 \le t < t_2 \,\}$$

for $s \in \mathbb{S}$, $(x, t_i) \in W$. Let

$$\mathcal{A}_t = \mathcal{A}(\mathbf{C} \times [0, t]).$$

A *probabilistic cellular automaton*

$$PCA(\mathbf{P}, \mathbf{C})$$

is characterized by saying which random space-time configurations are considered *trajectories*. Now a trajectory is not a single space-time configuration (sample path) but a distribution over space-time configurations that satisfies the following condition. The condition depends on a *transition matrix* $\mathbf{P}(s, (r_{-1}, r_0, r_1))$. For an arbitrary space-time configuration η, and space-time point (x, t), let

$$\overline{\mathbf{P}}(\eta, s, x, t) = \mathbf{P}(s, (\eta(x - 1, t), \eta(x, t), \eta(x + 1, t))). \tag{2.9}$$

We will omit the parameter η when it is clear from the context. The condition says that the random space-time configuration η is a trajectory if and only if the following holds. Let $x_0, \dots, x_{n+1}$ be given with $x_{i+1} = x_i + 1$. Let us fix an arbitrary space-time configuration ζ and an arbitrary event $\mathcal{H} \ni \zeta$ of positive probability in $\mathcal{A}_{t-T}$. Then we require

$$\mathrm{Prob}\left\{ \bigcap_{i=1}^{n} \{\eta(x_i,t) = \zeta(x_i,t)\} \;\middle|\; \mathcal{H} \cap \bigcap_{i=0}^{n+1} \{\eta(x_i,t-1) = \zeta(x_i,t-1)\} \right\} = \prod_{i=1}^{n} \overline{\mathbf{P}}(\eta, \zeta(x_i,t), x_i, t-1).$$

A probabilistic cellular automaton is *noisy* if $\mathbf{P}(s,\mathbf{r}) > 0$ for all $s, \mathbf{r}$. Bandwidth can be defined for transition probabilities just as for transition functions.

Example 2.4. As a simple example, consider a deterministic cellular automaton with a "random number generator". Let the local state be a record with two fields, *Det* and *Rand* where *Rand* consists of a single bit. In a trajectory (μ, η), the field $\eta.Det(x, t+1)$ is computed by a deterministic transition function from $\eta(x-1,t)$, $\eta(x,t)$, $\eta(x+1,t)$, while $\eta.Rand(x,t+1)$ is obtained by "coin-tossing". ♢

A trajectory of a probabilistic cellular automaton is a discrete-time Markov process. If the set of sites consists of a single site then $\mathbf{P}(s,\mathbf{r})$ is the transition probability matrix of this so-called finite Markov chain. The Markov chain is finite as long as the number of sites is finite.

2.4. **Continuous-time probabilistic cellular automata.** For later reference, let us define here (1-dimensional) probabilistic cellular automata in which the sites make a random decision "in each moment" on whether to make a transition to another state or not. These will be called *continuous-time interacting particle systems*. A systematic theory of such systems and an overview of many results available in 1985 can be found in [20]. Here, we show two elementary constructions, the second one of which is similar to the one in [16]. The system is defined by a matrix $\mathbf{R}(s,\mathbf{r}) \geq 0$ of transition rates in which all "diagonal" elements $\mathbf{R}(r_0, (r_{-1}, r_0, r_1))$ are 0.

2.4.1. *Discrete-time approximation.* Consider a generalization $PCA(\mathbf{P}, B, \delta, \mathbf{C})$ of probabilistic cellular automata in which the sites are at positions iB for some fixed B called the *body size* and integers i, and the switching times are at $0, \delta, 2\delta, 3\delta, \dots$ for some small positive δ. Let $M_\delta = PCA(\mathbf{P}, 1, \delta)$ with $\mathbf{P}(s,\mathbf{r}) = \delta\mathbf{R}(s,\mathbf{r})$ when $s \neq r_0$ and $1 - \delta\sum_{s' \neq r_0} \mathbf{R}(s',\mathbf{r})$ otherwise. (This definition is sound when δ is small enough to make the last expression nonnegative.) With any fixed initial configuration $\eta(\cdot, 0)$, the trajectories η_δ of M_δ will converge (weakly) to a certain random process η which is the continuous-time probabilistic cellular automaton with these rates, and which we will denote

$$CCA(\mathbf{R}, \mathbf{C}).$$

The process defined this way is a Markov process, i.e. if we fix the past before some time t_0 then the conditional distribution of the process after t_0 will only depend on the configuration at time t_0. For a proof of the fact that η_δ converges weakly to η, see [20], [16] and the works quoted there. For a more general definition allowing simultaneous change in a finite number of sites, see [20]. A continuous-time interacting particle system is *noisy* if $\mathbf{R}(s,\mathbf{r}) > 0$ for all $s \neq r_0$.

2.5. **Perturbation.**

2.5.1. *Discrete time.* Intuitively, a deterministic cellular automaton is fault-tolerant if even after it is "perturbed" into a probabilistic cellular automaton, its trajectories can keep the most important properties of a trajectory of the original deterministic cellular automaton. We will say that a random space-time configuration (μ, η) is a *trajectory* of the *ε-perturbation*

$$CA_\varepsilon(Tr, \mathbf{C})$$

of the transition function Tr if the following holds. For all $x_0, \dots x_{n+1}, t$ with $x_{i+1} = x_i + 1$ and events $\mathcal{H}$ in $\mathcal{A}_{t-1}$ with $\mu(\mathcal{H}) > 0$, for all $0 < i_1 < \cdots < i_k < n$,

$$\mu\left\{ \bigcap_{j=1}^{k} \{\eta(x_{i_j}, t) \neq \overline{Tr}(\eta, x_{i_j}, t-1)\} \;\middle|\; \mathcal{H} \cap \bigcap_{i=0}^{n+1} \{\eta(x_i, t-1) = s_i\} \right\} \leq \varepsilon^k.$$

Note that $CA_\varepsilon(Tr, \mathbf{C})$ is not a probabilistic cellular automaton. If we have any probabilistic cellular automaton $PCA(\mathbf{P}, \mathbf{C})$ such that $\mathbf{P}(s, \mathbf{r}, \mathbf{C}) \leq 1 - \varepsilon$ whenever $s = Tr(\mathbf{r})$ then the trajectories of this are trajectories of $CA_\varepsilon(Tr, \mathbf{C})$; however, these do not exhaust the possibilities. We may think of the trajectory of a perturbation as a process created by an "adversary" who is trying to defeat whatever conclusions we want to make about the trajectory, and is only restricted by the inequalities that the distribution of the trajectory must satisfy.

2.5.2. *Continuous time.* By the *ε-perturbation* of a continuous-time interacting particle system with transition rates given by $\mathbf{R}(s, \mathbf{r})$, we understand the following: in the above construction of a process, perturb the matrix elements $\mathbf{R}(s, \mathbf{r})$ by some arbitrary amounts smaller than ε. Note that this is a more modest kind of perturbation since we perturb the parameters of the process once for all and the perturbed process is again a continuous-time interacting particle system.

2.5.3. *Remembering a few bits.* Suppose that the bit string that is a local state has some field F (it can e.g. be the first two bits of the state). We will say that Tr *remembers* field F if there is an $\varepsilon > 0$ such that for each string $s \in \{0,1\}^{|F|}$ there is a configuration ξ_s such that for an infinite $\mathbf{C}$, for all trajectories (μ, η) of the ε-perturbation $CA_\varepsilon(Tr, \mathbf{C})$ with $\eta(\cdot, 0) = \xi_s$, for all x, t we have

$$\mu\{\, \eta(x,t).F = s \,\} > 2/3.$$

We define similarly the notions of remembering a field for a probabilistic transition matrix $\mathbf{P}$ and a probabilistic transition rate matrix $\mathbf{R}$.

One of the main theorems in [10] says:

Theorem 2.5 (1-dim, non-ergodicity, discrete time)**.**
There is a one-dimensional transition function that remembers a field.

One of the new results is the following

Theorem 2.6 (1-dim, non-ergodicity, continuous time)**.**
There is a one-dimensional transition-rate matrix that remembers a field.

3. Codes

3.1. **Colonies.** For the moment, let us concentrate on the task of remembering a single bit in a field called *Main_bit* of a cellular automaton. We mentioned in Subsection 1.2 that in one dimension, even this simple task will require the construction and maintenance of some non-local organization, since this is the only way a large island can be eliminated.

This organization will be based on the concept of colonies. Let x be a site and Q a positive integer. The set of Q sites $x+i$ for $i \in [0, Q-1]$ will be called the *Q-colony* with *base* x, and site $x+i$ will be said to have *address* i in this colony. Let us be given a configuration ξ of a cellular automaton M with state set $\mathbb{S}$. The fact that ξ is "organized into colonies" will mean that one can break up the set of all sites into non-overlapping colonies of size Q, using the information in the configuration ξ in a translation-invariant way. This will be achieved with the help of an *address field* $Addr$ which we will always have when we speak about colonies. The value $\xi(x).Addr$ is a binary string which can be interpreted as an integer in $[0, Q-1]$. Generally, we will assume that the $Addr$ field is large enough (its size is at least $\log Q$). Then we could say that a certain Q-colony $\mathcal{C}$ is a "real" colony of ξ if for each element y of $\mathcal{C}$ with address i we have $\xi(y).Addr = i$. In order to allow, temporarily, smaller address fields, let us actually just say that a Q-colony with base x is a "real" colony of the configuration ξ if its base is its only element having $Addr = 0$.

Cellular automata working with colonies will not change the value of the address field unless it seems to require correction. In the absence of faults, if such a cellular automaton is started with a configuration grouped into colonies then the sites can always use the $Addr$ field to identify their colleagues within their colony.

Grouping into colonies seems to help preserve the *Main_bit* field since each colony has this information in Q-fold redundancy. The transition function may somehow involve the colony members in a coordinated periodic activity, repeated after a period of U steps for some integer U, of restoring this information from the degradation caused by faults (e.g. with the help of some majority operation).

Let us call U steps of work of a colony a *work period*. The best we can expect from a transition function of the kind described above is that unless too many faults happen during some colony work period the *Main_bit* field of most sites in the colony will always be the original one. Rather simple such transition functions can indeed be written. But they do not accomplish qualitatively much more than a local majority vote for the *Main_bit* field among three neighbors.

Suppose that a group of failures changes the original content of the *Main_bit* field in some colony, in so many sites that internal correction is no more possible. The information is not entirely lost since most probably, neighbor colonies still have it. But correcting the information in a whole colony with the help of other colonies requires organization reaching wider than a single colony. To arrange this broader activity also in the form of a cellular automaton we use the notion of simulation with error-correction.

Let us denote by M_1 the fault-tolerant cellular automaton to be built. In this automaton, a colony $\mathcal{C}$ with base x will be involved in two kinds of activity during each of its work periods.

Simulation: Manipulating the collective information of the colony in a way that can be interpreted as the simulation of a single state transition of site x of some cellular automaton M_2.

Error-correction: Using the collective information (the state of x in M_2) to correct each site within the colony as necessary.

Of course, even the sites of the simulated automaton M_2 will not be immune to errors. They must also be grouped into colonies simulating an automaton M_3, etc.; the organization must be a *hierarchy of simulations*.

Reliable computation itself can be considered a kind of simulation of a deterministic cellular automaton by a probabilistic one.

3.2. **Block codes.**

3.2.1. *Codes on strings.* The notion of simulation relies on the notion of a *code*, since the way the simulation works is that the simulated space-time configuration can be decoded from the simulating space-time configuration. A code, φ between two sets R, S is, in general, a pair (φ^*, φ_*) where $\varphi_* : R \to S$ is the *encoding function* and $\varphi^* : S \to R$ is the *decoding function* and the relation

$$\varphi^*(\varphi_*(r)) = r$$

holds. A simple example would be when $R = \{0, 1\}$, $S = R^3$, $\varphi_*(r) = (r, r, r)$ while $\varphi^*((r, s, t))$ is the majority of r, s, t.

This example can be generalized to a case when $\mathbb{S}_1, \mathbb{S}_2$ are finite state sets, $R = \mathbb{S}_2$, $S = \mathbb{S}_1^Q$ where the positive integer Q is called the *block size*. Such a code is called a *block code*. Strings of the form $\varphi_*(r)$ are called *codewords*. The elements of a codeword $s = \varphi_*(r)$ are numbered as $s(0), \dots, s(Q-1)$. The following block code can be considered the paradigmatic example of codes.

Example 3.1. Suppose that $\mathbb{S}_1 = \mathbb{S}_2 = \{0, 1\}^{12}$ is the state set of both cellular automata M_1 and M_2. Let us introduce the fields $s.Addr$ and $s.Info$ of a state $r = (s_0, \dots, s_{11})$ in $\mathbb{S}_1$. The *Addr* field consists of the first 5 bits $s_0, \dots, s_4$, while the *Info* field is the last bit s_{11}. The other bits do not belong to any named field. Let $Q = 31$. Thus, we will use codewords of size 31, formed of the symbols (local states) of M_1, to encode local states of M_2. The encoding funcion φ_* assigns a codeword $\varphi_*(r) = (s(0), \dots, s(30))$ of elements of $\mathbb{S}_1$ to each element r of $\mathbb{S}_2$. Let $r = (r_0, \dots, r_{11})$. We will set $s(i).Info = r_i$ for $i = 0, \dots, 11$. The 5 bits in $s(i).Addr$ will denote the number i in binary notation. This did not determine all bits of the symbols $s(0), \dots, s(30)$ in the codeword. In particular, the bits belonging to neither the *Addr* nor the *Info* field are not determined, and the values of the *Info* field for the symbols $s(i)$ with $i \notin [0, 11]$ are not determined. To determine $\varphi_*(r)$ completely, we could set these bits to 0.

The decoding function is simpler. Given a word $s = (s(0), \dots, s(30))$ we first check whether it is a "normal" codeword, i.e. it has $s(0).Addr = 0$ and $s(i).Addr \neq 0$ for $i \neq 0$. If yes then , $r = \varphi^*(s)$ is defined by $r_i = s(i).Info$ for $i \in [0, 11]$, and the word is considered "accepted". Otherwise, $\varphi^*(s) = 0 \cdots 0$ and the word is considered "rejected".

Informally, the symbols of the codeword use their first 5 bits to mark their address within the codeword. The last bit is used to remember their part of the information about the encoded symbol. ♢

For two strings u, v, we will denote by

$$u \sqcup v \tag{3.1}$$

their concatenation.

Example 3.2. This trivial example will not be really used as a code but rather as a notational convenience. For every symbol set $\mathbb{S}_1$, blocksize Q and $\mathbb{S}_2 = \mathbb{S}_1^Q$, there is a special block code ι_Q called *aggregation* defined by

$$\iota_{Q*}((s(0), \dots, s(Q-1))) = s(0) \sqcup \cdots \sqcup s(Q-1),$$

and ι_Q^* defined accordingly. Thus, ι_{Q*} is essentially the identity: it just aggregates Q symbols of $\mathbb{S}_1$ into one symbol of $\mathbb{S}_2$. We use concatenation here since we identify all symbols with binary strings. ♢

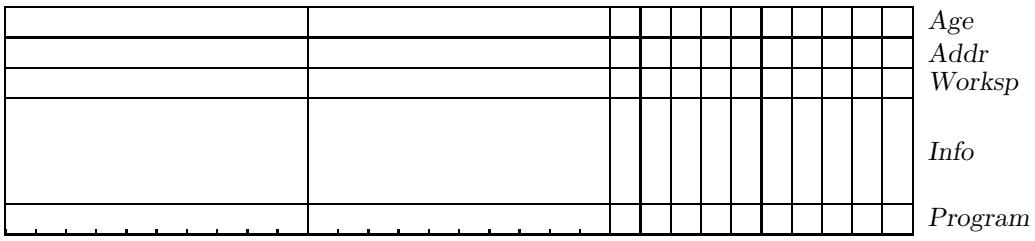

FIGURE 1. Three neighbor colonies with their tracks

The codes φ between sets R, S used in our simulations will have a feature similar to the acceptance and rejection of Example 3.1. The set R will always have a special symbol called *Vac*, the *vacant* symbol. An element $s \in S$ will be called *accepted* by the decoding if $\varphi^*(s) \neq Vac$, otherwise it is called *rejected*.

3.3. **Generalized cellular automata (abstract media).** A block code φ could be used to define a code on configurations between cellular automata M_1 and M_2. Suppose that a configuration ξ of M_2 is given. Then we could define the configuration $\xi_* = \varphi_*(\xi)$of M_1 by setting for each cell x of ξ and $0 \leq i < Q$,

$$\xi_*(Qx + i) = \varphi_*(\xi(x))(i).$$

The decoding function would be defined correspondingly. This definition of decoding is, however, unsatisfactory for our purposes. Suppose that ξ_* is obtained by encoding a configuration ξ via φ_* as before, and ζ is obtained by shifting ξ_*: $\zeta(x) = \xi_*(x-1)$. Then the decoding of ζ will return all vacant values since now the strings $(\zeta(Qx), \cdots, \zeta(Qx + Q - 1))$ are not "real" colonies. However, it will be essential for error correction that whenever parts of a configuration form a colony, even a shifted one, the decoding should notice it. With our current definition of cellular automata, the decoding function could not be changed to do this. Indeed, if ζ^* is the configuration decoded from ζ then $\zeta^*(0)$ corresponds to the value decoded from $(\zeta(0), \cdots, \zeta(Q-1))$, and $\zeta^*(1)$ to the value decoded from $(\zeta(Q), \cdots, \zeta(2Q-1))$. There is no site to correspond to the value decoded from $(\zeta(1), \cdots, \zeta(Q))$.

Our solution is to generalize the notion of cellular automata. Let us give at once the most general definition which then we will specialize later in varying degrees. The general notion is an *abstract medium*

$$AMed(\mathbb{S}, \mathbf{C}, Configs, Evols, Trajs).$$

Here, *Configs* is the set of functions $\xi : \mathbf{C} \to \mathbb{S}$ that are *configurations* and *Evols* is the set of functions $\eta : \mathbf{C} \times [0, \infty-] \to \mathbb{S}$ that are *space-time configurations* of the abstract medium. Further, *Trajs* is the set of random space-time configurations (μ, η) that are *trajectories*. In all cases that we will consider, the set *Trajs* will be given in a uniform way, as a function $Traj(\mathbf{C})$ of $\mathbf{C}$. The sets $\mathbb{S}$, $\mathbf{C}$, *Configs* and *Evols* are essentially superfluous since the set *Trajs* defines them implicitly—therefore we may omit them from the notation, so that eventually we may just write

$$AMed(Trajs).$$

Given media M_1, M_2 for the same $\mathbb{S}, \mathbf{C}$, we will write

$$M_1 \subset M_2$$

if $Trajs_1 \subset Trajs_2$. Let

$$M_1 \cap M_2$$

be the medium whose trajectory set is $Trajs_1 \cap Trajs_2$.

Let us now consider some special cases.

3.3.1. *Cellular abstract media.* All abstract media in this paper will be *cellular*: the sets *Configs* and *Evols* will be defined in the way given here. The set $\mathbb{S}$ of local states will always include a distinguished state called the *vacant state* Vac. If in a configuration ξ we have $\xi(x) \neq Vac$ then we will say that there is a *cell* at site x in ξ. We will have a positive number B called the *body size*. In ordinary cellular automata, $B = 1$. For a site x, interval $[x, x + B-]$ will be called the *body* of a possible cell with *base* x. A function $\xi : \mathbf{C} \to \mathbb{S}$ is a *configuration* if the cells in it have non-intersecting bodies.

Remark 3.3. Since not each site will be occupied by a cell, it is not even important to restrict the set of sites to integers; but we will do so for convenience. ♢

A function $\eta : \mathbf{C} \times [0, \infty-] \to \mathbb{S}$ is a *space-time configuration* if

(a) $\eta(\cdot, t)$ is a space configuration for each t;
(b) $\eta(x, t)$ is a right-continuous function of t;
(c) Each finite time interval contains only finitely many switching times for each site x;

A *dwell period* is a tuple (x, s, t_1, t_2) such that x is a site, s is a nonvacant state, and $0 \leq t_1 < t_2$ are times. The rectangle $[x, x + B-] \times [t_1, t_2-]$ is the *space-time body* of the dwell period. It is easy to see that the dwell periods in a space-time configuration have disjoint bodies. This completes the definition of the sets *Configs* and *Evols* in cellular abstract media, the only kind of media used in this paper. Therefore from now on, we may write

$$AMed(\mathbf{C}, Trajs, B).$$

We may omit any of the arguments if it is not needed for the context.

We will speak of a *lattice configuration* if all cells are at sites of the form iB for integers i. We can also talk about *lattice* space-time configurations: these have space-time bodies of the form

$$[iB, (i + 1)B-] \times [jT, (j + 1)T-]$$

for integers i, j.

A special kind of generalized cellular automaton is a straightforward redefinition of the original notion of cellular automaton, with two new but inessential parameters: a *deterministic cellular automaton*

$$CA(Tr, B, T, \mathbf{C})$$

is determined by $B, T > 0$ and a transition function $Tr : \mathbb{S}^3 \to \mathbb{S}$. We may omit some obvious arguments from this notation. A lattice space-time configuration η with parameters B, T is a *trajectory* of this automaton if

$$\eta(x, t) = Tr(\eta(x - B, t - T), \eta(x, t - T), \eta(x + B, t - T))$$

holds for all x, t with $t \geq T$. For a space-time configuration η let us write

$$\overline{Tr}(\eta, x, t, B) = Tr(\eta(x - B, t), \eta(x, t), \eta(x + B, t)). \tag{3.2}$$

We will omit the argument B when it is obvious from the context. Probabilistic cellular automata and perturbations are generalized correspondingly as

$$PCA(\mathbf{P}, B, T, \mathbf{C}), \qquad CA_\varepsilon(Tr, B, T, \mathbf{C}).$$

From now on, whenever we talk about a deterministic, probabilistic or perturbed cellular automaton we understand one also having parameters B, T.

We will have two kinds of abstract medium that are more general than these cellular automata. We have a *constant-period* medium if in all its trajectories, all dwell period lengths are multiples of some constant T. Otherwise, we have a *variable-period* medium.

3.3.2. *Block codes between cellular automata.* In a cellular abstract medium with body size B, a colony of size Q is defined as a set of cells $x+iB$ for $i \in [0, Q-1]$. Thus, the union of the cell bodies of body size B in a colony of size Q occupies some interval $[x, x+QB-]$. A block code will be called *overlap-free* if for every string $(s(0), \dots s(n-1))$, and all $i \le n-Q$, if both $(s(0), \dots, s(Q-1))$ and $(s(i+1), \dots, s(i+Q-1))$ are accepted then $i \ge Q$. In other words, a code is overlap-free if two accepted words cannot overlap in a nontrivial way. The code in Example 3.1 is overlap-free. All block-codes considered from now on will be overlap-free. Overlap-free codes are used, among others, in [18].

3.3.3. *Codes on configurations.* A block code φ of block size Q can be used to define a code on configurations between generalized abstract media M_1 and M_2. Suppose that a configuration ξ of M_2, which is an $AMed(QB)$, is given. Then we define the configuration $\xi_* = \varphi_*(\xi)$ of M_1, which is an $AMed(B)$, by setting for each cell x of ξ and $0 \le i < Q$,

$$\xi_*(x + iB) = \varphi_*(\xi(x))(i).$$

Suppose that a configuration ξ of M_1 is given. We define the configuration $\xi^* = \varphi^*(\xi)$ of M_2 as follows: for site x, we set $\xi^*(x) = \varphi^*(s)$ where

$$s = (\xi(x), \xi(x+B), \dots, \xi(x+(Q-1)B)). \tag{3.3}$$

If ξ is a configuration with $\xi = \varphi_*(\zeta)$ then, due to the overlap-free nature of the code, the value $\xi^*(x)$ is nonvacant only at positions x where $\zeta(x)$ is nonvacant. If ξ is not the code of any configuration then it may happen that in the decoded configuration $\varphi^*(\xi)$, the cells will not be exactly at a distance QB apart. The overlap-free nature of the code garantees that the distance of cells in $\varphi^*(\xi)$ is at least QB even in this case.

3.4. **Block simulations.** Suppose that M_1 and M_2 are deterministic cellular automata where $M_i = CA(Tr_i, B_i, T_i)$, and φ is a block code with

$$B_1 = B, \qquad B_2 = QB.$$

The decoding function may be as simple as in Example 3.1: there is an *Info* track and once the colony is accepted the decoding function depends only on this part of the information in it.

For each space-time configuration η of M_1, we can define $\eta^* = \varphi^*(\eta)$ of M_2 by setting

$$\eta^*(\cdot, t) = \varphi^*(\eta(\cdot, t)). \tag{3.4}$$

We will say that the code φ is a *simulation* if for each configuration ξ of M_2, for the trajectory (μ, η) of M_1, such that $\eta(\cdot, 0, \omega) = \varphi_*(\xi)$ for almost all ω, the random space-time configuration (μ, η^*) is a trajectory of M_2. (We do not have to change μ here since the ω in $\eta^*(x, t, \omega)$ is still coming from the same space as the one in $\eta(x, t, \omega)$.)

We can view φ_* as an encoding of the initial configuration of M_2 into that of M_1. A space-time configuration η of M_1 will be viewed to have a "good" initial configuration $\eta(\cdot, 0)$ if the latter is $\varphi_*(\xi)$ for some configuration of M_2. Our requirements say that from every trajectory of M_1 with good initial configurations, the simulation-decoding results in a trajectory of M_2.

Let us show one particular way in which the code φ can be a simulation. For this, the function Tr_1 must behave in a certain way which we describe here. Assume that

$$T_1 = T,\ T_2 = UT$$

for some positive integer U called the *work period size*. Each cell of M_1 will go through a period consisting of U steps in such a way that the *Info* field will be changed only in the last step of this period. The initial configuration $\eta(\cdot, 0) = \varphi_*(\xi)$ is chosen in such a way that each cell is at the beginning of its work period. By the nature of the code, in the initial configuration, cells of M_1 are grouped into colonies.

Once started from such an initial configuration, during each work period, each colony, in cooperation with its two neighbor colonies, computes the new configuration. With the block code in Example 3.1, this may happen as follows. Let us denote by r_-, r_0, r_+ the value in the first 12 bits of the *Info* track in the left neighbor colony, in the colony itself and in the right neighbor colony respectively. First, r_- and r_+ are shipped into the middle colony. Then, the middle colony computes $s = Tr_2(r_{-1}, r_0, r_1)$ where Tr_2 is the transition function or M_2 and stores it on a memory track. (It may help understanding how this happens if we think of the possibilities of using some mail, memory and workspace tracks.) Then, in the last step, s will be copied onto the *Info* track.

Such a simulation is called a *block simulation*.

Example 3.4. Let us give a trivial example of a block simulation which will be applied, however, later in the paper. Given a one-dimensional transition function $Tr(x, y, z)$ with state space $\mathbb{S}$, we can define for all positive integers Q an *aggregated* transition function $Tr^Q(u, v, w)$ as follows. The state space of of Tr^Q is $\mathbb{S}^Q$. Let $\mathbf{r}_j = (r_j(0), \dots, r_j(Q-1))$ for $j = -1, 0, 1$ be three elements of $\mathbb{S}^Q$. Concatenate these three strings to get a string of length $3Q$ and apply the transition function Tr to each group of three consecutive symbols to obtain a string of length $3Q - 2$ (the end symbols do not have both neighbors). Repeat this Q times to get a string of Q symbols of $\mathbb{S}$: this is the value of $Tr^Q(\mathbf{r}_{-1}, \mathbf{r}_0, \mathbf{r}_1)$.

For $M_1 = CA(\mathbb{S}, Tr, B, T)$ and $M_2 = CA(\mathbb{S}^Q, Tr^Q, QB, QT)$, the aggregation code ι_Q defined in Example 3.2 will be a block simulation of M_2 by M_1 with a work period consisting of $U = Q$ steps. If along with the transition function Tr, there were some fields $F, G, \dots \subset All$ also defined then we define, say, the field F in the aggregated cellular automaton as $\bigcup_{i=0}^{Q-1}(F + i\|\mathbb{S}\|)$. Thus, if $\mathbf{r} = r(0) \sqcup \cdots \sqcup r(Q-1)$ is a state of the aggregated cellular automaton then $\mathbf{r}.F = r(0).F \sqcup r(1).F \sqcup \cdots \sqcup r(Q-1).F$. ♢

A transition function Tr is *universal* if for every other transition function Tr' there are Q, U and a block code φ such that φ is a block simulation of $CA(Tr', Q, U)$ by $CA(Tr, 1, 1)$.

Theorem 3.5 (Universal cellular automata)**.** *There is a universal transition function.*

Sketch of proof: This theorem is proved somewhat analogously to the theorem on the existence of universal Turing machines. If the universal transition function is Tr then for simulating another transition function Tr', the encoding demarcates colonies of appropriate size with $Addr = 0$, and writes a string *Table* that is the code of the transition table of Tr' onto a special track called *Prog* in each of these colonies. The computation is just a table-look-up: the triple (r_-, r_0, r_+) mentioned in the above example must be looked up in the transition table. The transition function governing this activity does not depend on the particular content of the *Prog* track, and is therefore independent of Tr'. For references to the first proofs of universality (in a technically different but similar sense), see [4, 26] □

Note that a universal cellular automaton cannot use codes similar to Example 3.1. Indeed, in that example, the capacity of the cells of M_1 is at least the binary logarithm of the colony size, since

each colony cell contained its own address within the colony. But if M_1 is universal then the various simulations in which it participates will have arbitrarily large colony sizes.

The size Q of the simulating colony will generally be very large also since the latter contains the whole table of the simulated transition function. There are many special cellular automata M_2, however, whose transition function can be described by a small computer program and computed in relatively little space and time (linear in the size $\|\mathbb{S}_2\|$). The universal transition function will simulate these with correspondingly small Q and U. We will only deal with such automata.

3.5. **Single-fault-tolerant block simulation.** Here we outline a cellular automaton M_1 that block-simulates a cellular automaton M_2 correctly as long as at most a single error occurs in a colony work period of size U. The outline is very informal: it is only intended to give some framework to refer to later: in particular, we add a few more fields to the fields of local states introduced earlier. For simplicity, these fields are not defined here in a way to make the cellular automaton separable in the sense defined in 2.2.1. They could be made so with a few adjustments but we want to keep the introduction simple.

The automaton M_1 is not universal, i.e. the automaton M_2 cannot be chosen arbitrarily. Among others, this is due to the fact that the address field of a cell of M_1 will hold its address within its colony. But we will see later that universality is not needed in this context.

The cells of M_1 will have, besides the *Addr* field, also a field *Age*. If no errors occur then in the i-th step of the colony work period, each cell will have the number i in the field *Age*. There are also fields called *Mail*, *Info*, *Work*, *Hold*, *Prog*.

The *Info* field holds the state of the represented cell of M_2 in three copies. The *Hold* field will hold parts of the final result before it will be, in the last step of the work period, copied into *Info*. The role of the other fields is clear.

The program will be described from the point of view of a certain colony $\mathcal{C}$. Here is an informal description of the activities taking place in the first third of the work period.

1. From the three thirds of the *Info* field, by majority vote, a single string is computed. Let us call it the *input string*. This computation, as all others, takes place in the workspace field *Work*; the *Info* field is not affected. The result is also stored in the workspace.
2. The input strings computed in the two neighbor colonies are shipped into $\mathcal{C}$ and stored in the workspace separately from each other and the original input string.
3. The workspace field behaves as a universal automaton, and from the three input strings and the *Prog* field, computes the string that would be obtained by the transition function of M_2 from them. This string will be copied to the first third of the *Hold* track.

In the second part of the work period, the same activities will be performed, except that the result will be stored in the second part of the *Hold* track. Similarly with the third part of the work period. In a final step, the *Hold* field is copied into the *Info* field.

The computation is coordinated with the help of the *Addr* and *Age* fields. It is therefore important that these are correct. Fortunately, if a single fault changes such a field of a cell then the cell can easily restore it using the *Addr* and *Age* fields of its neighbors.

It is not hard to see that with such a program (transition function), if the colony started with "perfect" information then a single fault will not corrupt more than a third of the colony at the end of the work period. On the other hand, if two thirds of the colony was correct at the beginning of the colony work period and there is no fault during the colony work period then the result will be "perfect".

3.6. **General simulations.** The main justification of the general notion of abstract media is that it allows a very general definition of simulations: a *simulation* of abstract medium M_2 by abstract

medium M_1 is given by a pair

$$(\varphi_*, \Phi^*)$$

where Φ^* is a mapping of the set of space-time configurations of M_1 into those of M_2 (the decoding), and φ_* is a mapping of the set of configurations of M_2 to the set of configurations of M_1 (the encoding for initialization). Let us denote

$$\eta^* = \Phi^*(\eta).$$

We require, for each trajectory η for which the initial configuration has the encoded form $\eta(\cdot, 0) = \varphi_*(\xi)$, that η^* is a trajectory of M_2 with $\eta^*(\cdot, 0) = \xi$.

A simulation will be called *local*, if there is a finite space-time rectangle $V^* = I \times [-u, 0]$ such that $\Phi^*(\eta)(w, t)$ depends only on $\eta((w, t) + V^*)$. Together with the shift-invariance property, the locality property implies that a simulation is determined by a function defined on the set of configurations over V^*. All simulations will be local unless stated otherwise. Corollary 7.3 gives an example of non-local simulation.

If $u = 0$ then the configuration $\eta^*(\cdot, t)$ depends only on the configuration $\eta(\cdot, t)$. In this case, the simulation could be called "memoryless". For a memoryless simulation, the simulation property is identical to the one we gave at the beginning of Subsection 3.4. Our eventual simulations will not be memoryless but will be at least *non-anticipating*: we will have $u > 0$, i.e. the decoding looks back on the space-time configuration during $[t - u, t]$, but still does not look ahead. In particular, the value of $\eta^*(\cdot, 0)$ depends only on $\eta(\cdot, 0)$ and therefore the simulation always defines also a decoding function φ^* on space-configurations. From now on, this decoding function will be considered part of the definition of the simulation, i.e. we will write

$$\Phi = (\varphi_*, \varphi^*, \Phi^*).$$

Suppose that a sequence $M_1, M_2, \dots$ of abstract media is given along with simulations $\Phi_1, \Phi_2, \dots$ such that Φ_k is a simulation of M_{k+1} by M_k. Such a system will be called an *amplifier*. Amplifiers are like renormalization groups in statistical physics. Of course, we have not seen any nontrivial example of simulation other than between deterministic cellular automata, so the idea of an amplifier seems far-fetched at this moment.

3.6.1. *Simulation between perturbations.* Our goal is to find nontrivial simulations between cellular automata M_1 and M_2, especially when these are not deterministic. If M_1, M_2 are probabilistic cellular automata then the simulation property would mean that whenever we have a trajectory (μ, η) of M_1 the random space-time configuration η^* decoded from η would be a trajectory of M_2. There are hardly any nontrivial examples of this sort since in order to be a trajectory of M_2, the conditional probabilities of $\varphi^*(\eta)$ must satisfy certain equations defined by $\mathbf{P}_2$, while the conditional probabilitiees of η satisfy equations defined by $\mathbf{P}_1$.

There is more chance of success in the case when M_1 and M_2 are perturbations of some deterministic cellular automata since in this case, only some inequalities must be satisfied. The goal of improving reliability could be this. For some universal transition function Tr_2, and at least two different initial configurations ξ_i $(i = 0, 1)$, find Tr_1, Q, U, c with $B_1 = B$, $B_2 = BQ$, $T_1 = T$, $T_2 = TU$ and a block simulation Φ_1 such that for all $\varepsilon > 0$, if $\varepsilon_1 = \varepsilon$, $\varepsilon_2 = c\varepsilon^2$ and M_k is the perturbation

$$CA_{\varepsilon_k}(Tr_k, B_k, T_k, \mathbf{Z})$$

then Φ_1 is a simulation of M_2 by M_1. The meaning of this is that even if we have to cope with the fault probability ε the simulation will compute Tr_2 with a much smaller fault probability $c\varepsilon^2$. The hope is not unreasonable since in Subsection 3.5, we outlined a single-fault-tolerant block simulation while the probability of several faults happening during one work period is only of the order of

$(QU\varepsilon)^2$. However, it turns out that the only simply stated property of a perturbation that survives noisy simulation is a certain initial stability property (see below).

3.6.2. *Error-correction.* Even if the above goal can be achieved, the reason for the existence of the simulated more reliable abstract medium is to have feedback from it to the simulating one. The nature of this feedback will be defined in the notion of error-correction to whose definition now we proceed. Let us call the set

$$\Sigma_0 = \{0, 1, \#, *\} \tag{3.5}$$

the *standard alphabet*. Symbol # will be used to delimit binary strings, and $*$ will serve as a "don't-care" symbol. Each field F of a cell state such that the field size is even, can be considered not only a binary string but a string of (half as many) symbols in the standard alphabet. If r, s are strings in $(\Sigma_0)^n$ then

$$r \preceq s$$

will mean that $s(i) = r(i)$ for all $0 \le i < n$ such that $r(i) \neq *$. Thus, a don't-care symbol $r(i)$ imposes no restriction on s in this relation. There will be two uses of the don't-care symbol.

- The more important use will come in defining the code used for error-correction in a way that it requires the correction of only those parts of the information in which correction is desirable.
- The less important use is in defining the notion of "monotonic output": namely, output that contains more and more information as time proceeds. This is convenient e.g. for continuous-time cellular automata, where it is difficult to say in advance when the computation should end.

For codes φ_*, ψ_*, we will write

$$\psi_* \preceq \varphi_*$$

if for all s we have $\psi_*(s) \preceq \varphi_*(s)$.

Let us define the notion of error-correction. Let $\Phi = (\varphi_*, \varphi^*, \Phi^*)$ be a simulation whose encoding φ_* is a block code with block size Q, between abstract cellular media M_i $(i = 1, 2)$. Let $T_i > 0$ $(i = 1, 2)$ be some parameters, and $\varphi_{**} \preceq \varphi_*$ a block code of blocksize Q. We say that Φ has the *ε-error-correction property* with respect to φ_{**}, T_1, T_2 if the following holds for every configuration ξ of M_2 and every trajectory (μ, η) of M_1 with $\eta(\cdot, 0) = \varphi_*(\xi)$.

Let $\eta^* = \Phi^*(\eta)$ and let x_1, x_2 be sites where x_1 has address a in the Q-colony with base x_2, let t_0 be some time. Let $\mathcal{E}$ be the event that $\eta^*(x_2, \cdot)$ is nonvacant during $[t_0 - T_2/3, t_0]$ and let $\mathcal{E}'$ be the event that for each t in $[t_0 - T_1/3, t_0]$ there is a t' in $[t_0 - T_2/3, t]$ with

$$\varphi_{**}(\eta^*(x_2, t'))(a) \preceq \eta(x_1, t).$$

Then $\mathrm{Prob}\{\, \mathcal{E} \cap \neg\mathcal{E}' \,\} < \varepsilon$.

Informally, this means that for all x_1, x_2, a in the given relation, the state $\eta(x_1, t)$ is with large probability what we expect by encoding some $\eta^*(x_2, t')$ via φ_{**} and taking the a-th symbol of the codeword. Error-correction is only required for a code $\varphi_{**} \preceq \varphi_*$ since φ_* determines the value of many fields as a matter of initialization only: these fields need not keep their values constant during the computation, and therefore φ_{**} will assign don't-care symbols to them. The code φ_{**} will thus generally be obtained by a simple modification of φ_*.

Example 3.6. Let

$$B_1 = 1, \qquad B_2 = Q, \qquad T_1 = 1, \qquad T_2 = U$$

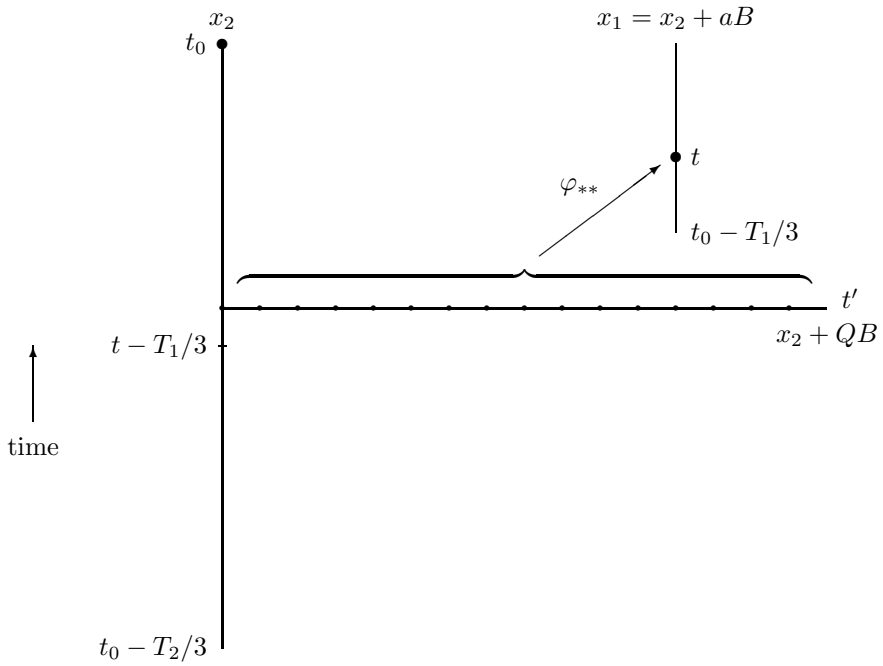

FIGURE 2. Error correction

for some $U > Q$. Assume that for $k = 1, 2$, our media M_k are cellular generalized media with body sizes B_k and state spaces $\mathbb{S}_k$. Assume further that M_2 has at least a field F_2 with $|F_2| \leq Q/3$ and M_1 has at least the fields $F_1, Addr, Age$, with $|F_1| = 2$. For a state $s \in \mathbb{S}_2$, let the string

$$s' = \varphi_*(s) \in \mathbb{S}_1^Q$$

be defined as follows. Take the binary string $s.F_2$, repeat it 3 times, pad it with $*$'s to a string of size Q of the standard alphabet: let this be a string $(f(0), \ldots, f(Q-1))$. Now for each address b, let

$$s'(b).F_1 = f(b), \qquad s'(b).Addr = b, \qquad s'(b).Age = 0,$$

and let all other fields of $s'(b)$ be filled with $*$'s. The definition of $s'' = \varphi_{**}(s)$ starts as the definition of s' with the only difference that set $s''(b).Age = *$. Thus, the code $\varphi_*(s)$ encodes a redundant version of $s.F_2$ onto the F_1 track of the block s' and initializes the $Addr$ and Age tracks to the values they would have at the beginning of a work period. The code $\varphi_{**}(s)$ leaves all tracks other than F_1 and $Addr$ undetermined, since it will have to be compared with the state of the colony also at times different from the beginning of the work period. $\diamondsuit$

Suppose that an amplifier $(M_k, \Phi_k)_{k\geq 1}$ is given along with the sequences $\varphi_{k**}, T_k, \varepsilon''_k$. We will call this structure an *error-correcting amplifier* if for each k, the simulation Φ_k has the ε''_k- *error correction property* with respect to φ_{k**}, T_k, T_{k+1}.

3.7. **Remembering a bit: proof from an amplifier assumption.** The following lemma will be proved later in the paper.

Lemma 3.7 (Initially Stable Amplifier)**.** *We can construct the following objects, for $k = 1, 2, \dots$.*

(a) *Media M_k over state space $\mathbb{S}_k$, simulations $\Phi_k = (\varphi_{k*}, \varphi^{k*}, \Phi^{k*})$ and sequences φ_{k**}, ε''_k, T_k forming an ε''_k- error correcting amplifier with $\sum_k \varepsilon''_k < 1/6$.*

(b) *(Initial stability) Parameters ε_k, B_1 with $\sum_k \varepsilon_k < 1/6$, transition function Tr_1 and two configurations ξ_0, ξ_1 such that, defining $\xi^1_u = \xi_u$, $\xi^{k+1}_u = \varphi^{k*}(\xi^k_u)$ for $u = 0, 1$, we have*

$$M_1 = CA_{\varepsilon_1}(\mathrm{Tr}_1, B_1, T_1, \mathbf{Z}).$$

Further, for each k, u, for each trajectory η of M_k with $\eta(\cdot, 0) = \xi^k_u$, for all $t < T_k$, for each site x, we have

$$\mathrm{Prob}\{\, \eta(x,t) \neq \eta(x,0) \,\} < \varepsilon_k.$$

(c) *Parameters Q_k such that the codes φ_{k*} are block codes with block size Q_k.*

(d) *(Broadcast) Fields F^k for the state spaces $\mathbb{S}_k$ such that for each k, for each address $a \in [0, Q_k - 1]$ and state $s \in \mathbb{S}_{k+1}$, for each $u \in \{0, 1\}$ and site x we have*

$$\xi^k_u(x).F^k = u, \tag{3.6}$$

$$s.F^{k+1} \preceq \varphi_{k**}(s)(a).F^k. \tag{3.7}$$

Equation (3.6) says for the configurations ξ^k_u, (obtained by decoding from the initial configuration ξ_u) in each cell of medium M_k, field F_k has value u. Equation (3.7) says the following. Assume for symbol $s \in \mathbb{S}_{k+1}$ we have $s.F_{k+1} = u \neq *$. Then the encoding function φ_{**} encodes s into a colony of Q_k symbols $r_0, \dots, r_{Q_k - 1}$ such that for each a, we have $s_a.F_k = u$. Thus, the F_{k+1} field of s gets "broadcast" into the F_k field of each symbol of the code of s. This way, even if property (3.6) were assumed only for a fixed level k, property would (3.7) imply it for all $i < k$.

Let us use this lemma to prove the first theorem.

Proof of Theorem 2.5. Let us use the amplifier defined in the above lemma. Let η^1 be a trajectory of the medium M_1 with initial configuration ξ_u. Let η^k be defined by the recursion $\eta^{k+1} = \Phi^*_k(\eta^k)$. Let (x_1, t_1) be a space-time point in which we want to check $\eta(x_1, t_1).F^1 = u$. There is a sequence of points $x_1, x_2, \dots$ such that x_{k+1} is a cell of $\eta^{k+1}(\cdot, 0)$ containing x_k in its body with some address b_k. There is a first n with $t_1 < T_n/3$. Let $\mathcal{F}_k$ be the event that $\eta^k(x_k, t).F^k = u$ for t in $[t_1 - T_k/3, t_1]$. The theorem follows from the bounds on $\sum_k \varepsilon_k$ and $\sum_k \varepsilon''_k$ and from

$$\mathrm{Prob}\{\, \neg(\mathcal{F}_1 \cap \dots \cap \mathcal{F}_n) \,\} \leq \varepsilon_n + \sum_{k=1}^{n-1} \varepsilon''_k. \tag{3.8}$$

To prove this inequality, use

$$\neg(\mathcal{F}_1 \cap \dots \cap \mathcal{F}_n) = \neg\mathcal{F}_n \cup \bigcup_{k=1}^{n-1} (\neg\mathcal{F}_k \cap \mathcal{F}_{k+1}).$$

By the construction, $\eta^n(x_n, 0).F^n = u$. Since the duration of $[0, t_1]$ is less than T_n we have $\mathrm{Prob}\{\, \eta^n(x_n, t_1) \neq \eta^n(x_n, 0) \,\} < \varepsilon_n$ by the initial stability property, proving $\mathrm{Prob}\{\, \neg\mathcal{F}_n \,\} \leq \varepsilon_n$. The error-correction property and the broadcast property imply $\mathrm{Prob}\{\, \mathcal{F}_{k+1} \cap \neg\mathcal{F}_k \,\} \leq \varepsilon''_k$. □

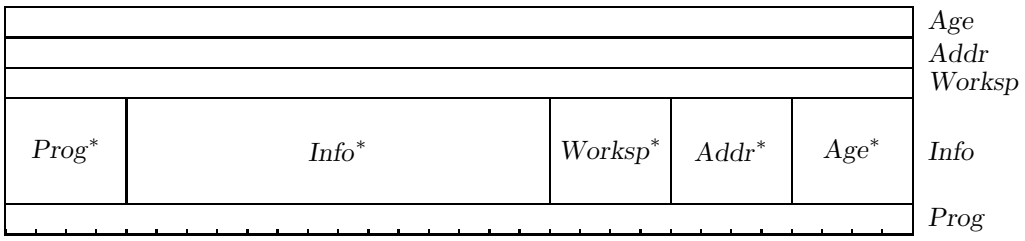

FIGURE 3. Fields of a cell simulated by a colony

4. HIERARCHY

4.1. **Hierarchical codes.** The present section may seem a long distraction from the main course of exposition but many readers of this paper may have difficulty imagining an infinite hierarchical structure built into a configuration of a cellular automaton. Even if we see the possibility of such structures it is important to understand the great amount of flexibility that exists while building it. Formally, a hiearchy will be defined as a "composite code". Though no decoding will be mentioned in this subsection, it is still assumed that to all codes φ_* mentioned, there belongs a decoding function φ^* with $\varphi^*(\varphi_*(x)) = x$.

4.1.1. *Composite codes.* Let us discuss the hierarchical structure arising in an amplifier. If φ, ψ are two codes then $\varphi \circ \psi$ is defined by $(\varphi \circ \psi)_*(\xi) = \varphi_*(\psi_*(\xi))$ and $(\varphi \circ \psi)^*(\zeta) = \psi^*(\varphi^*(\zeta))$. It is assumed that ξ and ζ are here configurations of the appropriate cellular automata, i.e. the cell body sizes are in the corresponding relation. The code $\varphi \circ \psi$ is called the *composition* of φ and ψ.

For example, let M_1, M_2, M_3 have cell body sizes $1, 31, 31^2$ respectively. Let us use the code φ from Example 3.1. The code $\varphi^2 = \varphi \circ \varphi$ maps each cell c of M_3 with body size 31^2 into a "supercolony" of $31 \cdot 31$ cells of body size 1 in M_1. Suppose that $\zeta = \varphi_*^2(\xi)$ is a configuration obtained by encoding from a lattice configuration of body size 31^2 in M_3, where the bases of the cells are at positions $-480 + 31^2 i$. (We chose -480 only since $480 = (31^2 - 1)/2$ but we could have chosen any other number.) Then ζ can be broken up into colonies of size 31 starting at any of those bases. Cell 55 of M_1 belongs to the colony with base $47 = -480 + 17 \cdot 31$ and has address 8 in it. Therefore the address field of $\zeta(55)$ contains a binary representation of 8. The last bit of this cell encodes the 8-th bit the of cell (with base) 47 of M_2 represented by this colony. If we read together all 12 bits represented by the *Info* fields of the first 12 cells in this colony we get a state $\zeta^*(47)$ (we count from 0). The cells with base $-15 + 31j$ for $j \in \mathbb{Z}$ with states $\zeta^*(-15 + 31j)$ obtained this way are also broken up into colonies. In them, the first 5 bits of each state form the address and the last bits of the first 12 cells, when put together, give back the state of the cell represented by this colony. Notice that these 12 bits were really drawn from 31^2 cells of M_1. Even the address bits in $\zeta^*(47)$ come from different cells of the colony with base 47. Therefore the cell with state $\zeta(55)$ does not contain information allowing us to conclude that it is cell 55. It only "knows" that it is the 8-th cell within its own colony (with base 47) but does not know that its colony has address 17 within its supercolony (with base $-15 \cdot 31$) since it has at most one bit of that address.

4.1.2. *Infinite composition.* A code can form composition an arbitrary number of times with itself or other codes. In this way, a hierarchical, i.e. highly nonhomogenous, structure can be defined using cells that have only a small number of states. A *hierarchical code* is given by a sequence

$$(\mathbb{S}_k, Q_k, \varphi_{k*})_{k \geq 1} \tag{4.1}$$

where $\mathbb{S}_k$ is an alphabet, Q_k is a positive integer and $\varphi_{k*} : \mathbb{S}_{k+1} \to \mathbb{S}_k^{Q_k}$ is an encoding function. Since $\mathbb{S}_k$ and Q_k are implicitly defined by φ_{k*} we can refer to the code as just (φ_k).

We will need a composition $\varphi_{1*} \circ \varphi_{2*} \circ \cdots$ of the codes in a hierarchical code since the the definition of the initial configuration for M_1 in the amplifier depends on all codes φ_{i*}. What is the meaning of this? We will want to compose the codes "backwards", i.e. in such a way that from a configuration ξ^1 of M_1 with cell body size 1, we can decode the configuration $\xi^2 = \varphi_1^*(\xi^1)$ of M_2 with cell body size $B_2 = Q_1$, configuration $\xi^3 = \varphi_2^*(\xi^2)$, of M_3 with body size $B_3 = Q_1 Q_2$, etc. Such constructions are not unknown, they were used e.g. to define "Morse sequences" with applications in group theory as well in the theory of quasicrystals ([18, 24]).

Let us call a sequence $a_1, a_2, \ldots$ with $0 \le a_k < Q_k$ *non-degenerate for* $Q_1, Q_2, \ldots$ if there are infinitely many k with $a_k > 0$ and infinitely many k with $a_k < Q_k - 1$. The pair of sequences

$$(Q_k, a_k)_{k=1}^{\infty} \tag{4.2}$$

with non-degenerate a_k will be called a *block frame* of our hierarchical codes. All our hierarchical codes will depend on some fixed block frame, $((Q_k, a_k))$, but this dependence will generally not be shown in the notation.

Remarks 4.1.

1. The construction below does not need the generality of an arbitrary non-degenerate sequence: we could have $a_k = 1$ throughout. We feel, however, that keeping a_k general makes the construction actually more transparent.
2. It is easy to extend the construction to degenerate sequences. If e.g. $a_k = 0$ for all but a finite number of k then the process creates a configuration infinite in right direction, and a similar construction must be added to attach to it a configuration infinite in the left direction.

♢

For a block frame $((Q_k, a_k))$, a finite or infinite sequence $(s_1, a_1), (s_2, a_2), \ldots$ will be called *fitted* to the hierarchical code (φ_{k*}) if

$$\varphi_{k*}(s_{k+1})(a_k) = s_k$$

holds for all k. For a finite or infinite space size N, let

$$\begin{aligned} B_1 &= 1, \\ B_k &= Q_1 \cdots Q_{k-1} \text{ for } k > 1, && (4.3) \\ K = K(N) &= \sup_{B_k < N} k + 1, && (4.4) \\ o_k &= -a_1 B_1 - \cdots - a_{k-1} B_{k-1}, \\ C_k(x) &= o_k + x B_k, && (4.5) \end{aligned}$$

The following properties are immediate:

$$\begin{aligned} o_1 &= C_1(0) = 0, \\ o_k &= o_{k+1} + a_k B_k, && (4.6) \\ 0 &\in o_k + [0, B_k - 1]. \end{aligned}$$

Proposition 4.2. *Let us be given a fitted sequence $(s_k, a_k)_{k \ge 1}$. Then there are configurations ξ^k of M_k over $\mathbb{Z}$ such that for all $k \ge 1$ we have*

$$\begin{aligned} \varphi_{k*}(\xi^{k+1}) &= \xi^k, \\ \xi^k(o_k) &= s_k. \end{aligned}$$

The infinite code we are interested in is ξ^1. Note that in this construction, s_k is the state of the site o_k in configuration ξ^k whose body contains the site 0. This site has address a_k in a colony with base o_{k+1} in ξ^{k+1}.

Proof. Let

$$\xi_k^k \tag{4.7}$$

be the configuration of M_k which has state s_k at site o_k and arbitrary states at all other sites $C_k(x)$, with the following restriction in case of a finite space size N. Let $\xi_k^k \neq Vac$ only for $k \leq K = K(N)$ and $\xi_K^K(o_K + z) \neq Vac$ only if

$$0 \leq zB_K < N. \tag{4.8}$$

Let

$$\xi_k^i = \varphi_{i*}(\varphi_{(i+1)*}(\cdots \varphi_{(k-1)*}(\xi_k^k)\cdots)) \tag{4.9}$$

for $k > i \geq 1$. We have

$$\xi_{k+1}^k(o_k) = \varphi_{k*}(\xi_{k+1}^{k+1}(o_{k+1}))(a_k) = \xi_k^k(o_k) \tag{4.10}$$

where the first equation comes by definition, the second one by fittedness. The encoding conserves this relation, so the partial configuration $\xi_{k+1}^i(o_{k+1} + [0, B_{k+1} - 1])$ is an extension of $\xi_k^i(o_k + [0, B_k - 1])$. Therefore the limit $\xi^i = \lim_k \xi_k^i$ exists for each i. Since (a_k) is non-degenerate the limit extends over the whole set of integer sites. □

Though ξ^1 above is obtained by an infinite process of encoding, no infinite process of decoding is needed to yield a single configuration from it: at the k-th stage of the decoding, we get a configuration ξ^k with body size B_k.

4.1.3. *Controlling, identification.* The need for some freedom in constructing infinite fitted sequences leads to the following definitions. For alphabet $\mathbb{S}$ and field F let

$$\mathbb{S}.F = \{\, w.F : w \in \mathbb{S} \,\}.$$

Then, of course, $\|\mathbb{S}.F\| = |F|$. Let $D = \{d_0, \ldots, d_{|D|-1}\} \subset [0, Q-1]$ be a set of addresses with $d_i < d_{i+1}$. For a string s, let

$$s(D).F$$

be the string of values $(s(d_0).F, \ldots, s(d_{|D|-1}).F)$ so that

$$s.F = s([0, Q-1]).F.$$

Field F *controls* an address a in code φ_* *via* function $\gamma : \mathbb{S}_1.F \to \mathbb{S}_1$ if

(a) For all $r \in \mathbb{S}_1.F$ there is an s with $\varphi_*(s)(a).F = r$; in other words, $\varphi_*(s)(a).F$ runs through all possible values for this field as s varies.
(b) For all s we have $\varphi_*(s)(a) = \gamma(\varphi_*(s)(a).F)$; in other words, the field $\varphi_*(s)(a).F$ determines all the other fields of $\varphi_*(s)(a)$.

From now on, in the present subsection, whenever we denote a field by F^k and a code by φ_{k*} we will implicitly assume that F^k controls address a_k in φ_{k*} unless we say otherwise. (The index k in F^k is not an exponent.)

Suppose that fields F^1, F^2 are defined for cellular automata M_1 and M_2 between which the code φ with blocksize Q is given. Suppose that set D satisfies $|D| = |F^2|/|F^1|$. We say that in φ_*, field F^1 over D is *identified with* F^2 if in any codeword $w = \varphi_*(s)$, the string $w(d_0).F^1 \sqcup \cdots \sqcup w(d_{|D|-1}).F^1$ is identical to $s.F^2$. Conversely, thus $w(d_i).F^1 = s.F^2([i|F^1|, (i+1)|F^1| - 1])$. The identification of F^2

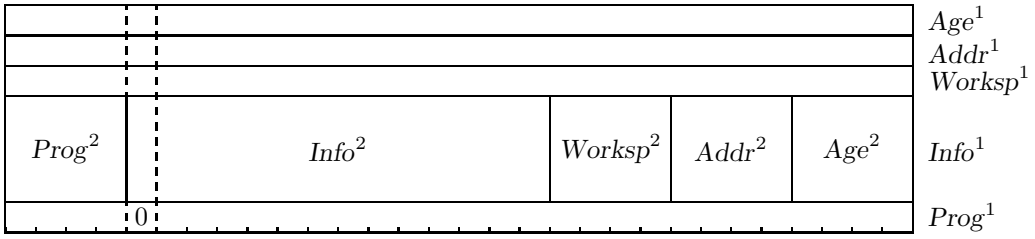

FIGURE 4. Assume that in the code of this figure, *Addr*, *Age* and *Worksp* are constant and $Prog^1$ has always 0 in its position 4. Then address 4 is controlled by $Info^1$, and $Info^2$ is identified with $Info^1$ over the addresses 4-17.

with F^1 over D implies that if a simulation has error-correction in F^1 over D then the information restored there is $s.F^2$.

Example 4.3. Consider the simulation outlined in Subsection 3.5, and let us call the encoding φ_*. Let F^1 be the *Info* field of M_1: we denote it by $Info^1$. Assume further that the simulated medium M_2 is of a similar type, so it has an $Info^2$ field. Assume for simplicity

$$6 \mid Q, \qquad |Info^1| = 2, \qquad |Info^2| = Q/3.$$

The $Info^1$ field of cells in interval $[0, Q/3-1]$ of a simulating colony represents the whole state of a simulated cell of M_2. The $Info^1$ field on $[Q/3, Q-1]$ is only used for redundancy. Only a segment of the $Info^1$ field of $[0, Q/3-1]$, say the one on $[0, Q/6-1]$ is used to represent $Info^2$ of the simulated cell. The rest is used for encoding the other fields. Hence $Info^1$ on $[0, Q/6-1]$ is identified with $Info^2$ in our code φ_*.

Let $s' = \varphi_*(s)(1)$ be the state of the cell with address 1 of a colony of M_1 which is the result of encoding state s of M_2. Let $s'.Info^1$ be the third and fourth bits of $s.Info^2$, $s'.Addr = 1$, and $s'.F = 0$ for all fields different from these two. Then $Info^1$ controls address 1 in the code φ_*. $\diamondsuit$

If for each address a, the field F^1 over $\{a\}$ is identified with F^2 then we say that F^2 is *broadcast* to F^1 (since this means that the code copies the value of $s.F^2$ into the F^1 field of each cell of $\varphi_*(s)$).

Let us be given

$$\Psi = ((\mathbb{S}_k, Q_k, \varphi_{k*}, F^k, \gamma_k, a_k) : k \geq 1) \tag{4.11}$$

where $1 \leq a_k \leq Q_k - 2$, such that in code φ_{k*},

(a) F^k over $\{a_k\}$ is identified with F^{k+1};
(b) F^k controls address a_k for φ_{k*} via γ_k;

Such a system of fields F^k will be called a *primitive shared field* for the hierarchical code (φ_{k*}). If also each code φ_{k*}, broadcasts F^{k+1} into F^k then we will say that the fields F^k form a *broadcast field*. Note that the field still controls only a single address a_k. The *Main_bit* field mentioned in Subsection 3.1 would be an example.

Proposition 4.4. *For any hierarchical code with primitive shared field given as in* (4.11) *above, for all possible values $u_1 \in \mathbb{S}_1.F^1$ the infinite sequence $(s_k, a_k)_{k\geq 1}$ with $s_k = \gamma_k(u_1)$ is fitted.*

The proof is immediate from the definitions.

Let us denote the configurations ξ^1, ξ^1_k that belong to this fitted infinite sequence according to Proposition 4.2 (and its proof) by

$$\begin{aligned} \xi^1 &= \Gamma(u_1; \Psi) = \Gamma(u_1), \\ \xi^1_k &= \Gamma(u_1; k, \Psi) = \Gamma(u_1; k). \end{aligned} \tag{4.12}$$

4.1.4. *Coding an infinite sequence.* Let us show now how a doubly infinite sequence of symbols can be encoded into an infinite starting configuration.

Let us be given

$$\Psi = (\mathbb{S}_k, Q_k, \varphi_{k*}, F^k, q_k, \gamma_k, a_k)_{k\geq 1} \tag{4.13}$$

where $2 \leq q_k \leq Q_k$ and the sequence a_k is non-degenerate, such that in code φ_{k*},

(a) F^k over $[0, q_k - 1]$ is identified with F^{k+1};

(b) F^k controls address a_k for φ_{k*} via γ_k;

Such a system will be called a *shared field* for the fields F^k, and the hierarchical code (φ_{k*}), and the fields F^k will be denoted as

$$F^k(\Psi).$$

The identification property implies that for all $0 \leq a < q_k$, we have

$$\varphi_{k*}(s)(a).F^k = s.F^{k+1}([a|F^k|, (a+1)|F^k| - 1]). \tag{4.14}$$

Example 4.5. Let us show some examples of codes φ_k in which F^k over $[0, q_k - 1]$ is identified with F^{k+1}.

A code $\psi = (\psi_*, \psi^*)$ with $\psi^* : R^Q \to S$ will be called *d-error-correcting with blocksize* Q if for all u, v, if u differs from $\psi_*(v)$ in at most d symbols then $\psi^*(u) = v$. Assume that both R and S are of the form $\{0,1\}^n$ (for different n). A popular kind of error-correcting code are codes ψ such that ψ_* is a linear mapping when the binary strings in S and R^Q are considered vectors over the field $\{0,1\}$. These codes are called *linear codes*. It is sufficient to consider linear codes ψ which have the property that for all s, the first $|S|$ bits of the codeword $\psi_*(s)$ are identical to s: they are called the *information bits*. (If a linear code is not such, it can always be rearranged to have this property.) In this case, the remaining bits of the codeword are called *error-check bits*, and they are linear functions of s. Applying such linear codes to our case, for $s \in \mathbb{S}_{k+1}$, let

$$w = \varphi_{k*}(s).$$

Then we will have

$$\sqcup_{0\leq a<Q_k} w(a).F^k = \psi_{k*}(s.F^{k+1})$$

for a linear code ψ_k whose information bits are in $\sqcup_{0\leq a<q_k} w(a).F^k$ and error-check bits are in $\sqcup_{q_k\leq a<Q_k} w(a).F^k$. For $d = 1$, if we are not trying to minimize the amount of redundancy in the error correction then we may want to use the tripling method outlined in Subsection 3.5 and Example 4.3, which sets $q_k = Q_k/3$. In this case, the error-check bits simply repeat the original bits twice more. ♢

Example 4.6. In a digression that can be skipped at first reading, let us define the more sophisticated linear code we will be using in later construction (a generalization of the so-called Reed-Solomon code, see [7]).

Let our codewords (information symbols and check symbols together) be binary strings of length Nl for some l, N. Binary strings of length l will be interpreted as elements of the Galois field $GF(2^l)$ and thus, each binary string c of length Nl will be treated as a vector $(c(0), \ldots, c(N-1))$

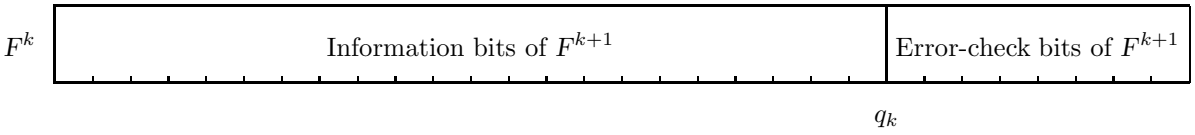

FIGURE 5. Error-correcting code in a shared field

over $GF(2^l)$. (Note that the word "field" is used in two different senses in the present paper.) Let us fix N distinct nonzero elements α_i of $GF(2^l)$ and let $t < N/2$ be an integer. The codewords are those vectors c that satisfy the equation

$$\sum_{i=0}^{N-1} \alpha_i^j c(i).F^k = 0 \ (j = 1, \dots, 2t) \tag{4.15}$$

where the addition, multiplication and taking to power j are performed in the field $GF(2^l)$. These are $2t$ linear equations. If we fix the first $N - 2t$ elements of the vector in any way, (these are the information symbols) the remaining $2t$ elements (the error check symbols) can be chosen in a way to satisfy the equations, by solving a set of $2t$ linear equations. This set of equations is always solvable, since its determinant is a Vandermonde determinant.

Below, we will show a procedure for correcting any $\nu \le t$ nonzero errors. This shows that for the correction of error in any $\le t$ symbols, only $2t$ error-check symbols are needed.

If $E = (e_0, \dots, e_{N-1})$ is the sequence of errors then the word that will be observed is $C + E$. Only e_{i_r} are nonzero for $r = 1, \dots, \nu$. Let $Y_r = e_{i_r}, X_r = \alpha_{i_r}$. Then we define the *syndrome* S_j for $j = 1, \dots, 2t$ by

$$S_j = \sum_i (c_i + e_i)\alpha_i^j = \sum_i e_i \alpha_i^j = \sum_r Y_r X_r^j \tag{4.16}$$

which can clearly be computed from the codeword: it is the amount by which the codeword violates the j-th error check equation. We will show, using the last expression, that Y_r and X_r can be determined using S_j. We first define the auxiliary polynomial

$$\Lambda(x) = \prod_r (1 - xX_r) = \sum_{s=0}^{\nu} \Lambda_s x^s$$

whose roots are X_r^{-1}. Let us show how to find the coefficients Λ_s for $s > 0$. We have, for any $r = 1, \dots, \nu$, and any $j = 1, \dots, 2t - \nu$:

$$0 = Y_r X_r^{j+\nu} \Lambda(X_r^{-1}) = \sum_s \Lambda_s Y_r X_r^{j+\nu-s}.$$

Hence, summing for r,

$$0 = \sum_{s=0}^{\nu} \Lambda_s (\sum_r Y_r X_r^{j+\nu-s}) = \sum_{s=0}^{\nu} \Lambda_s S_{j+\nu-s} \ (j = 1, \dots, 2t - \nu) \tag{4.17}$$

hence using $\Lambda_0 = 1$, $\sum_{s=1}^{\nu} \Lambda_s S_{j+\nu-s} = -S_{j+\nu}$. This is a system of linear equations for Λ_s whose coefficients are the syndroms, known to us, and whose matrix M_ν is nonsingular. Indeed, $M_\nu = ABA^T$ where B is the diagonal matrix $\langle Y_r X_r \rangle$ and A is the Vandermonde matrix $A_{j,r} = X_r^{j-1}$.

A decoding algorithm now works as follows. For $\nu = 1, 2, \ldots, t$, see if M is nonsingular, then compute $\Lambda(x)$ and find its roots by simple trial-and-error, computing $\Lambda(\alpha i^{-1})$ for all i. Then, find Y_r by solving the set of equations (4.16) and see if the resulting corrected vector C satisfies (4.15). If yes, stop.

(There is also a faster way for determining $\Lambda(x)$, via the Euclidean algorithm, see [7]).

To make the code completely constructive we must find an effective representation of the field operations of $GF(2^l)$. This finite field can be efficiently represented as the set of remainders with respect to an irreducible polynomial of degree l over $GF(2)$, so what is needed is a way to generate large irreducible polynomials. Now, it follows from the theory of fields that

$$x^{2 \cdot 3^s} + x^{3^s} + 1$$

is irreducible over $GF(2)$ for any s. So, the scheme works for all l of the form $2 \cdot 3^s$. ♢

In a hierarchical code, with a shared field, there is a function $X(y)$ with the property that site y of the original information will map to site $X(y)$ in the code. To define this function, let

$$B'_k, o'_k, C'_k(y)$$

be defined like $B_k, o_k, C_k(y)$ but using q_k in place of Q_k. For all k, every integer y can be represented uniquely in the form

$$y = \sum_{i=1}^{k} (y'_i - a_i) B'_i \tag{4.18}$$

where $0 \le y'_i < q_i$ for $i < k$. Since (a_k) is non-degerate for (q_k), this is true even with $k = \infty$, in which case the above sum is finite. Let

$$\begin{aligned} X(y,i;k) = X(y,i;k,\Psi) &= \sum_{m=i}^{k} (y'_m - a_m) B_m, \\ X(y,i) &= X(y,i;K(N)), \\ X(y;k) &= X(y,1;k), \\ X(y) &= X(y,1;K(N)), \end{aligned} \tag{4.19}$$

Define the same notation for X' with B'_k instead of B_k. Notice that $X(0,i) = X'(0,i) = 0$, $X'(y,1) = y$. Clearly, the sites of form $o_i + X(y,i;k)$ for all possible y will form a lattice of distance B_i. If $i < k$ then the definitions give

$$o_i + X(y,i;k) = o_{i+1} + X(y,i+1;k) + y'_i B_i. \tag{4.20}$$

Using the notation ι_Q introduced in 3.2.1, let us define the aggregated configurations

$$\varrho^k = \iota_{B'_k *}(\varrho) \tag{4.21}$$

of body size B'_k over $\mathbb{Z}$. Then, of course,

$$\varrho^{k+1} = \iota_{q_k *}(\varrho^k). \tag{4.22}$$

Let

$$\begin{aligned} \mathit{Visible}(k,N) &= \{\, y : 0 \le y - o_K < N B'_K / B_K \,\}. \\ &= \bigcup_{0 \le z B_k < N} C'_k(z) \end{aligned} \tag{4.23}$$

Then $X(y)$ is defined whenever $y \in \mathit{Visible}(k, N)$, i.e. the symbols $\varrho(y)$ can be recovered after encoding whenever y is in this interval.

Proposition 4.7. *For a hierarchical code with a shared field as given in* (4.13)*, for an arbitrary configuration $\varrho = \varrho^1$ in $(\mathbb{S}_1.F^1)^{\mathbb{Z}}$, there are configurations ξ^k over $\mathbb{Z}$ such that for all $k \geq 1$, $y \in \mathbb{Z}$ we have*

$$\varphi_{k*}(\xi^{k+1}) = \xi^k, \tag{4.24}$$

$$\xi^1(X(y)).F^1 = \varrho^1(y). \tag{4.25}$$

More generally, we have

$$\xi^i(o_i + X(y,i)).F^i = \varrho^i(o'_i + X'(y,i)) \tag{4.26}$$

for $1 \leq i \leq k$. If the space is $\mathbb{Z}_N$ for a finite N then all these configurations with the above properties exist for all k with $B_k \leq N$, and (4.25) *holds whenever $y \in \mathit{Visible}(K(N), N)$.*

Proof. The proof is mechanical verification: we reproduce it here only to help the reader check the formalism.

1. Let us construct ξ^k.

   For infinite space size, let

   $$\xi^k_k(C_k(y)) = \gamma_k(\varrho^k(C'_k(y))). \tag{4.27}$$

   For finite space size N, define the above only for $B_k \leq N$ and y in $[0, \lfloor N/B_k \rfloor - 1]$, where $C_k(y)$ on the left-hand side is taken $\pmod N$. In all other sites x, let $\xi^k_k(x) = \mathit{Vac}$. Let ξ^i_k be defined again by (4.9). We define ξ^i as in the proof of Proposition 4.2. It is sufficient to show (4.10) again to prove that the limits in question exist. By definition,

   $$\xi^k_{k+1}(o_k) = \varphi_{k*}(\xi^{k+1}_{k+1}(o_{k+1}))(a_k) = \varphi_{k*}(\gamma_{k+1}(\varrho^{k+1}(o'_{k+1})))(a_k).$$

   By the controlling property, its F^k field r completely determines the last expression via γ_k. By the identification property (4.14) and the aggregation property (4.22),

   $$r = \varrho^{k+1}(o'_{k+1})([a_k B'_k, (a_k+1)B'_k - 1]) = \varrho^k(o'_{k+1} + a_k B'_k) = \varrho^k(o'_k).$$

   By definition, $\xi^k_k(o_k) = \gamma_k(\varrho^k(o'_k))$ which proves the statement.

2. Let us show (4.26).

   Proof. We use induction on i, from k down to 1. The case $i = k$ says

   $$\xi^k(C_k(y'_k - a_k)).F^k = \varrho^k(C'_k(y'_k - a_k))$$

   which follows from the definition of ξ^k. Assume that the statement was proved for numbers $> i$: we prove it for i. By the definitions of $X(y,i)$ and ξ^i and by (4.20) we have

   $$\begin{aligned}\xi^i(o_i + X(y,i)) &= \xi^i(o_{i+1} + X(y,i+1) + y'_i B_i) \\ &= \varphi_{i*}(\xi^{i+1}(o_{i+1} + X(y,i+1)))(y'_i) = \Delta_1.\end{aligned} \tag{4.28}$$

   By induction,

   $$\xi^{i+1}(o_{i+1} + X(y,i+1)).F^{i+1} = \varrho^{i+1}(o'_{i+1} + X'(y,i+1)) = \varrho^{i+1}(z)$$

   where $z = o'_{i+1} + X'(y,i+1)$ can also be written in the form $C'_{i+1}(x)$ for some x. Now we have, by the identification property (4.14) and the aggregation property (4.22)

   $$\Delta_1.F^i = \varrho^i(z + y'_i B'_i) = \varrho^i(o'_{i+1} + X'(y,i+1) + y'_i B'_i) = \varrho^i(o'_i + X'(y,i)) \tag{4.29}$$

   where the third equality is implied by the definition of $X'(y,i)$.

□

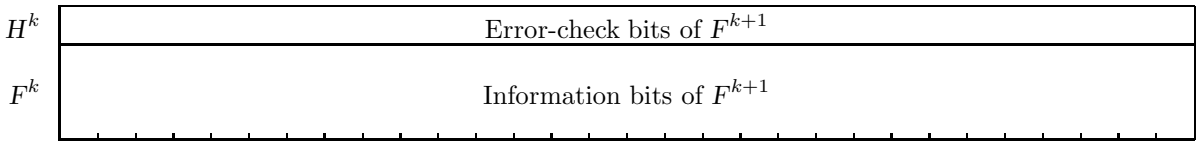

FIGURE 6. Error-correcting code in a shared field, with at least one information bit per cell

In analogy with (4.12), we will denote this code as follows:

$$\begin{aligned} \xi_k^1 &= \Gamma(\varrho; k, \Psi) = \Gamma(\varrho; k), \\ \xi^1 &= \Gamma(\varrho; \infty, \Psi) = \Gamma(\varrho; \Psi) = \Gamma(\varrho). \end{aligned} \tag{4.30}$$

Γ is the *limit code* with *approximations* $\Gamma(\cdot; k)$ and the function $X(y)$ is the *site map* of the system Ψ. Note that for finite space size N, we have $\Gamma(\varrho) = \Gamma(\varrho; k)$ for the largest k with $B_k \le N$.

The proof also shows that $X(y; k)$ plays the role of the site map for the approximation:

$$\xi_k^1(X(y; k)).F^1 = \varrho^1(y), \tag{4.31}$$

The growth of the quotients $X(y)/y$ is a measure of how the infinite code stretches its input, i.e. of the "space redundancy". (Strictly speaking, the redundancy is defined as $X(y)/y - 1$.) The value $X(y; k)/y$ for each approximating code is limited since it stretches blocks of size B'_k into blocks of size B_k. If a code has $q_k = Q_k$ as in the example below then $X(y) = y$.

Example 4.8. Let us show a variant of Example 4.5 with $q_k = Q_k$. The details can be skipped at first reading.

Field F^k is a binary string of length $l_k = l_1 Q_1 \cdots Q_{k-1}$. Let $s \in S_{k+1}$,

$$\begin{aligned} w &= \varphi_{k*}(s), \\ v(a) &= w(a).F^k \ (a = 1, \dots, Q_k - 1). \end{aligned}$$

The information symbols of the code are $v(a)$ for $0 < a < Q_k - 1$. Let the positive integers m_k, n_k be such that

$$m_k h_k = l_k,\ m_k n_k \le Q_k.$$

A narrower track $w.H^k$ contains the error-check bits for the same information, where $|H^k| = h_k$ (see Figure 6). For each $0 \le i < n_k$, the concatenation of strings $w(a).H^k$ with $a \in [im_k, (i+1)m_k - 1]$ will be denoted

$$v(i + Q_k).$$

These are the error-check symbols. With the code of Example 4.6, we have $n_k = 2t$ and $Q_k + n_k = N$. The redundancy of the code is $1/m_k$. ♢

Proposition 4.9. *In the example above,* $\sum_k 1/m_k < \infty$.

Proof. For the capacity of the cells in M_k we have

$$\|\mathbb{S}_k\| = l_k + h_k + r_k = l_k(1 + 1/m_k) + r_k$$

where r_k is the number of bits not belonging to F^k or H^k. The state of a cell of M_{k+1} must be stored in the fields of the cells of the colony representing it, excluding the error-correcting bits in H^k. Hence

$$\begin{aligned} l_{k+1} + h_{k+1} + r_{k+1} &\le Q_k(l_k + r_k), \\ h_{k+1} &\le Q_k r_k - r_{k+1}, \\ 1/m_{k+1} &\le r_k/l_k - r_{k+1}/l_{k+1}, \\ \sum_{k=2}^{\infty} 1/m_k &\le r_1/l_1. \end{aligned} \tag{4.32}$$

□

4.1.5. *Infinitely many fields.* The above construction will be used mainly to encode the input ϱ^1 into the configuration ξ^1, and to find the sites where the output can be retrieved. The information is kept on each level k in field F^k. In the case when besides information storage also computation will be going on, several configurations may have to be encoded, representing e.g. the output of the same computation at different times (see Subsection 6.5).

Here we will set up the framework for coding infinitely many sequences, each to it own track. Since any infinite sequence can be broken up into infinitely many infinite subsequences this elaboration is routine, but it is worth fixing some notation. Readers interested only in information conservation can skip this construction.

Let us be given, for k=1,2, . . . , $0 \le i < k$,

$$\Psi = (\mathbb{S}_k, Q_k, \varphi_{k*}, (F_j^k)_{j=1}^k, q_k, p_k, \gamma_k)_{k\ge 1} \tag{4.33}$$

such that $2 < q_k + p_k \le Q_k$, and in code φ_{k*} the following properties hold:

(a) For each k, field $\bigcup_{j\le k} F_j^k$ controls a_k via γ_k;
(b) F_j^k is identified with F_j^{k+1} over $[0, q_k + p_k - 1]$ if $j < k$ and over $[0, q_k - 1]$ if $j = k$;
(c) F_k^k over $q_k + [0, p_k - 1]$ is identified with F_{k+1}^{k+1};

Such a system will be called a *standard system of shared fields* and we will write

$$F_j^k = F_j^k(\Psi).$$

For simplicity, we only consider infinite space.

Proposition 4.10. *For a standard system of shared fields Ψ as in* (4.33)*, there are functions $X_j(y, \Psi) = X_j(y)$ with $X_j(0) = 0$ such that for any infinite sequence d_j of integers and any infinite sequence of configurations*

$$\varrho_j \in (\mathbb{S}_1.F_1^1)^{\mathbb{Z}} \text{ for } j \ge 1$$

there are configurations ξ^k such that for all $k, j \ge 1$, $y \in \mathbb{Z}$ we have

$$\begin{aligned} \varphi_{k*}(\xi^{k+1}) &= \xi^k, \\ \xi^1(X_j(y) + d_j B_j).F_1^1 &= \varrho_j(y). \end{aligned} \tag{4.34}$$

The proof of this proposition is routine computation, so we omit it here.

The sequence d_j gives additional freedom of shifting the origins independently for each j.

In analogy with (4.12) we define

$$\begin{aligned} \xi^1 &= \Gamma((\varrho_j);\Psi) = \Gamma((\varrho_j)),\\ \xi^1_k &= \Gamma((\varrho_j);k,\Psi) = \Gamma((\varrho_j);k). \end{aligned} \tag{4.35}$$

and call $\Gamma()$ the *limit code* and $X_j(y)$ the *site map* of this many-field hierarchical code. We may want to use the same configuration ϱ_1 for each ϱ_j: e.g. if we start the computation on infinitely many levels simultaneously, from the same input.

4.2. **The active level.** In the (preliminary) definition of error-correction in Subsection 3.6, we used a code $\varphi_{**} \preceq \varphi_*$. Let us discuss the typical structure of the codes φ_{k**} belonging to a hierarchical code.

If φ_{k**} coincides with φ_{k*} over the field F^k, i.e. for all k, s, a we have $\varphi_{k**}(s)(a).F^k = \varphi_{k*}(s)(a).F^k$ then we will say that (F^k) are *broadcast fields* resp. *shared fields* with respect to φ_{k**}, too, whenever they are such in the code φ_{k*}. Next, we give a slight refinement of this notion for the case of reliable computation.

The definitions given here are only needed if the cellular automata are also meant to be used for computation: they are not strictly needed if the goal is only information storage. However, it will be convenient to use them even in that case.

For a shared field, essentially the same space can be used to store the track F^k as the one used for F^{k+1} since the information on the two tracks is the same. Therefore these fields cannot be used by the different levels independently of each other for computation. The mechanism enforcing this is the error-correction property which restricts the value of F^k by the value of F^{k+1} (with which it is identified). Thus, changing the information in track F^k we change it on all levels below. We should therefore know which level is the "active" one, the one being changed directly rather than as a consequence of the code constraints. This level should not be disturbed by error-correction coming from higher levels.

The active level will be marked in the following way. Let us be given, for $k = 1, 2, \ldots$, new fields G^k with $|G^k| = 2$. The four possible values of G^k will be identified with $-1, 0, 1, *$. We will say that (F^k, G^k) define a sequence of *guarded shared fields* if the following properties hold:

(a) For all $0 \le a < Q$,

$$\varphi_{k**}(s)(a).G^k = \begin{cases} -1 & \text{if } s.G^{k+1} \le 0,\\ * & \text{otherwise;} \end{cases}$$

(b)

$$\varphi_{k**}(s)([0, q_k - 1]).F^k = \begin{cases} s.F^{k+1} & \text{if } s.G^{k+1} \le 0,\\ * \cdots * & \text{otherwise.} \end{cases} \tag{4.36}$$

Notice that since $\varphi_{k**} \preceq \varphi_{k*}$, this also imposes some conditions on (φ_{k*}).

Typically, for a certain k we will have $\xi^i(\cdot).G^i > 0$ for all $i > k$, $\xi^k(\cdot).G^k = 0$, and $\xi^i(\cdot).G^i < 0$ for $i < k$. This distinguished k will show the "active" level, on which the field F^k can be changed: G^k shows whether we are on, below or above the active level. The fact $\xi^{k+1}(\cdot).G^{k+1} > 0$ will imply that the level $(k+1)$ does not restrict us from changing $\xi^k(x).F^k$ (e.g. by computation). The levels below the active one are the ones subject to error-correction coming from F^k since the properties imply that the F^i for all $i < k$ behaves like a shared field in code (φ_{i**}) just as it does in code (φ_{i*}). The guard field G^{k+1} is broadcast whenever it is negative.

With a guarded shared field, if the active level is k then the definition of $\Gamma(u_1; k) = \xi^1_k$ in (4.12) will always imply the additional property

$$\xi^k_k(x).G^k = 0 \tag{4.37}$$

for all x. Subsequent codings by $\varphi_{(k-1)*}$, etc. imply $\xi^i_k.G^i = -1$ for $i < k$. The active level ∞ will mean $\xi^k_k(x).G^k = -1$ for all k. All propositions in the present section about the existence of encoded configurations, can be enhanced to include a guard field, with an arbitrarily chosen active level (possibly ∞).

Remarks 4.11.
1. All the tracks F^k for different k "can use the same space" since G^{k+1} has the information showing the way F^k depends on F^{k+1}. However, each track G^k must be represented in new space since G^k is not identified with G^{k+1}.
2. The only place in the present paper where the possibility of changing the G^k field is exploited is 5.2.1.

♢

4.3. **Major difficulties.** The idea of a simulation between two perturbed cellular automata is, unfortunately, flawed in the original form: the mapping defined in the naive way is not a simulation in the strict sense we need. The problem is that a group of failures can destroy not only the information but also the organization into colonies in the area where it occurs. This kind of event cannot therefore be mapped by the simulation into a transient fault unless destroyed colonies "magically recover". The recovery is not trivial since "destruction" can also mean replacement with something else that looks locally as legitimate as healthy neighbor colonies but is incompatible with them. One is reminded of the biological phenomena of viruses and cancer. Rather than give up hope let us examine the different kinds of disruption that the faults can cause in a block simulation by a perturbed cellular automaton M_1.

Let us take as our model the informally described automaton of Subsection 3.5. The information in the current state of a colony can be divided into the following parts:

- "information": an example is the content of the *Info* track.
- "structure": the *Addr* and *Age* tracks.
- "program": the *Prog* track.

More informally, the "structure" does not represent any data for the decoding but is needed for coordinating cooperation of the colony members. The "program" determines which transition function will be simulated. The "information" determines the state of the simulated cell: it is the "stuff" that the colony processes.

Disruptions are of the following kinds (or a combination of these):

(1) Local change in the "information";
(2) Locally recognizable change in the "structure";
(3) Program change;
(4) Locally unrecognizable change in "structure";

A locally recognizable structure change would be a change in the address field. A locally unrecognizable change would be to erase two neighbor colonies based, say, at BQ and $2BQ$ and to put a new colony in the middle of the gap of size $2BQ$ obtained this way, at position $1.5BQ$. Cells within both the new colony and the remaining old colonies will be locally consistent with their neighbors; on the boundary, the cells have no way of deciding whether they belong to a new (and wrong) colony or an old (and correct) one.

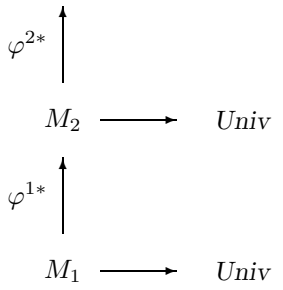

FIGURE 7. An amplifier in which the simulations φ^{k*} are "hard-wired". Universality is not lost since each medium M_k also simulates some universal cellular automaton.

The only kind of disruption whose correction can be attempted along the lines of traditional error-correcting codes and repetition is the first one: a way of its correction was indicated in Subsection 3.5. The three other kinds are new and we will deal with them in different ways.

To fight locally recognizable changes in the structure, we will use the method of destruction and rebuilding. Cells that find themselves in structural conflict with their neighbors will become vacant. Vacant cells will eventually be restored if this can be done in a way structurally consistent with their neighbors.

To fight program changes, our solution will be that the simulation will not use any "program" or, in other words, "hard-wire" the program into the transition function of each cell. We will not lose universality this way: the automata will still be universal, i.e. capable of simulating every other automaton by appropriate block simulation; but this particular simulation will differ from the others in that the transition function will perform it without looking at any program.

To fight locally unrecognizable changes, we will "legalize" all the structures brought about this way. Consider the example where a single colony sits in a gap of size $2BQ$. The decoding function is defined even for this configuration. In the decoded configuration, the cell based at site 0 is followed by a cell at site $1.5BQ$ which is followed by cells at sites $3BQ, 4BQ$, etc. Earlier, we did not have any use for these illegal configurations. We must legalize them now. Indeed, since they can be eliminated only with their own active participation, we must have rules (trajectory conditions) applying to them. This is the real reason for the introduction of generalized cellular automata.

The generalized cellular automaton actually needed will be called a *robust medium*. The generalization of the notion of the medium does not weaken the original theorem: the fault-tolerant cellular automaton that we eventually build is a cellular automaton in the old sense. The more general media are only needed to make rules for all the structures that arise in simulations by a random process.

5. Main theorems in discrete time

Some theorems starting with the present section will have names of the form "(FCA,[description])" where [description] gives a shorthand characterization of the properties of the automaton constructed.

5.1. **Relaxation time and ergodicity.**

5.1.1. *Ergodicity.* Let

$$\mathbb{S}_n = \mathbb{S}^{([-n,n]\cap\mathbb{Z})^d}$$

be the set of configurations on the segment $[-n,n]\cap\mathbb{Z}$. Then we can view $\mathbf{s}\in\mathbb{S}_n$ as the vector $(s_{-n},\dots,s_n)$. For a measure ν over configurations, let

$$\nu(\mathbf{s}) = \nu\{\xi(i) = s_i,\ i = -n,\dots,n\}. \tag{5.1}$$

For $n=0$, we have the special case $\nu(s) = \nu\{\xi(0)=s\}$. A sequence ν_k of measures *weakly converges* to measure ν if for for all n, for each $\mathbf{s}\in\mathbb{S}_n$ we have $\lim_k \nu_k(\mathbf{s}) = \nu(\mathbf{s})$.

If (μ,η) is a random trajectory of a probabilistic cellular automaton then let μ^t be the distribution of the configuration $\eta(\cdot,t)$. Then there is a linear operator P (called the *Markov operator*) determined by the transition function $\mathbf{P}(s,\mathbf{r})$ such that $\mu^{t+1} = P\mu^t$. To show this it is enough to show how $\mu^1(\mathbf{s})$ is determined by μ^0. According to the definition of a trajectory, we have

$$\mu^1(\mathbf{s}) = \sum_{\mathbf{r}}\prod_{j=-n}^{n}\mathbf{P}(s_j,(r_{j-1},r_j,r_{j+1}))\mu^0(\mathbf{r}) \tag{5.2}$$

where the summation goes over all possible strings $\mathbf{r}\in\mathbb{S}_{n+1}$.

We call a measure α over configurations *invariant* if $P\alpha=\alpha$. It is well-known and easy to prove using standard tools (see a reference in [27]) that each continuous linear operator over probability measures has an invariant measure. The invariant measures describe the possible limits (in any reasonable sense) of the distributions μ^t. A probabilistic cellular automaton is called *ergodic* if it has only one invariant measure. It is called *mixing* if also for every possible measure μ^0 over configurations, $\mu^t = P^t\mu^0$ converges to one and the same invariant measure. Intuitively, if a process is mixing then the distributions μ^t will look more and more like the invariant measure and contain less and less information about the initial distribution μ^0. In other words, all information about the initial configuration will be eventually lost.

Remark 5.1. Consider a discrete-time Markov process over a compact space. For an initial distribution μ, let P_μ be the measure for the whole process. Let us define the translation operators: $(T'\eta)(x,t) = \eta(x,t+1)$ for space-time configurations, $(Tg)(\eta) = g(T'\eta)$ for functions over space-time configurations. Traditionally, the process with initial distribution μ is called *mixing* if for each time t, each pair of continuous functions f,g where f is defined over the $<t$ sigma-algebra, we have

$$\int fT^s g dP_\mu - \int f dP_\mu \int T^s g dP_\mu \to 0.$$

as $s\to\infty$. It can be seen that a probabilistic cellular automaton is mixing in our sense if each process obtained from it by choosing some initial distribution is mixing in this traditional sense. ♢

A noisy cellular automaton, whenever the set of sites is finite, is a finite Markov chain with all positive transition probabilities. This is mixing by a well-known elementary theorem of probability theory. If the set of sites is infinite then noisiness does not imply even ergodicity.

Remark 5.2. No examples are known of noisy cellular automata over an infinite space that are ergodic and not mixing. ♢

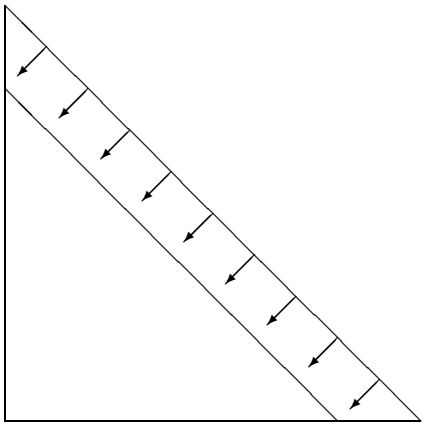

FIGURE 8. The Toom rule's effect on a large triangular island

The first example of a non-ergodic noisy cellular automaton was given by Toom. (See e.g. [27].) One of the simplest deterministic cellular automata R given by Toom can be defined as follows. We start from a two-dimensional deterministic cellular automaton R with set of states $\{0, 1\}$, by the neighborhood $H = \{(0, 0), (0, 1), (1, 0)\}$. The transition function $Tr_R(x_1, x_2, x_3)$ is the majority of x_1, x_2, x_3. Thus, in a trajectory of R, to obtain the next state of a cell, take the majority among the states of its northern and eastern neighbors and itself. Toom showed that the rule R remembers a field (namely the whole state of the cell) in the sense defined in Subsection 2.5 and is hence non-ergodic.

Remark 5.3. The notion of ergodicity was defined only for trajectories of R_ε that are also trajectories of some probabilistic cellular automaton R' (as defined in Subsection 2.3) such that $R' \subset R_\varepsilon$, i.e. that all trajectories of R' are trajectories of R_ε. The difference between R_ε and R' is that the local transition probabilities of a trajectory of R' are fixed and the same everywhere in space-time, while R_ε requires them only to be within some range. Toom's theorem implies that no such R' is ergodic. ♢

5.1.2. *Relaxation time: a measure of information loss.* Some readers, computer scientists in particular, may want to know already now what relevance can results on inifinite cellular automata have on have on the possibilities of computation or information storage in finite systems. We will try to answer this question here. To stay in the context of the Toom rule, let us consider a (finite or infinite) two-dimensional space $\mathbf{C}$. With an extension of the notation (5.1) to two dimensions (where $\mathbf{s}$ is now the array $(s_{ij} : -n \le i, j \le n)$), for any fixed $\mathbf{s}$, let us define the variation distance for μ and ν when the latter are restricted to $\mathbb{S}_n$:

$$d^n(\mu, \nu) = \sum \{\, |\mu(\mathbf{s}) - \nu(\mathbf{s})| : \mathbf{s} \in \mathbb{S}_n \,\}.$$

Of course, $0 \le d_n(\mu, \nu) \le 2$. Distance 2 means that the two measures have disjoint support. Consider a set of sites

$$\mathbf{C}_m = \mathbb{Z}_m \times \mathbb{Z}_m$$

where m can be finite or infinite. Suppose that for some m, the local transition matrix (for simplicity, with nearest-neighbor interaction) gives rise to a mixing Markov process with Markov operator P_m.

Let

$$D^n_m(t) = \bigvee_{\mu,\nu} d^n_m(P^t_m\mu, P^t_m\nu).$$

Remark 5.4. It is easy to see that this is equal to

$$\bigvee_{\mathbf{r},\mathbf{s}} d^n_m(P^t_m\nu_{\mathbf{r}}, \nu_{\mathbf{s}})$$

where $\nu_{\mathbf{s}}(\mathbf{s}) = 1$, i.e. $\nu_{\mathbf{s}}$ is the measure concentrated on configuration $\mathbf{s}$. ♢

Notice that $D^n_m(t)$ is monotonically decreasing in t since

$$d^n_m(P^{t+u}_m\nu, \mu_m) = d^n_m(P^t_m(P^u_m\nu), \mu_m).$$

P_m is mixing if and only if for each $\mathbf{s}, \nu$ we have $P^t_m\nu(\mathbf{s}) \to \mu_m(\mathbf{s})$. By the weak compactness of the space of measures, this is equivalent to saying that

$$\lim_t D^n_m(t) = 0$$

holds for all n. The mixing of P_m implies that there is an integer function

$$r_m(n, \delta)$$

with $D^n_\infty(t) < \delta$ for all $t \geq r_m(n, \delta)$. We will call $r_m(n, \delta)$ the *relaxation time* . This is obviously an increasing function of n (defined only for $n \leq (m-1)/2$) and a decreasing function of δ. In the cases we are interested in, the order of magnitude of $r_m(n, \delta)$ as a function of n does not change fast with the change of δ: $r_m(n, 0.1))$ is not much larger than $r_m(n, 1.9)$. This means that once μ_m is not separated well from any $P^t_m\nu$ it soon becomes fairly close to all of them. Therefore we will generally consider δ fixed.

5.1.3. *Relaxation time as a function of space size.* If $m < \infty$ then the medium is always mixing. Assume now that the medium is also mixing for $m = \infty$: we try to understand the implications of this fact for finite m. We have the following relation:

Lemma 5.5. *For all $n < (m-1)/2$, for all δ with $r_\infty(n, \delta) < (m-1)/2 - n$ we have*

$$r_m(n, \delta) \leq r_\infty(n, \delta).$$

This means that if the medium is mixing for $m = \infty$ then increasing m in the case of $m < \infty$ does not increase the relaxation time significantly for any fixed n: in each segment of length n of any finite medium, information is being lost at least as fast as in the infinite medium.

Proof. Take m, n, δ satisfying the above conditions and let $r = r_\infty(n, \delta)$. Due to the monotonicity of $D^n_m(t)$, it is enough to prove that $D^n_m(r) \leq D^n_\infty(r)$. Take a measure ν over configurations of $\mathbf{C}_m$, this will give rise to some measure ν_∞ over configurations of period m in $\mathbf{C}_\infty$ in a natural way, where ν_∞ is such that for all $n < (m-1)/2$ and all $\mathbf{s} \in \mathbb{S}_n$ we have $\nu_\infty(\mathbf{s}) = \nu(\mathbf{s})$. Then, $r < (m-1)/2 - n$ implies $2n + 1 + 2r < m$ and therefore via (5.2) we have

$$P^r_m\nu(\mathbf{s}) = P^r_\infty\nu_\infty(\mathbf{s}).$$

□

We found that mixing of P_∞ implies a kind of *uniform forgetfulness* in the sense that increasing the size m of the space does not help increasing the relaxation time beyond $r_\infty(n, \delta)$.

5.1.4. *Forgetfulness: a variant of ergodicity.* Non-ergodicity does not express quite adequately the losing of all information about the initial configuration in case of cellular automata, where namely the space-time is translation-symmetric (this was noticed by Charles Radin and Andrei Toom).

Let ξ_0 be the configuration over the one-dimensional integer lattice that is 0 in the even places and 1 in odd places, and ξ_1 that is 1 in the even places and 0 in the odd places. Let μ_i be the measure concentrated on ξ_i and let P be some linear operator obtained from a transition function. Suppose that the measures $P^n\mu_0$ converge to some measure ν_0. Then the measures $P^n\mu_1$ converge to some measure ν_1. Even if ν_1 is different from ν_0 they differ only by a translation in space. If the translations of ν_0 are the only invariant measures of P then we would still say that in some sense, P loses all information about the initial configuration: we might say, it loses all "local" information. Indeed, a cell has no way of knowing whether it has even or odd coordinates.

We can say that P is *strongly not forgetful* if it has two disjoint (weakly) closed translation-invariant sets of measures. Our non-ergodic examples all have this property.

5.2. **Information storage and computation in various dimensions.** Let us be given an arbitrary one-dimensional transition function Tr and the integers N, L. We define the three-dimensional transition function Tr' as follows. The interaction neighborhood is $H \times \{-1, 0, 1\}$ with the neighborhood H defined in Subsection 5.1 above. The transition function Tr' says: in order to obtain your state at time $t + 1$, first apply the transition function R in each plane defined by fixing the third coordinate. Then, apply transition function Tr on each line obtained by fixing the first and second coordinates. (The papers [14] and [11] use a neighborhood instead of H that makes some proofs easier: the northern, south-eastern and south-western neighbors.)

For an integer m, we define the space $\mathbf{C} = \mathbb{Z}_N \times \mathbb{Z}_m^2$. For a trajectory ξ of $CA(Tr)$ on $\mathbb{Z}_N$, we define the trajectory ζ of $CA(Tr')$ on $\mathbf{C}$ by

$$\zeta(i, j, n, t) = \xi(n, t). \tag{5.3}$$

Thus, each cell of $CA(Tr)$ is repeated m^2 times, on a whole "plane" (a torus for finite m) in $\mathbf{C}$.

Then it is proved in earlier work that there are constants $\varepsilon_0, c_1, d_1 > 0$ such that the following holds. For all N, L, and $m = c_1 \log(NL)$, for any trajectory ξ of $CA(Tr)$ over $\mathbb{Z}_N$, if the trajectory ζ of $CA(Tr')$ is defined by (5.3) then for any $\varepsilon < \varepsilon_0$, any trajectory (μ, η) of $CA_\varepsilon(Tr')$ such that $\eta(\cdot, 0) = \zeta(\cdot, 0)$ we have for all t in $[0, L]$ and all $w \in \mathbf{C}$

$$\mu\{\, \eta(w, t) \neq \zeta(w, t) \,\} \leq d_1\varepsilon.$$

This theorem says that in case of the medium $CA_\varepsilon(Tr')$ and the trajectories (μ, ζ), the probability of deviation can be uniformly bounded by $d_1\varepsilon$. The trajectories η encode (by (5.3)) an arbitrary computation (e.g. a universal Turing machine), hence this theorem asserts the possibility of reliable computation in three-dimensional space. The coding is repetition $O(\log^2(NL))$ times, i.e. it depends on the *size* $N \cdot L$ of the computation. The decoding is even simpler: if a plane of $\mathbf{C}$ represents a state s of a cell of $CA(Tr)$ then each cell in this plane will be in state s with large probability. The simulation occurs in "real time".

The original proof of a slightly weaker version of this result used a sparsity technique borrowed from [10]. In its current form, the theorem was proved in [5] using an adaptation of the techniqe of [27].

Theorem 2.5 shows that one-dimensional noisy and strongly not forgetful probability operators exist. The proof of that theorem seems to require almost the whole complexity of the constructions of the present paper (though the continuous-time case adds some additional nuisance to each part of the proof). Once the basic structure (an amplifier, as asserted in Lemma 3.7) is in place, the simulations in it support arbitrary computational actions and allow the formulation of several other theorems.

The following theorem asserts the possibility of storing much more information. For the finite version, recall the definition of *Visible*(k, N) in (4.23).

Theorem 5.6 (FCA, 1-dim, storage, discrete time)**.** *There is*

- *a transition function* Tr' *with a statespace having a field* F^1*;*
- *a hierarchical code* Ψ *with a shared field* (F^k) *as in* (4.13) *such that* $|Visible(K(N), N)| = N$*;*
- *constants* $d_1, c_2 > 0$*;*

such that for $h_2(n) = n^{c_2/\log\log n}$*, for all* $\varepsilon > 0$*,* $t > 0$*, for any configuration* ϱ *with states in* $\mathbb{S}.F^1$*, trajectory* η *of* $CA_\varepsilon(Tr')$ *over* $\mathbf{C} = \mathbb{Z}_N$ *(for finite or infinite* N*) with initial configuration* $\Gamma(\varrho; \Psi)$*, for all* y *in Visible*$(K(N), N)$*, we have*

$$\text{Prob}\{\, \varrho(y) \neq \eta(X(y; \Psi), t).F^1 \,\} < d_1\varepsilon + t\varepsilon^{h_2(N)}.$$

The code can be chosen to have $X(y; \Psi) = y$*.*

The function $\varepsilon^{h_2(N)}$ shows the length of time for which a space of size N can hold information.

A fault-tolerant computation is the trajectory of some perturbed medium $CA_\varepsilon(Tr')$ simulating the trajectory of an arbitrary deterministic medium. The simulated transition function Tr is arbitrary, but we might as well choose it to be a fixed universal cellular automaton. Then $CA(Tr')$ can be called a "universal fault-tolerant cellular automaton". The exact form of the theorems formulating the possibility of reliable computation is somewhat arbitrary. It may seem more natural if the input of the computation is not given in the initial configuration but is being fed through some cell throughout the time of the computation. There are several choices also concerning simplicity, the tradeoffs of space- and time-redundancy, etc. These theorems should be considered therefore as just illustrations of the possible use of the amplifiers of Lemma 3.7.

The basic idea is to implement a large but finite computation of *Univ* by choosing the "active level" k mentioned above so large that during the planned computation, it is unlikely that there will be any fault at all in the work of medium M_k. If the computation involves N cells and L steps then this means that we should have, say, $NL\varepsilon_k < 1/3$. Theorem 5.8 realizes this idea. In it, there are still several choices: simpler encoding at the price of larger space redundancy, or more complex encoding at the proce of less space redundancy and more time redundancy.

To formulate the theorems exactly, let Tr be any deterministic transition function with distinguished fields *Input*, *Output* in the standard alphabet. We say that Tr has *monotonic output* if for all trajectories η of $CA(Tr)$ we have

$$\eta(x, t).Output \preceq \eta(x, t + 1).Output.$$

We call the transition function Tr, all of whose fields have even size (i.e. they are in the standard alphabet) together with some distinguished fields *Input*, *Output*, a *standard computing transition function* if it

(a) is a separable transition function as defined in 2.2.1;
(b) never changes *Input*;
(c) has monotonic output;
(d) never changes anything if one of the three arguments is vacant or if the middle argument has all #'s in its input field, or if all three arguments have $*$ in all their fields.

Remark 5.7. The *Input* and *Output* fields here have nothing to do with the *Inbuf* and *Outbuf* fields introduced in 2.2.1. These are input and output of a certain large-scale computation, while *Inbuf* and *Outbuf* concern the step-for-step interaction of a cell with its neighbors. ♢

Let $\mathbb{S}$ be the set of states for Tr, and let $\varrho \in (\mathbb{S}.\mathit{Input})^{\mathbb{Z}}$ be an "input configuration". Then the configuration

$$\zeta = \mathit{Init}(\varrho) \tag{5.4}$$

with states in $\mathbb{S}$, is constructed as follows.

(a) For all $x \in \mathbb{Z}$, $\zeta(x).\mathit{Input} = \varrho(x)$ and $\zeta(x).\mathit{Output}$ is filled with $*$;
(b) For all $x \in \mathbb{Z}$, all other fields of $\zeta(x)$ are filled with $*$;
(c) For all $x \notin \mathbb{Z}$, we have $\zeta(x) = \mathit{Vac}$.

If ϱ is a configuration in $\mathbb{S}^{\mathbb{Z}}$ where $\|\mathbb{S}\|$ is even, i.e. elements of $\mathbb{S}$ can be viewed as strings in the standard alphabet, then we say that ϱ is *supported by* interval I if $\varrho(x) = * \cdots *$ or vacant for all $x \notin I$. The *support* of ϱ is defined to be the smallest interval supporting ϱ and containing 0. Let

$$\mathit{Supp}(\varrho) \tag{5.5}$$

be the support of ϱ. For a space-time configuration ζ of a standard computing medium, let

$$\mathit{Supp}(\zeta, t) = \bigcup_{u \le t} \mathit{Supp}(\zeta(\cdot, u)). \tag{5.6}$$

Theorem 5.8 (FCA, 1-dim, discrete time, known bounds)**.** *Let Tr be a standard computing transition function. There is*

- *a transition function Tr' with a statespace having a field Output^1;*
- *constants $d_1, c_0, c_1 > 0$;*
- *a hierarchical code with a shared field defined by Ψ as in* (4.13)

such that for $h_0(t) = \log\log t + c_0$, $h_1(t) = t \cdot (c_1 \log t)^{5 \log\log\log t}$, for all $\varepsilon, s, t > 0$, for any trajectory ζ of Tr over $\mathbb{Z}$ with $\zeta(\cdot, 0)$ of the form $\mathit{Init}(\varrho)$ (see (5.4)*), satisfying $2|\mathit{Supp}(\zeta, t)| \le s$, for any finite $K \ge h_0(t)$ and possibly infinite N satisfying $\mathit{Supp}(\zeta, t) \subset \mathit{Visible}(K, N)$, for any trajectory η of $CA_\varepsilon(\mathrm{Tr}')$ over $\mathbf{C} = \mathbb{Z}_N$ with initial configuration $\Gamma(\varrho; K, \Psi)$, for all $y \in \mathit{Supp}(\zeta, t)$, we have*

$$\mathrm{Prob}\{\, \zeta(y,t).\mathit{Output} \not\preceq \eta(X(y; K, \Psi), h_1(t)).\mathit{Output}^1 \,\} < d_1\varepsilon.$$

The code can be chosen to have $X(y; K, \Psi) = y$ and $|\mathit{Visible}(K, N)| = N$.

Thus, to find a computation result $\zeta(y, u)$ of $CA(Tr)$, from some input, we must do the following. Find a bound s on $2|\mathit{Supp}(\zeta, t)|$ (the amount of space required by the computation if we put the origin somewhere into the input), then the level $K \ge h_0(t)$ on which the trajectory η of the fault-tolerant automaton $CA_\varepsilon(\mathrm{Tr}')$ carries out the computation to the desired stage u with the desired reliability until step t. Embed the input into a configuration ϱ whose support contains 0. Encode ϱ into an initial configuration $\Gamma(\zeta(\cdot, 0); K)$ in a space whose size N is large enough to have $\mathit{Supp}(\zeta, t) \subset \mathit{Visible}(K, N)$. "Run" the trajectory η until time $h_1(t)$, (it will be programmed to keep the active level at K), and finally look up the result in site $X(y; K)$. Due to monotonic output, it is always possible to look up the result later. If you look at it too early the worst thing that can happen is that it will still be undefined.

The function $h_1(t)/t$ measures *time redundancy* while $X(y; h_0(t))/y$ measures *space redundancy* (which, as the theorem says, can be made constant).

The above theorem strengthens the result of [10] by eliminating the need for the decoding of the computation result: we just have to look in the right place for it. In two dimensions such a construction was given in [11]. The time redundancy in that paper is $O(\log^{2+\delta}(NL))$ where N, L are the upper bounds on the space and time requirements of the computation. We believe that this two-dimensional time redundancy can be reduced to $\log^{1+\delta}(NL)$.

5.2.1. *Adaptive encoding.* Theorem 5.8 is not entirely satisfactory since its initial encoding (the active level) depends on the knowledge of a time and space bound on the whole computation. It is possible to remove this limitation. Instead of formulating the theorem, we just describe the idea of the construction. The initial encoding populates the space with cells of M_k for all $k < K(N)$ where N is the space size. Chose a level k_1 of encoding for the computation input depending only on the size of the input. For an appropriately chosen (universal) increasing sequence (L_k), after L_k computation steps have been simulated, the active level will be raised from k to $k+1$. The parameters k_1 and (L_k) are chosen to guarantee with high probability that none of level k cells involved in the actual computation will fail before the level will be raised to $k+1$.

5.2.2. *Self-organization.* In the just-discussed hierarchical fault-tolerant computations, the initial configuration is highly non-uniform since it encodes the hierarchy of simulations described above. If we forgo this initialization then the hierarchy must apparently be built up spontaneously. Initially, all sites in the media $M_2, M_3, \ldots$ could be vacant but cells could emerge spontaneously, as lower-level cells organize themselves (with the help of some randomness) into colonies simulating higher-level cells. We call this feature *self-organization*, but postpone its formal definition a few sections later.

6. Media

An abstract medium has been defined in 3.3 as essentially a set of stochastic processes called its trajectories. Let us narrow this category somewhat. The kind of abstract medium considered in this paper will simply be called a *medium*. We will also restrict the kind of simulations among media to mappings that by their very form will obviously be simulations: they will be called *canonical simulations*.

6.1. **Trajectories.** Here, we will show in what form the set of trajectories of a medium will be given.

6.1.1. *Weak trajectories.* In defining the conditions making up a medium we will make use of inequalities of the form

$$\mathbf{E}g \leq b$$

where g is an event function (as defined in Subsection 2.3), measurable in $\mathcal{A}(V)$ for some rectangle V.

Example 6.1. Let $V = [a_1+, a_2] \times [t+, u]$, $B < d = 0.2(a_2 - a_1)$. Suppose that we are given some $\mathbf{r} \in \mathbb{S}^3$, $s \in \mathbb{S}$. Let $g = 1$ if there is a space-time point (x, t_0) in V, and an $\varepsilon < d$ such that $\eta(x + iB, t) = r_i$ for $i = -1, 0, 1$ during $[t_0 - \varepsilon, t_0-]$ and $\eta(x, t_0) = s$. Thus, the event $g = 1$ marks a certain kind of transition in the window. ♢

A medium will be given by many such conditions. Let us fix an arbitrary countable set $\mathbb{T}$ which will be called the set of *condition types*. A *medium* is given as

$$M = Med(\mathbb{S}, \mathbf{C}, g(\cdot, \cdot, \cdot), b(\cdot)). \tag{6.1}$$

Here, $\mathbb{S}$ is the set of possible local states and $\mathbf{C}$ is the set of possible cell sites. The space $\mathbf{C}$ can also be omitted whenever it is clear from the context. For all types $\alpha \in \mathbb{T}$ and rational rectangles V, the medium assigns an event function

$$g(\alpha, V, \eta)$$

and a bound $b(\alpha) \in [0, 1]$. (We do not insist that $g(\alpha, V, \cdot)$ be measurable in $\mathcal{A}(V)$ though usually this will be the case.) Therefore from now on, a *local condition* will be defined as a pair

$$(\alpha, V).$$

The set $\mathbb{S}$ is determined by g implicitly. Also, generally local conditions will only be defined for small V, and in a translation-invariant way (in space, but only almost so in time since the time 0 is distinguished). Therefore one and the same pair g, b can serve for all sufficiently large finite spaces $\mathbf{C}$ as well as the infinite space, just like with deterministic cellular automata, where the same transition function determines trajectories in finite spaces as well as the infinite space.

When we are given media $M_1, M_2, \ldots$ then $g_i(), b_i()$ automatically refer to M_i.

Two local conditions are *disjoint* when their rectangles are. A random space-time configuration η will be called a *weak trajectory* if for each time $u \geq 0$, for each set of disjoint local conditions

$$(\alpha_i, V_i)_{i \leq N}$$

with $\min_i \pi_{\mathrm{t}} V_i \geq u$, if h is an event function over $\mathcal{A}_u$ then

$$\mathbf{E}(h \prod_i g(\alpha_i, V_i, \eta)) \leq \mathbf{E}h \prod_i b(\alpha_i). \tag{6.2}$$

This says that we can multiply the probabilities of violating local conditions in disjoint windows, i.e. that these violations happen "independently".

6.1.2. *Trajectories.* The media we are going to be interested in have these three types of condition.

(1) One type recognizes in V a certain kind of value called "damage", and has $b = \varepsilon$ for some small ε;
(2) One type recognizes if there is no damage in V and the transition does not occur according to a certain transition function. This has $b = 0$.
(3) One type corresponds to some "coin tossing" event, and has $b = 0.5 + \varepsilon'$ for a suitable ε'.

If we need coin-tossing (we need it only for the results on self-organization) then we must also consider certain sets of conditions that are chosen randomly. A *random local condition* is a (measurable) function $(\alpha, V(\eta))$ assigning a local condition to each space-time configuration η. We recall the definition of stopping times from the theory of stochastic processes, specialized to our present needs. A real random variable σ with values in $[0, \infty-]$ is a *stopping time derived from* η if for each t the event $\sigma \leq t$ is in $\mathcal{A}_t$. Let $\mathcal{A}_\sigma$ be the algebra of events A with the property that $A \cap \{\sigma \leq t\} \in \mathcal{A}_t$. Assume that for each t in $[0, \infty-]$ there is random variable $V^t(\eta)$ measurable in $\mathcal{A}_t$ with values that are rational rectangles with

$$\pi_{\mathrm{t}} V^t(\eta) \subset [t, \infty]$$

and such that with probability 1 the function $V^t(\eta)$ is right semi-continuous in t. This will be called a *rectangle process derived from* η. We mention the following well-known fact from the theory or stochastic processes:

Theorem 6.2 (see [21])**.** *V^σ is a random variable measurable in $\mathcal{A}_\sigma$.*

We will omit the words "derived from η" when they are clear from the context. A *random-stopping local condition* is given by a triple

$$(\alpha, V^t(\eta), \sigma) \tag{6.3}$$

where α is a type, $V^t(\eta)$ is a rectangle process and σ is a stopping time, both derived from η. This condition is supposed to express the bound

$$\mathbf{E} g(\alpha, V^\sigma, \eta) \leq b(\alpha).$$

For some finite set N of indices, let us now be given a sequence

$$(\alpha_i, V_i^t, \sigma_i)_{i \in N} \tag{6.4}$$

of random-stopping local conditions. It is is called a *system of disjoint random local conditions* if (for almost all η) all the rectangles $V_i^{\sigma_i}(\eta)$ are disjoint, and there is a constant upper bound on the σ_i and a constant segment containing all $\pi_{\mathrm{s}} V_i^t$. A random space-time configuration η will be called a *trajectory* if for each set of disjoint random local conditions as in (6.4) with $\sigma_i \geq u$, if h is an event function over $\mathcal{A}_u$ then

$$\mathbf{E}(h \prod_i \bar{g}(i, \eta)) \leq \mathbf{E} h \prod_i b(\alpha_i) \tag{6.5}$$

where

$$\bar{g}(i, \eta) = g\big(\alpha_i, V^{\sigma_i(\eta)}(\eta), \eta\big). \tag{6.6}$$

This also says that we can multiply the probabilities of violating local conditions in disjoint windows, but requires that this be even true if the windows are chosen by random stopping rules.

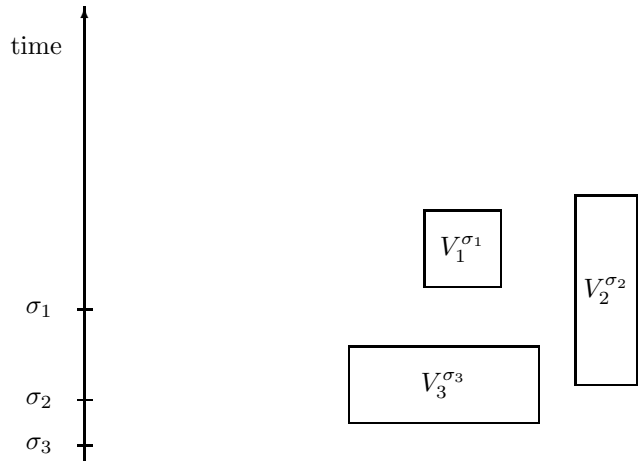

FIGURE 9. A system of disjoint random local conditions

6.1.3. *Examples of simple media.*

Example 6.3. The first example will show how a *PCA* fits into this framework. Let us be given a probabilistic cellular automaton $PCA(\mathbf{P}, B, T)$. For each state vector $\mathbf{r} \in \mathbb{S}^3$ and state s, let us form type $\alpha(s, \mathbf{r})$.This condition type says that the transition rules are obeyed at times of the form nT. Then for cell x, time t of the form nT with $n > 0$, let

$$g\big(\,\alpha(s,\mathbf{r}),\ \{x\}\times[t-T+,t],\ \eta\,\big) = \{\eta(x+iB,t-T) = r_i\ (i=-1,0,1),\ \eta(x,t)=s\}, \qquad b(\alpha(s,r)) = \mathbf{P}(s,\mathbf{r}). \tag{6.7}$$

For all cells x and rational times u we define some new type $\beta = \beta(x,u)$by $b(\beta(x,u)) = 0$ and

$$g\big(\,\beta(x,u),\ \{(x,u)\},\ \eta\,\big) = \{\,\eta(x,u) \neq \eta(x, T\lfloor u/T\rfloor)\,\}.$$

This condition says that $\eta(x,u) = \eta(x, T\lfloor u/T\rfloor)$ with probability 1. For all other combinations α, V we set $g(\alpha, V, \eta) = 0$. It is easy to see that the trajectories of $PCA(\mathbf{P}, B, T)$ are trajectories of the medium defined this way. ♢

Example 6.4. The new conditions introduced below are "loosened-up" versions of the above ones since it is not always desirable to be as restrictive. Let us be given a probabilistic cellular automaton $PCA(\mathbf{P}, B, T)$. For each set of states E, and state vector $\mathbf{r} = (r_{-1}, r_0, r_1)$ let

$$\mathbf{P}(E,\mathbf{r}) = \sum_{q\in E} \mathbf{P}(q,\mathbf{r}).$$

Let $\mathbf{K}$ be the set of functions $K : \mathbb{S}^3 \to 2^{\mathbb{S}}$ such that

$$K(\mathbf{r}) \subset \{\, s \neq r_0 : \mathbf{P}(s,\mathbf{r}) > 0\,\}. \tag{6.8}$$

For a space-time trajectory η and a $K \in \mathbf{K}$ let us write

$$\overline{K}(x,t,\eta) = K(\eta(x-B,t), \eta(x,t), \eta(x+iB,t)).$$

For all $K', K'' \in \mathbf{K}$ with $K' \supset K''$ (i.e. $K'(\mathbf{r}) \supset K''(\mathbf{r})$ for all $\mathbf{r}$), let

$$c(K', K'') = \bigvee_{\mathbf{r}} \frac{\mathbf{P}(K''(\mathbf{r}), \mathbf{r})}{\mathbf{P}(K'(\mathbf{r}), \mathbf{r})}$$

be the least upper bound, over $\mathbf{r}$, on the conditional probability of getting into the set $K''(\mathbf{r})$ provided we get into the set $K''(\mathbf{r})$. For each pair of such functions $K' \supset K''$ let us form the type

$$\alpha = \alpha(K', K'').$$

For each cell x and $u \le v$ of the form nT, let us form the rectangle $V = \{x\} \times [u+, v]$. For each such pair α, V let $g(\alpha, V, \eta) = 1$ if there is a first $t \in [u+, v]$ with $\eta(x, t) \in K'(x, t - T, \eta)$ and in it, we have $\eta(x, t) \in K''(x, t - T, \eta)$. Also, let $b(\alpha) = c(K', K'')$. ♢

Proposition 6.5. *If M is the medium defined by the conditions above then $PCA(\mathbf{P}, B, T) \subset M$.*

Proof sketch. The statement follows by a standard argument from the strong Markov property of the Markov process $PCA(\mathbf{B}, B, T)$ (see [21]). □

Example 6.6. Consider a continuous-time cellular automaton $CCA(\mathbf{R}, B)$. For any functions $K, K', K'' \in \mathbf{K}$ as in Example 6.4, let

$$\mathbf{R}(K, \mathbf{r}) = \sum_{q \in K} \mathbf{R}(q, \mathbf{r}),$$

$$c(K', K'') = \bigvee_{\mathbf{r}} \frac{\mathbf{R}(K''(\mathbf{r}), \mathbf{r})}{\mathbf{R}(K'(\mathbf{r}), \mathbf{r})}.$$

We define each $\alpha(K', K'')$, $V = \{x\} \times [u+, v]$ and $g(\alpha, V, \eta)$ as before, but now for all rational $u < v$. ♢

Theorem 6.7. *Every trajectory of $CCA(\mathbf{R}, B)$ is a trajectory of the medium defined in the above example.*

Proof sketch. As in 2.4.1, for a small $\delta > 0$, let $M_\delta = PCA(\mathbf{P}, B, \delta)$ with $\mathbf{P}(s, \mathbf{r}) = \delta \mathbf{R}(s, \mathbf{r})$ when $s \ne r_0$ and $1 - \delta \sum_{s' \ne r_0} \mathbf{R}(s', \mathbf{r})$ otherwise. Then a transition to the limit $\delta \to 0$ in Proposition 6.5 completes the proof. The transition uses the fact that trajectories of $M(\delta)$ converge in distribution to trajectories of $CCA(\mathbf{R}, B)$. □

Remark 6.8. Of course in both Proposition 6.5 and in Theorem 6.7, instead of the first time with η switching into K' we could have asked about the second time, or the third time, etc. ♢

6.2. **Canonical simulations.**

6.2.1. *Definition of canonical simulations.* Let us introduce a special kind of simulation, that establishes certain relations between the local conditions of the media and thus makes it easier to prove it is a simulation. Let M_1, M_2 be media and let φ^* be a mapping from the space-time configurations of M_1 into the evolutions of M_2: we will write $\eta^* = \varphi^*(\eta)$. We say that φ^* is the map for a *canonical simulation* if the following holds. Let (α, V) be any local condition for M_2. Then there is a system of random local conditions

$$\big(\beta(\alpha, i, j), W^t(\alpha, i, j, V, \eta), \tau(\alpha, i, j, V, \eta)\big) \tag{6.9}$$

with $i = 1, \dots, n(\alpha)$, $j = 1, \dots, k(\alpha, i)$,

$$\inf \pi_t V \le \tau(\alpha, i, j, V, \eta), \tag{6.10}$$

such that for each fixed i, the subsystem containing elements for that i forms a system of disjoint random local conditions with $W^t(\alpha,i,j,V,\eta)\subset V$, and, defining

$$\begin{aligned}\bar W(\alpha,i,j,V,\eta)&=W^{\tau(\alpha,i,j,V,\eta)}(\alpha,i,j,V,\eta),\\ \bar g_1(\alpha,i,j,V,\eta)&=g_1(\beta(\alpha,i,j),\bar W(\alpha,i,j,V,\eta),\eta),\end{aligned}\tag{6.11}$$

we have

$$\begin{aligned}g_2(\alpha,V,\eta^*)&\le\sum_i\prod_j\bar g_1(\alpha,i,j,V,\eta),\\ b_2(\alpha)&\ge\sum_i\prod_j b_1(\beta(\alpha,i,j)).\end{aligned}\tag{6.12}$$

A simulation map will be called a *deterministic canonical simulation* when the system in (6.9) is deterministic: neither $\tau(\alpha,i,j,V,\eta)$ nor $W^t(\alpha,i,j,V,\eta)$ depends on η. Under the condition mentioned in the definition of weak trajectories (see (6.2)), we will be able to provide deterministic canonical simulations and this is why weak trajectories are sufficient. Canonical simulations are "cooked up" to make the following theorem true.

Theorem 6.9 (Canonical Simulation)**.** *If φ^* is a canonical simulation map then for each trajectory η of M_1, the expression $\varphi^*(\eta)$ defines a trajectory of M_2.*

Proof. Let η be a trajectory of M_1 and $(\alpha_l,V_l^t,\sigma_l)_{l\in N}$ a system of disjoint random local conditions in M_2. Let u be some time such that $\sigma_l\ge u$ for each l, and let h be an event function measurable in $\mathcal{A}_u$. We must show

$$\mathbf{E}\big(\,h\prod_l\bar g_2(l,\eta^*)\,\big)\le\mathbf{E}h\prod_l b_2(\alpha_l),$$

where $\bar g_2(l,\zeta)$ is defined as in (6.6). By the assumption, for each l,V there is a system of disjoint random local conditions

$$\big(\,\beta(\alpha_l,i,j),W^t(\alpha_l,i,j,V,\eta),\tau(\alpha_l,i,j,V,\eta)\,\big)$$

for $i=1,\dots,n(\alpha_l)$ $j=1,\dots,k(\alpha_l,i)$ with the properties defined for canonical simulations. Since V has a countable range, we obtain measurable functions even if we substitute $V_l^{\sigma_l}(\eta^*)$ into V here. Let

$$\bar V_l=V_l^{\sigma_l}(\eta^*)$$

and recall (6.11).

1. $\bar{\bar W}^t=\bar W^t(\alpha_l,i,j,\bar V_l,\eta)$ is a rectangle process and $\tau(\alpha_l,i,j,\bar V_l,\eta)$ is a stopping time.

   Proof. It is easy to see that $\pi_{\mathrm{t}}\bar{\bar W}^t\subset[t,\infty]$ and that the process $\bar{\bar W}^t$ is upper semicontinuous almost everywhere. Let us show that for each t it is measurable in $\mathcal{A}_t$. We write

   $$\bar W^t(\alpha_l,i,j,\bar V_l,\eta)=W\Leftrightarrow\bigvee_V\big(\,\bar W^t(\alpha_l,i,j,V,\eta)=W\wedge V=\bar V_l\,\big).$$

   Now (6.10) and $\tau(\alpha_l,i,j,V,\eta)\le t$ implies $\inf\pi_{\mathrm{t}}V\le t$. On the other hand by the definition of random-stopping local conditions, $V=\bar V_l(=V_l^{\sigma_l})$ implies $\sigma_l\le\inf\pi_{\mathrm{t}}V$. Hence we can write the above expression as

   $$\bigvee_V\big(\,\bar W^t(\alpha_l,i,j,V,\eta)=W\wedge V=V_l^{\sigma_l}\,\big)\wedge\sigma_l\le t.$$

   By Theorem 6.2,

   $$\big\{\,\sigma_l\le t\wedge V=V_l^{\sigma_l}\,\big\}\in\mathcal{A}_t.\tag{6.13}$$

By the definition of the rectangle process W^t, also

$$\{\, \bar{W}^t(\alpha_l, i, j, V, \eta) = W \,\} \in \mathcal{A}_t,$$

hence indeed $\{\, \bar{\bar{W}}^t = W \,\} \in \mathcal{A}_t$. Finally, we have to show

$$\{\, \tau(\alpha_l, i, j, \bar{V}_l, \eta) \le t \,\} \in \mathcal{A}_t.$$

As above,

$$\tau(\alpha_l, i, j, \bar{V}_l, \eta) \le t \Leftrightarrow \bigvee_V \big(\, \tau(\alpha_l, i, j, V, \eta) \le t \wedge V = \bar{V}_l \,\big)$$

which again can be written as

$$\bigvee_V \big(\, \tau(\alpha_l, i, j, V, \eta) \le t \wedge V = V_l^{\sigma_l} \,\big) \wedge \sigma_l \le t.$$

We have $\{\, \tau(\alpha_l, i, j, V, \eta) \le t \,\} \in \mathcal{A}_t$ since $\tau(\alpha_l, i, j, V, \eta)$ is a stopping time; combining this with (6.13) completes the proof.

By (6.12) we have

$$g_2(\alpha, \bar{V}_l, \eta^*) \le \sum_i \prod_j g_1(\alpha_l, i, j, \bar{V}_l, \eta). \tag{6.14}$$

and

$$\prod_l b_2(\alpha_l) \ge \prod_l \sum_i \prod_j b_1(\beta(\alpha_l, i, j)) = \sum_{i(\cdot)} \prod_{l,j} b_1(\beta(\alpha_l, i(l), j))$$

where the last sum is over all functions $i(\cdot)$ with $i(l) \in [1, n(\alpha_l)]$. Similarly, by (6.14)

$$g_2(\alpha, \bar{V}_l, \eta^*) \le \sum_{i(\cdot)} \prod_{l,j} \bar{g}_1(\alpha_l, i(l), j, \bar{V}_l, \eta).$$

Therefore it is sufficient to show

$$\mathbf{E}\big(\, h \prod_{l,j} \bar{g}_1(\alpha_l, i(l), j, \bar{V}_l, \eta) \,\big) \le \mathbf{E} h \prod_{l,j} b_1(\beta(\alpha_l, i(l), j)).$$

Let us fix a function $i(l)$. Recall that by (6.11),

$$\bar{g}_1(\alpha, i, j, \bar{V}_l, \eta) = g_1(\beta(\alpha, i, j), \bar{W}(\alpha_l, i, j, \bar{V}_l, \eta), \eta).$$

For any l, i, the rectangles $\bar{W}(\alpha_l, i, j, \bar{V}_l, \eta)$ are all disjoint and belong to $\bar{V}_l$. Also the sets $\bar{V}_l$ are disjoint, therefore all rectangles $\bar{W}(\alpha_l, i(l), j, \bar{V}_l, \eta)$ are disjoint. Recall that also by (6.11),

$$\bar{W}(\alpha, i, j, \bar{V}_l, \eta) = W^{\tau(\alpha, i, j, \bar{V}_l, \eta)}(\alpha, i, j, \bar{V}_l, \eta),$$

therefore the system

$$\big(\, \beta(\alpha_l, i(l), j), \bar{W}(\alpha, i, j, \bar{V}_l, \eta), \tau(\alpha, i, j, \bar{V}_l, \eta) \,\big)$$

taken for all l, j, is a system of disjoint random local conditions with $u \le \tau(\alpha, i, j, \bar{V}_l, \eta)$, and hence the proof is finished by the trajectory property of η. □

6.2.2. *Frequency of switching times.* Adding certain conditions does not really change some media. Let us call a canonical simulation *injective* if the mapping φ^* is the identity, and the functions $b_2(\alpha), g_2(\alpha,\cdot,\cdot)$ differ from b_1, g_1 only in that they are defined for some additional types α, too. Proving a certain probability bound for the behavior of η_1 within some window does not create a canonical simulation yet: the bound must be formulated in such a way that this type of probability bound can be multiplied with itself and all the other bounds over a system of disjoint windows (and these windows can even be chosen via stopping times).

Let us illustrate canonical simulations on the the following example. The simulating medium M_1 is the *CCA* of Example 6.6. The conditions to add impose lower and upper bounds on the frequency of switching times of M_1. Namely, for some function $K \in \mathbf{K}$ let

$$a_{-1}(K) = \bigwedge_{\mathbf{r}} \mathbf{R}(K(\mathbf{r}), \mathbf{r}),$$
$$a_1(K) = \bigvee_{\mathbf{r}} \mathbf{R}(K(\mathbf{r}), \mathbf{r}).$$

Let us call a time u of the space-time configuration η a *K-switching-time* of a cell x if for all $t < u$ sufficiently close to u we have $\eta(x,u) \in K(x,t,\eta)$, i.e. we have just jumped into the target set determined by $K(\cdot)$.

For some rational constant $D > 0$, integers $l \geq 0$ and $j = -1, 1$ let

$$g\big(\,\alpha(K,D,k,j),\ \{x\} \times [u+, u+D],\ \eta\,\big) \tag{6.15}$$

be the event function of type $\alpha(K,D,k,j)$ for $j = -1$ [$j = 1$] for the event that site x has at most [at least] k times during $[u+, u+D]$, that are K-switching times.

Proposition 6.10. *For each transition rate matrix* $\mathbf{R}$ *and function* $K \in \mathbf{K}$ *there is a function* $\kappa(\varepsilon) > 0$ *such that for all* $k > 0$*, defining, with* $a_j = a_j(K)$

$$b(\alpha(K,D,(a_j + j\varepsilon)D, j)) = e^{-\kappa(\varepsilon)D} \qquad \text{unless } j = 1 \text{ and } (a_1+\varepsilon)D = 1,$$
$$b(\alpha(K,D,1,1)) = e^{-\kappa(\varepsilon)D} \wedge a_1 D \tag{6.16}$$

(and defining $b(\alpha(K,D,k,j)) = 1$ *for all other choices of* k*) the trajectories of* $CCA(\mathbf{R}, B)$ *are also trajectories of the medium obtained by adding all these conditions.*

The proof proceeds by first estimating the event in question by sums of products of single switching events and using then a standard large-deviation estimate.

6.3. **Primitive variable-period media.** We say that $T_\bullet > 0$ is a *dwell period lower bound* of a space-time configuration η if no dwell period of η is shorter than $T_\bullet$. A continuous-time cellular automaton has no such lower bound.

In the amplifier $M_0, M_1, \ldots$ we will construct eventually on top of a continuous-time probabilistic cellular automaton M_0, all abstract media $M_1, M_2, \ldots$ will have a dwell period lower bound, only M_0 will not have one. M_1 will be a so-called *primitive variable-period medium*: these can be considered the continuous-time extension of the notion of an ε-perturbation of a deterministic cellular automaton with coin-tossing. On the other hand, $M_2, M_3, \ldots$ will only fit into a more general framework (non-adjacent cells).

Let us thus proceed to the definition of medium

$$\mathit{Prim\_var}(\mathbb{S}, \mathbf{C}, Tr, B, T_\bullet, T^\bullet, \varepsilon).$$

The set $\mathbb{S}$ of states is implicit in the transition function therefore from now on, it will be omitted. We have dwell period lower and upper bounds $T_\bullet \leq T^\bullet$ and a failure probability bound $\varepsilon > 0$. The

local state, as in Example 2.4, is a record with two fields, *Det* and *Rand* where *Rand* consists of a single bit which is, say, always the first bit.

To simplify the upper-bounding of dwell periods, we assume that the transition function has the property

$$Tr(r_{-1}, r_0, r_1).Det \neq r_0.Det. \tag{6.17}$$

For a space-time configuration η, site x and rational number $a > 0$ let $\sigma_1 = \sigma_1(x, a, \eta)$, and σ_2 be the first and second switching times $t > a$ of η in x. Let us list the different types of local condition.

(a) Conditions of type $\alpha(dw\_p\_lb)$ imposing $T_\bullet$ as a dwell period lower bound. We have $b(\alpha(dw\_p\_lb)) = \varepsilon$, and

$$g\big(\,\alpha(dw\_p\_lb),\ \{x\} \times [a, a + T^\bullet],\ \eta\,\big)$$

is the event function saying that $\eta(x, t)$ has two switching times closer than $T_\bullet$ to each other during $[a+, a + T_\bullet-]$.

(b) Conditions of type $\alpha(dw\_p\_ub)$ imposing $T^\bullet$ as a dwell period upper bound. We have $b(\alpha(dw\_p\_ub)) = \varepsilon$, and

$$g\big(\,\alpha(dw\_p\_ub),\ \{x\} \times [a+, a + T^\bullet],\ \eta\,\big)$$

is the event function saying that $\eta(x, t)$ has no switching times in $[a+, a + T^\bullet]$.

(c) Conditions of type $\alpha(comput)$ saying that the new value of the *Det* field at a switching time obeys the transition function. We have $b(\alpha(comput)) = \varepsilon$, and

$$g\big(\,\alpha(comput),\ \{x\} \times [a, a + 2T^\bullet],\ \eta\,\big) \\ = \{\eta(x, \sigma_2).Det \notin \{\,\overline{Tr}(\eta, x, t, B).Det : t \in [\sigma_1, \sigma_2 - T_\bullet/2]\,\}\}.$$

Here we used the notation (3.2).

(d) Conditions of type $\alpha(coin, j)$ for $j = 0, 1$ saying that the new value of the *Rand* field at a switching time is obtained nearly by a fresh coin-toss:

$$b(\alpha(coin, j)) = 0.5 + \varepsilon,$$
$$g\big(\,\alpha(coin, j),\ \{x\} \times [a, a + T^\bullet],\ \eta\,\big) = \{\eta(x, \sigma_2).Rand = j\}.$$

Condition (c) says that unless an exceptional event occurred, whenever a state transition takes place at the end of a dwell period $[\sigma_1, \sigma_2]$ it occurs according to a transition made on the basis of the states of the three neighbor cells and the random bit at some time in the *observation interval* $[\sigma_1, \sigma_2 - T_\bullet/2]$. Since the observation interval does not include the times too close to σ_2, information cannot propagate arbitrarily fast.

Example 6.11. All trajectories of the ordinary deterministic cellular automaton $CA(Tr, B, T)$ are also trajectories of $Prim\_var(Tr, B, T, T, 0)$. ♢

Theorem 6.12 (Simulation by *CCA*)**.** *For medium*

$$M = Prim\_var(Tr, 1, T_\bullet, 1, \varepsilon)$$

with (6.17) *and* $T_\bullet < 1$ *there is a noisy transition rate* $\mathbf{R}(s, \mathbf{r})$ *over some state space* $\mathbb{S}'$ *and a function* $\pi : \mathbb{S}' \to \mathbb{S}$ *such that* $\pi(\eta(x, t))$ *is a trajectory of* M *for each trajectory* η *of* $CCA(\mathbf{R}, 1)$*.*

The proof is a straightforward application of Proposition 6.10.

6.4. **Main theorems (continuous time).** The first publication showing the possibility of reliable computation with a continuous-time medium (in two dimenstions) is [30]. Here, we formulate the new results for variable-period one-dimensional information storage and computation. The following theorems are just like Theorems 2.5, 5.6 and 5.8 except that they have a variable-period medium $Prim\_var(Tr', 1, T_\bullet, 1, \varepsilon)$ with $T_\bullet = 0.5$ in place of the perturbed discrete-time automaton $CA_\varepsilon(Tr')$. See the Corollary below for continuous-time interacting particle systems.

Theorem 6.13 (1-dim non-ergodicity, variable-period)**.**
There is a one-dimensional transition function such that its primitive variable-period perturbations remember a field.

Theorem 6.14 (FCA, 1-dim, storage, variable-period)**.** *There is*

- *a transition function Tr;*
- *a hierarchical code Ψ with a shared field (F^k) as in* (4.13)*;*
- *constants $d_1, c_2 > 0$;*

such that for $h_2(n) = n^{c_2/\log\log n}$, all $\varepsilon > 0$, $t > 0$, for $T_\bullet = 0.5$, any configuration ϱ with states in $\mathbb{S}.F^1$, trajectory η of $Prim\_var(Tr, 1, T_\bullet, 1, \varepsilon)$, over $\mathbf{C} = \mathbb{Z}_N$ (for finite or infinite N) with initial configuration $\Gamma(\varrho; \Psi)$, for all for all y in $Visible(K(N), N)$, we have

$$\mathrm{Prob}\{\, \varrho(y) \neq \eta(X(y;\Psi), t).F^1 \,\} < d_1\varepsilon + t\varepsilon^{h_2(N)}.$$

The code can be chosen to have $X(y; K, \Psi) = y$.

The main result of the present paper is the following one, asserting the existence of a one-dimensional fault-tolerant variable-period cellular automaton:

Theorem 6.15 (FCA, 1-dim, variable period, known bounds)**.** *This theorem says the same as Theorem 5.8, with again $Prim\_var(Tr', 1, 0.5, 1, \varepsilon)$, replacing $CA_\varepsilon(Tr')$.*

For the interpretation of this theorem, see the comment after Theorem 5.8. The construction indicated in 5.2.1 has a similar variable-period counterpart. Theorem 6.12 implies the following.

Corollary 6.16 (Interacting particle system construction)**.** *In Theorems 6.13, 6.14, 6.15, we can replace Tr' with a rate matrix $\mathbf{R}$ and $Prim\_var(Tr', 1, T_\bullet, 1, \varepsilon)$, with a CCA with rates coming from an arbitrary ε-perturbation of $\mathbf{R}$. This proves, in particular, Theorem 2.6.*

6.5. **Self-organization.** If the input information is a single symbol then an (infinite or large finite) hierarchical initial configuration in the above reliable simulation theorems is less natural than one consisting of the repetition of this symbol.

Theorem 6.17 (FCA, 1-dim, storage, variable-period, self-organizing)**.** *For each m there is*

- *a transition function Tr with state space $\mathbb{S}$ having a field Output;*
- *a mapping $\gamma : \{0,1\}^m \to \mathbb{S}$;*
- *constants $c_1, c_2 > 0$,*

such that for $h_2(n) = n^{c_2/\log\log n}$, $\varepsilon > 0$, $t > 0$, for any string s of length m in the standard alphabet, any trajectory η of $Prim\_var(Tr, 1, T_\bullet, 1, \varepsilon)$ over $\mathbf{C} = \mathbb{Z}_N$ (for finite or infinite N) having initial configuration $\eta(y, 0) = \gamma(s)$ for all y, the following holds for all x:

$$\mathrm{Prob}\{\, s \neq \eta(x,t).Output \,\}/c_1 < \varepsilon + t\varepsilon^{h_2(N)} + (t^2\varepsilon) \wedge (t^{-0.4} + N^{-0.4}).$$

Though it is not seen from the theorem, hierarchical organization actually emerges here: the configuration $\eta(\cdot, t)$ will consist of islands of varying levels of "organization" and the level will be generally growing with t. For infinite N we can ignore the terms involving N (since they converge

to 0), obtaining $\varepsilon + (t^2\varepsilon) \wedge t^{-0.4}$. The term $t^{-0.4}$ decreases rather slowly. This term is essentially the upper bound on the probability that organization does not grow to the required level until time t.

To formulate a theorem about self-organizing computation, assume that the input is again, as in the storage case, just a binary string s fitting into a single cell. (If you want a larger input, we would have to encode it into a higher-order cell and repeat that cell.) However, the simulated computation may last forever producing ever-growing monotonic output. Where to read the currently available output when the input has no distinguished origin? This must depend on the length of the output. We can expect that there is a constant c_0 such that for some sequences of lengths $B'_1 < B'_2 < \cdots$, $B_1 < B_2 < \cdots$, if the computation output is a string of length $\geq B'_k$ by some time t, then by all times exceeding a certain time $h_1(t)$, its first segment of length B'_k "can be located somewhere" in every interval $I_k(x) = [x, x + c_0 B_k]$. Since this requires the short starting segments of the output to occur more densely than the longer ones, we store the output on infinitely many different tracks simultaneously. As for the infinitely many fields F^k_i in Proposition 4.10, let $Y + X_k(y)$ denote the position where the y-th symbol of the k-th track is stored, if the storage starts at position Y. (In our construction, Y will be the site of a cell of the simulated medium M_k whose body fits into $I_k(x)$.)

The theorem on self-organizing computation says that there is a constant d_0 and a function $Y_k(\cdot,\cdot,\cdot)$ such that for every k, x, y, t, in every interval $I_k(x)$, with $Y = Y_k(x, h_1(t), \eta)$, with probability $\geq 1 - d_0\varepsilon$ we have

$$\zeta(y,t).\mathit{Output} \preceq \eta(Y + X_k(y), h_1(t)).F^1_1.$$

7. Some simulations

7.1. **Simulating a cellular automaton by a variable-period medium.** The random nature of the switching times of a variable-period medium is a tame kind of nondeterminism; any deterministic cellular automaton can be simulated by a variable-period medium. To prove this we first introduce an auxiliary concept. Let us define the *totally asynchronous cellular automaton*

$$ACA(Tr) = ACA(Tr, 1, 1)$$

associated with transition function Tr as follows: η is a trajectory if for all x, t we have either $\eta(x, t+1) = \eta(x, t)$ or the usual

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

A site x is *free* in a configuration ξ if $Tr(\xi(x-1), \xi(x), \xi(x+1)) \neq \xi(x)$. The set of free sites will be denoted by $L(\xi)$. For a space configuration ξ and a set E of sites, let us define the new configuration $Tr(\xi, E)$ by

$$Tr(\xi, E)(x) = \begin{cases} Tr(\xi(x-1), \xi(x), \xi(x+1)) & \text{if } x \in E \\ \xi(x) & \text{otherwise.} \end{cases}$$

Now we can express the condition that η is a trajectory of $ACA(Tr)$ by saying that for every t there is a set U with

$$\eta(\cdot, t+1) = Tr(\eta(\cdot, t), U). \tag{7.1}$$

Let the *update set*

$$U(t, \eta) \tag{7.2}$$

be the set of sites x with $\eta(x, t+1) \neq \eta(x, t)$. The initial configuration and the update sets $U(t, \eta)$ determine η. For any set A, let us use the indicator function

$$\chi(x, A) = \begin{cases} 1 & \text{if } x \in A, \\ 0 & \text{otherwise.} \end{cases}$$

For given η we define the function $\tau(x, t) = \tau(x, t, \eta)$ as follows:

$$\begin{aligned} \tau(x, 0) &= 0, \\ \tau(x, t+1) &= \tau(x, t) + \chi(x, U(t, \eta)). \end{aligned} \tag{7.3}$$

We can call $\tau(x, t)$ the *effective age* of x in η at time t: this is the number of effective updatings that x underwent until time t. Given a transition function Tr and an initial configuration ξ, we say that the function *has invariant histories* if there is a function $\zeta(x, u) = \zeta(x, u, \xi)$ such that for all trajectories $\eta(x, t)$ of $ACA(Tr)$ with $\eta(\cdot, 0) = \xi$ we have

$$\eta(x, t) = \zeta(x, \tau(x, t, \eta), \xi). \tag{7.4}$$

This means that after eliminating repetitions, the sequence $\zeta(x, 1), \zeta(x, 2), \ldots$ of values that a site x will go through during some trajectory, does not depend on the update sets, only on the initial configuration (except that the sequence may be finite if there is only a finite number of successful updates). The update sets influence only the delays in going through this sequence. Let

$$Tr(\xi, E, F) = Tr(Tr(\xi, E), F).$$

We call a transition function Tr *commutative* if for all configurations ξ and all distinct pairs $x, y \in L(\xi)$ we have $Tr(\xi, \{x\}, \{y\}) = Tr(\xi, \{y\}, \{x\})$. The paper [12] proves the theorem that if a transition function is commutative then it has invariant histories. In Theorem 7.1 below, we will give a simple

example of a universal commutative transition function. For that example, the theorem can be proved much easier.

Below, we will use the smallest absolute-value remainders

$$b \bmod m \tag{7.5}$$

with respect to a positive integer $m > 2$, defined by the requirement $-m/2 < b \bmod m \le m/2$.

Theorem 7.1 (Commutative Simulation)**.** *Let Tr_2 be an arbitrary transition function with state space $\mathbb{S}_2$. Then there is a commutative transition function Tr_1 with state space $\mathbb{S}_1 = \mathbb{S}_2 \times R$ (for an appropriate finite set R) with the following property. Each state $s \in \mathbb{S}_1$ can be represented as $(s.F, s.G)$ where $s.F \in \mathbb{S}_2$, $s.G \in R$. Let ξ_2 be an arbitrary configuration of $\mathbb{S}_2$ and let ξ_1 be a configuration of $\mathbb{S}_1$ such that for all x we have $\xi_1(x).F = \xi_2(x)$, $\xi_1(x).G = 0 \cdots 0 \in R$. Then for the trajectory η_1 of $CA(Tr_1)$, with initial configuration ξ_1, the function $\eta_1(x,t).F$ is a trajectory of $CA(Tr_2)$. Moreover, in η_1, the state of each cell changes in each step.*

In other words, the function Tr_1 behaves in its field F just like the arbitrary transition function Tr_2, but it also supports asynchronous updating.

Proof. Let $U > 2$ be a positive integer and

$$Cur, Prev, Age$$

be three fields of the states of $\mathbb{S}_1$, where $F = Cur$, $G = (Prev, Age)$. The field Age represents numbers mod U. It will be used to keep track of the time of the simulated cells mod U, while $Prev$ holds the value of Cur for the previous value of Age.

Let us define $s' = Tr_1(s_{-1}, s_0, s_1)$. If there is a $j \in \{-1, 1\}$ such that $(s_j.Age - s_0.Age) \bmod U < 0$ (i.e. some neighbor lags behind) then $s' = s_0$ i.e. there is no effect. Otherwise, let $r_0 = s_0.Cur$, and for $j = -1, 1$, let r_j be equal to $s_j.Cur$ if $s_j.Age = s_0.Age$, and $s_j.Prev$ otherwise.

$$\begin{aligned} s'.Cur &= Tr_2(r_{-1}, r_0, r_1), \\ s'.Prev &= s_0.Cur, \\ s'.Age &= s_0.Age + 1 \bmod U. \end{aligned}$$

Thus, we use the Cur and $Prev$ fields of the neighbors according to their meaning and update the three fields according to their meaning. It is easy to check that this transition function simulates Tr_2 in the Cur field if we start it by putting 0 into all other fields.

Let us check that Tr_1 is commutative. If two neighbors $x, x+1$ are both allowed to update then neither of them is behind the other modulo U, hence they both have the same Age field. Suppose that x updates before $x+1$. In this case, x will use the Cur field of $x+1$ for updating and put its own Cur field into $Prev$. Next, since now x is "ahead" according to Age, cell $x+1$ will use the $Prev$ field of x for updating: this was the Cur field of x before. Therefore the effect of consecutive updating is the same as that of simultaneous updating. □

The commutative medium of the above proof will be called the *marching soldiers* scheme since its handling of the Age field reminds one of a chain of soldiers marching ahead in which two neighbors do not want to be separated by more than one step.

Remark 7.2. In typical cases of asynchronous computation, there are more efficient ways to build a commutative transition function than to store the whole previous state in the $Prev$ field. Indeed, the transition function typically has a bandwidth (see 2.2.1) smaller than $\|\mathbb{S}\|$. ♢

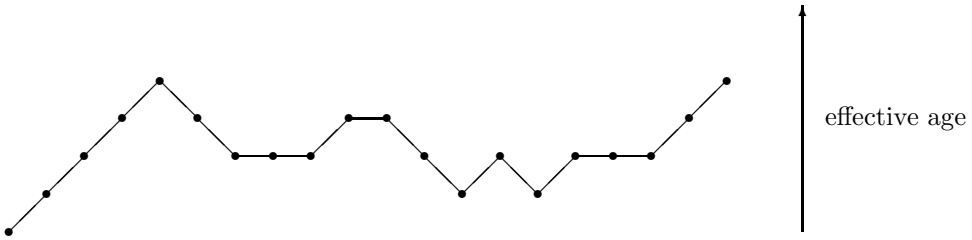

FIGURE 10. The Marching Soldiers scheme. The effective age of neighbor sites differs by at most 1.

Corollary 7.3 (Variable-period simulation)**.** *For every deterministic transition function Tr_2 over some state-space $\mathbb{S}_2$, there is a set of states $\mathbb{S}_1$, a transition function Tr_1 over $\mathbb{S}_1$, and a code that for any values $T_{\bullet 1} \le T^{\bullet}{}_1$, is a simulation of $CA(Tr_2)$ by Prim_var$(Tr_1, 1, T_{\bullet 1}, T^{\bullet}{}_1, 0)$.*

Proof. Let Tr_1 be the commutative transition function given by Theorem 7.1, with the fields F, G. Let ξ_2 be an arbitrary configuration of S_2 and let ξ_1 be a configuration of S_1 defined in the statement of the same theorem. Let η_1 be a trajectory of Prim_var$(Tr_1, 1, T_{\bullet 1}, T^{\bullet}{}_1, 0)$, with the starting configuration ξ_1.

An update set $U(t, \eta_1)$ similar to (7.2) can be defined now for the trajectory η_1 as follows: x is in $U(t, \eta_1)$ iff t is a switching time of η_1. Similarly, $\tau(x, t, \eta_1)$ can be defined as in (7.3):

$$\tau(x, 0) = 0,$$
$$\tau(x, t) = \tau(x, t-) + \chi(x, U(t, \eta_1)).$$

With these, let us define

$$\sigma(x, s, \xi) = \bigwedge\{\, t : \tau(x, t, \eta') = s \,\},$$
$$\eta_2(x, s, \xi) = \eta_1(x, \sigma(x, s)).F$$

By the cited theorem, η_2 is a trajectory of $CA(Tr_2)$. □

The simulation in this theorem is not a local one in the sense defined in Subsection 3.6 since it changes the time scale. For an analysis of such simulations, see [5].

7.2. **Functions defined by programs.** Let us recall the definition of a standard computing transition function as introduced in Subsection 5.2, and let us call a cellular automaton with such a transition function a *standard computing medium*. For a standard computing transition function Tr, integers s and t and string X consider a trajectory η of $CA(Tr)$ over the rectangle $[0, s] \times [0, t]$ with an initial configuration in which $\eta(0, 0) = \eta(s, 0) = \# \cdots \#$, further X is on the input track on $[1, s-1]$ (padded with $*$'s to fill up $[1, s-1]$), $*$'s on the output track and 0's on the other tracks. This defines a trajectory η since the #'s on the input field in cells 0 and s imply that the cells outside the interval $[0, s]$ will have no effect.

Assume that at time t, the *Output* track has no $*$ on $[1, s-1]$. Then the output never changes anymore (it is monotonic in standard computing media). The string w on the *Output* track on $[1, s-1]$ will be called the *result* of the computation, denoted by

$$w = Tr(X; s, t).$$

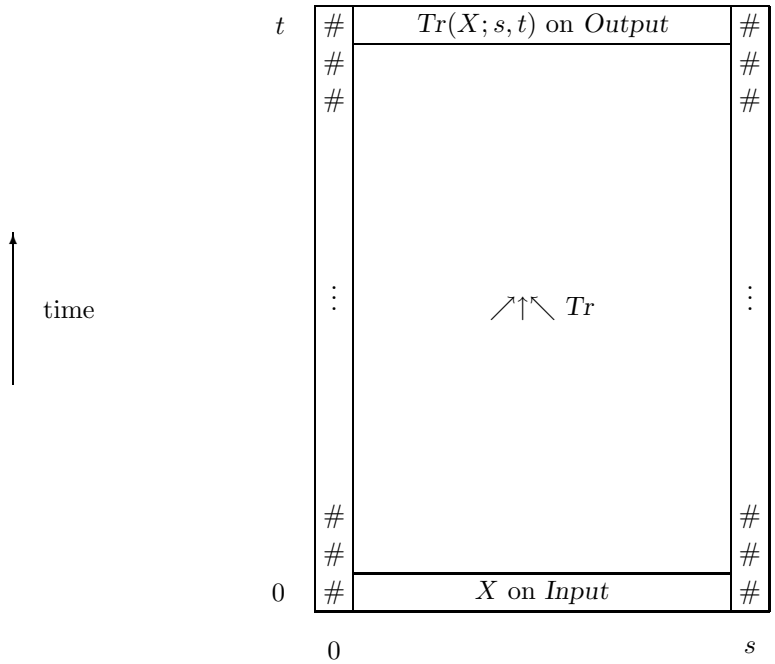

FIGURE 11. Definition of $trans(X;s,t)$

7.2.1. *Efficient universality.* We say that the standard computing transition function Tr_0 is *efficiently universal* if for every standard computing transition function Tr, there is a string *prog* and constant c such that for all strings X and positive integers s, t we have

$$Tr(X;s,t) = Tr_0(prog \sqcup X; cs, c(s+t))$$

whenever the left-hand side is defined. In other words, Tr_0 can simulate the computation of any other standard computing transition Tr if we prepend a "program of Tr". The space and time needed for this simulation can only be a constant factor greater than those of the simulated computation.

Theorem 7.4. *There are efficiently universal standard computing transition functions.*

Sketch of the proof. This theorem can be proven similarly to Theorem 3.5. The main addition to the construction is that before Tr_0 starts a block simulation of Tr the input will be distributed bit-by-bit to the colonies simulating the cells of Tr. At the end, the output will be collected from these colonies. □

We fix an efficiently universal standard computing transition function

(7.6) *Univ*

for the definition of various string functions. In view of Theorem 7.1, we can (and will) also assume that *Univ* is commutative. For a string function $f(X)$ defined over some domain E we will say that

prog is a program for f with *time complexity bound* t *and space complexity bound* s *over* E if we have

$$Univ(prog \sqcup X; s, t) = f(X).$$

It is often convenient to define a finite function (like a transition function of some medium, or a code) by its program (for *Univ*) rather than its transition table, since for many interesting functions, the program can be given in a more concise form.

7.3. **The rule language.** This subsection defines a language for describing a transition function that is more convenient than giving a complete transition table. A transition function will generally be defined by a set of *rules* which impose some conditions on it. In case some rules are contradictory and no precedence is stated between them then rules given later override the ones given earlier. Rules are sometimes associated with certain fields. The lowest-priority rule referring to a field is called the *default*.

Some rules will not be applicable directly, but will rather be used by other rules: these will be called *subrules*. A subrule R is just like other rule, except that, typically, there is some other rule saying in which time interval of the work period to invoke R.

Our use of the rule language will not be completely formal but it will be evident how to make it formal. In a semi-formal description of a rule, $\mathbf{x}$ refers to the cell to which it is applied. This is only a convenient shorthand that can be replaced with references to fields of the cell and its neighbors. We will write

$$\vartheta_j(x) = x + jB$$

for the site j steps from x. In the condition as well as the action of the rule, a field $F(\mathbf{x})$ will simply be written as F. We will often write

$$F^j = \eta.F(\vartheta_j(\mathbf{x})).$$

For example, for field $Addr$, instead of writing "`if` $\eta.Addr(\mathbf{x}) = 0$ `and` $Addr(\vartheta_j(\mathbf{x})) = 1$" we will write "`if` $Addr = 0$ `and` $Addr^j = 1$".

The simplest possible rule is an assignment of the form $F := g$ where F is a field and g is a function of some fields, e.g. $F := G^{-1} + H^1$. Another possible rule is the conditional:

`cond`{
`?` C_1
`!` A_1
...
`?` C_k
`!` A_k
`[?!` A_{k+1}`]`
}

Here, $C_1, C_2, \ldots$ are *conditions*: inequalities and equations in terms of the states of $\vartheta_j(\mathbf{x})$, and $A_1, \ldots, A_{k+1}$ are possible *actions* each of which can be another rule. We can use logical connectives, even (finite) quantifiers in the conditions. The above rule says that if condition C_1 is satisfied then the action A_1 must be performed. Else if C_2 holds then action A_2 must be performed, etc. Finally, and optionally, (this is meant by "[,]") if all conditions fail then we may require a default action A_{k+1} to be performed. The symbols `?`, `!`, `?!` abbreviate *condition*, *action* and *default* respectively. An alternative way of writing would be

`if` C_1 {
`then` A_1
`else if` C_2

`then` A_2
...
`else` A_{k+1}
}

Remark 7.5. We prefer the first form (borrowed from the programming language LISP) only in order to put all conditions formally on the same level. ♢

The rule

$$R_1 \parallel R_2$$

is a combination of two rules. It says that both rules must be carried out simultaneously. In case the two rules are just two assignments we will sometimes omit the sign $\parallel$. The rule

$$R_1; R_2$$

asks for carrying out the rules R_1 and R_2 consecutively. This informal notation can be understood in terms of a field *Age* which we are always going to have and can be replaced with a conditional. E.g., whenever we write

Retrieve;
Eval

then this can be replaced, using appropriate constants t_i, with

`cond` {
`?` $t_1 \le Age < t_2$
`!` *Retrieve*
`?` $t_2 \le Age < t_3$
`!` *Eval*
}

We will also use the construct

`for` $i = a$ `to` b ⟨ `some rule referring to` i⟩.

The construct

`repeat` k `times` $\langle \cdots \rangle$

is a special case.

A subrule can have parameters (arguments). Subrule $P(i)$ with parameter i can be viewed as a different rule for each possible value of i. An example could be a simple subrule *Read_Mail*(j) which for $j \in \{-1, 1\}$ has the form

$$Mail := Mail^j$$

for the field *Mail*. We will also have *functions*: these are defined by the information available in the arguments of the transition function, and can always be considered a shorthand notation: e.g., we could denote

$$f(i) = Addr + Age^i.$$

The construct

`let` $k = \ldots$

can be understood as notational shorthand (though could also be incorporated into the rule language).

Example 7.6. Here is a description of the transition function given in the proof of Theorem 7.1 (Asynchronous Simulation).

```
rule March {
   for j ∈ {−1, 0, 1}, {
      let r(j) = Cur^j if Age^j = Age, and Prev^j otherwise;  }
   cond {
      ?  ∀j ∈ {−1, 1}  (Age^j − Age) amod U ≥ 0
      !  {
         Prev := Cur;
         Cur := Tr_2(r(−1), r(0), r(1))  ||
         Age := Age + 1 mod U
         }}}
```

♢

We will also have a field *Addr*. A *colony* is a segment of cells with addresses growing from 0 to $Q-1$ modulo some constant Q.

A *location* is a pair (F, I) where F is a field, and I is an interval of addresses in the colony. We will denote it as

$$F(I).$$

As an element of the language, of course, a location is simply a triple consisting of the field, and the numbers determining the endpoints of the interval. A location is meant to specify a sequence of symbols on track F. If a shorthand name is given a location, this will generally begin with a minus sign, like

$$\_Info.$$

Remark 7.7. Occasionally, we may treat the union of two or three locations as one. It is easy to generalize the rules dealing with locations to this case. ♢

Let us be given a string S consisting of a constant number of runs of the same symbol. For example, 0^m1^n has one run of 0's and a run of 1's. Let us also be given a location *loc*. Then a rule $Write(S, loc)$, writing the string S to the location *loc*, can be written as a conditional rule, as long as there are runs of 0's and 1's:

```
subrule Write(0^m 1^n, F([a, a + m + n − 1])){
   cond {
      ?  a ≤ Addr < Q ∧ (a + m)
      !  F := 0
      ?  a + m ≤ Addr < Q ∧ (a + m + n)
      !  F := 1
      }}
```

The rule language can contain some definitions of names for constant strings of symbols, of the form

$$Param_0 = s_0, Param_1 = s_1,\ Param_2 = s_2, \dots . \tag{7.7}$$

where s_i are some strings. The names $Param_i$ are allowed to be used in the rules. Let us even agree that the whole program is nothing but a sequence of such definitions, where the first string s_0 is the sequence of all the rules. In this case, using the name $Param_0$ in the rules is a kind of self-reference, but its interpretation is clear since it denotes a definite string.

Theorem 7.8 (Rule Language)**.** *There is a string Interpr and an integer* (interpr_coe) *such that the following holds. If string P is a description of a transition rule Tr over state set $\mathbb{S}$ in the above*

language (along with the necessary parameters) then the machine Univ *defined in* (7.6) *computes* $Tr(r_{-1}, r_0, r_1)$ *(possibly padded with* $*$*'s) from*

$$Interpr \sqcup P \sqcup r_{-1} \sqcup r_0 \sqcup r_1$$

within computation time $(\text{interpr\_coe})(|P|+1)^2\|\mathbb{S}\|$ *and space* $(\text{interpr\_coe})(|P|+\|\mathbb{S}\|+1)$.

Sketch of proof. A detailed proof would be tedious but routine. Essentially, each line of a rule program is some comparison involving some fields: substrings of a state argument r_i. We have $(|P|+1)$ squared in the time upper bound since we may have to look up some parameter repeatedly. □

From now on, by a *rule program Trans_prog* of the transition function Tr, we will understand some string to be interpreted by *Interpr*.

7.4. **A basic block simulation.** The simulation described here is used just to demonstrate the use of the notation and to introduce some elements of the later construction in a simple setting. Let a transition function Tr_2 be given. We want to define a cellular automaton $M_1 = CA(Tr_1)$, whose trajectories simulate the trajectories of $M_2 = CA(Tr_2, Q, U)$, with appropriate Q, U. Of course, there is a trivial simulation, when $M_1 = M_2$, but a more general scheme will be set up here. This simulation is not one of the typical simulations by a universal medium: the cell-size of M_1 depends on M_2 as in Example 3.1. The construction will be summarized in Theorem 7.13 below.

In the construction, we introduce some more notation that can be considered a shorthand in writing the rules but can also be incorporated into the rule language (without invalidating Theorem 7.8 (Rule Language)).

7.4.1. *Overall structure.* The transition function $Tr_2 : \mathbb{S}_2^3 \to \mathbb{S}_2$ to be simulated is given by a rule program $Trans\_prog_2$. To perform a simulated state transition of M_2, a colony of M_1 must do the following:

Retrieve: Retrieve the states of the represented neighbor cells from the neighbor colonies;
Evaluate: Compute the new state using Tr_2;
Update: Replace the old represented state with the new one.

The field *Addr* holds a number between 0 and $Q-1$, as discussed in Subsection 3.1. The default operation is to keep this field unchanged. The time steps within a work period of a colony are numbered consecutively from 0 to $U-1$. The field *Age* holds a number between 0 and $U-1$ intended to be equal to this number. The default operation is to increase this by 1 modulo U. These default operations will not be overridden in the simple, fault-free simulations. Using these fields, each cell knows its role at the current stage of computation.

On the track *Info*, each colony holds a binary string of length $\|\mathbb{S}_2\|$. For a string $S \in \mathbb{S}_1^Q$, the decoded value $\varphi^*(S)$ is obtained by taking this binary string. The encoding will be defined later. The default operation on the information field is to leave it unchanged. It will be overridden only in the last, updating step. For simplicity, let us take $|Info| = 2$, i.e. the *Info* track contains only symbols from the standard alphabet. The field *Cpt* will be used much of the time like the cells of a standard computing medium, so its size is the capacity $|Univ| = \|\mathbb{S}_{Univ}\|$ of the fixed efficiently universal standard computing medium. It has subfields *Input*, *Output*. The field *Cpt.Input* is under the control of the rule *Retrieve*, while rest of *Cpt* is under the control of the rule *Eval*.

The whole program can be summarized as

Retrieve;
Eval;
Update

7.4.2. *Mail movement.* Let x be the base of the current colony under consideration. For $i = -1, 0, 1$, we will indicate how to write a subrule

$$Copy(i, loc_1, loc_2)$$

that copies, from the colony with base $x - iQ$, the location loc_1 to location loc_2 of the current colony.

Remark 7.9. In the present section, for convenience, we assume that the tracks in loc_1 and loc_2 have the same width. Similarly, we assume that the mail track is at least as wide than any of the tracks that it serves during copying. This assumption might not be valid if the transition function we are designing is separable with some small bandwidth (as in 2.2.1) since the mail field must be part of *Buf*.

Copying can be organized also without the assumption. Suppose e.g. that the mail track and loc_2 is narrow, and the track of loc_1 is wider by a factor k_1 than these. Then loc_1 can be considered to be the union of k_1 locations of smaller width and the copying of loc_1 into loc_2 will need k_1 separate copying operations. ♢

With the help of this rule, we will have, with appropriate locations $\_Retrieved_m$,

```
subrule Retrieve {
   for m ∈ {−1, 0, 1} do {
      Copy(m, _Info, _Retrieved_m)
      }}
```

here *_Info* is the location on the *Info* track containing the represented string. For the rule *Copy* we use a framework a little more general than what would be needed here, with a variable-time version in mind. Let

$$Nb\_ind = \{-1, 0, 1\}, \tag{7.8}$$

$$Mail\_ind = \{-1.1, -0.1, 0.1, 1.1\}. \tag{7.9}$$

In colony with base x, track

$$Mail_k,\ k \in Mail\_ind.$$

is for passing information to colony with base $x + \text{sign}(k)\lfloor |k| \rfloor B$, in direction $\text{sign}(k)$. Field $Mail_k$ has subfields

$$Fromaddr, Fromnb, Info, Status.$$

For simplicity, let $|Mail_k.Info| = |Info|$. The field *Status* can have the symbolic values

$$Normal, Undef.$$

The default value of $Mail_k.Status$ is *Normal*. When $Mail_k.Status = Undef$ then the other subfields of $Mail_k$ will play no role therefore we will, informally, also write $Mail_k = Undef$. For adjacent cells x, y with $j = y - x$, and their colonies x^* and y^* (defined from their *Addr* field) we define the predicate

$$Edge_j(x) = \begin{cases} 0 & \text{if } x^* = y^*, \\ 1 & \text{if } x \text{ and } y \text{ are endcells of two adjacent colonies,} \\ \infty & \text{otherwise.} \end{cases}$$

The mail track $Mail_k$ of cell x will cooperate, in direction j, with mail track $peer(k, j)$ where

$$peer(k, j) = k - jEdge_j(x)$$

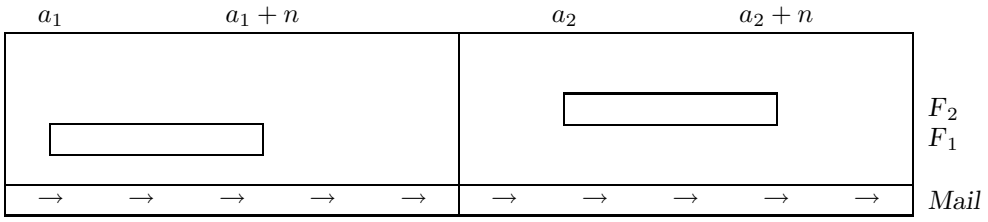

FIGURE 12. $Copy(-1, F_1([a_1, a_1 + n - 1]), F_2([a_2, a_2 + n - 1]))$

if the latter is in *Mail_ind*. For $j = -\operatorname{sign}(k)$, we define

$$Mail\_to\_receive(k) = Mail^j_{peer(k,j)}.$$

as the mail to be received into $Mail_k$ provided $peer(k, j) \in Mail\_ind$ and *Undef* otherwise. The one-step rule *Move_mail* gets mail from the neighbor cell:

```
subrule Move_mail {
   pfor k in Mail_ind do {
      Mail_k := Mail_to_receive(k)
      }}
```

Here, `pfor` is a "parallel `for`": the rule is carried out simultaneously for all k (i.e. for all fields indexed by k). A cell will typically copy the information to be sent into $Mail_k.Info$ and at the same time, its own address into $Mail_k.Fromaddr$, and $-\operatorname{sign}(k)\lfloor |k| \rfloor$ into $Mail_k.Fromnb$. In the copy rule here, $i \in \{-1, 0, 1\}$ refers to the direction of the sending colony as seen from the receiving colony. The next argument is the location of origin in the sending colony, the one following is the location to receive the information in the receiving colony.

```
subrule Copy(i, F_1([a_1, a_1 + n - 1]), F_2([a_2, a_2 + n - 1])) {
   let k = 1.1i if i ≠ 0 and 0.1 sign(a_2 − a_1) otherwise
   let l = 0.1 sign(k)
   Mail_k.Fromaddr := Addr
   ‖ Mail_k.Fromnb := −i
   ‖ Mail_k.Info := F_1;
   repeat 2Q times {
      Move_mail
      ‖ cond {
         ?  Addr ∈ [a_2, a_2 + n − 1] and
            Addr − a_2 = Mail_l.Fromaddr − a_1 and −i = Mail_l.Fromnb
         !  F_2 := Mail_l.Info
         }}}
```

Remark 7.10 (Indirectly given locations). The description $F_1([a_1, a_2])$ of a location can fit into a single field F_0 of some cell. We will allow that some argument loc_j of the rule $Copy(i, loc_1, loc_2)$ is given by a field F_0 this way. This will be done only if $i = 0$, i.e. the copying proceeds within one colony. When executing the copy rule, it will be assumed that field F_0 of each cell of the colony contains the same information loc_j. Therefore the rule can be written just as above, except that some of its parameters are read now from F_0. ♢

7.4.3. *The evaluation rule.* The subrule *Eval* controls the track $Cpt \setminus Cpt.Input$. The first steps of *Eval* write the interpreter, and the program of the transition function to be simulated into appropriate locations on the *Cpt.Input* track, followed by writing the program $Trans\_prog_2$ found as $Param_1$ after the present rules (see (7.7)):

$Write(Interpr, \_Interpr)$;
(1) $Write(Param_1, \_Prog)$.

(If $Tr_1 = Tr_2$ (self-simulation) is desired then write $Write(Param_0, \_Prog)$ in part (1) above, and $Param_1$, after it.) Then $Param_2, \dots$ will be written after $Param_1$. Next, the rule *Eval* executes

$$Copy(0, \_Retrieved_m, \_Arg_m)$$

for $m \in Nb\_ind$ where $\_Arg_m$ are locations for the arguments of the transition function on the *Cpt.Input* track. For initialization, the rule writes $* \cdots *$ to the *Output* track and the rest of the cells (including the endcells) of the *Cpt.Input* track, and 0's to the track $Cpt \setminus (Cpt.Input \cup Cpt.Output)$. Then for a sufficient number of steps, the transition function *Univ* will be applied to the *Cpt* track. According to Theorem 7.8 (Rule Language), the computation finishes in

$$(\text{interpr\_coe})(|Trans\_prog_2| + 1)^2 \|\mathbb{S}_2\|$$

steps, so this number of iterations is sufficient.

Remark 7.11. If the transition function to be simulated is separable as defined in 2.2.1 then what has been computed is not Tr_2 but the auxiliary function $Tr_2^{(w)}$. In this, the meaning of field $[3w, 4w-1]$ is an address, of a target field to be changed in the simulated cell. The evaluation rule will broadcast this number into the field *Target_addr* of each cell of the colony (it fits into a single field) to be used eventually in *Update*. ♢

The subrule

$$Update$$

copies the track *Cpt.Output* into track *Info*.

Remark 7.12. When the transition rule Tr_2 to be simulated is separable (as in 2.2.1) then the appropriate locations of *Cpt.Output* must be copied to the appropriate locations of *Info*. Some of this copying uses locations defined in an indirect way as in Remark 7.10, using the field *Target_addr* mentioned in Remark 7.11. ♢

7.4.4. *Summary in a theorem.* The encoding φ_* of a cell state v of M_2 into a colony of M_1 is defined as follows. The string v is written into *_Info*. The *Cpt* track and the mail tracks are set to all 1's. Each *Age* field is set to 0. For all i, the *Addr* field of cell i of the colony is set to i.

The theorem below states the existence of the above simulation. As a condition of this theorem, the parameters

$$Trans\_prog_2, \|\mathbb{S}_1\|, \|\mathbb{S}_2\|, Q, U \tag{7.10}$$

will be restricted by the following inequalities.

Cell Capacity Lower Bound:

$$\|\mathbb{S}_1\| \geq c_1 \lceil \log U \rceil + |Univ| + c_2$$

where c_1, c_2 can be easily computed from the following consideration. What we really need is $\|\mathbb{S}_1\| \geq |Addr| + |Age| + |Info| + |Mail| + |Cpt|$ where the following choices can be made:

$$|Info| = 2,$$
$$|Mail_i| = |Mail_i.Fromaddr| + |Mail_i.Fromnb| + |Mail_i.Info| = \lceil \log Q \rceil + 2 + 2,$$
$$|Cpt| = |Univ|,$$
$$|Addr| = |Age| = \lceil \log U \rceil.$$

Colony Size Lower Bound:

$$Q \geq (\text{interpr\_coe})(\|\mathbb{S}_2\| + |Trans\_prog_2| + \log U + 1).$$

With the field sizes as agreed above, this provides sufficient space in the colony for information storage and computation.

Work Period Lower Bound:

$$U \geq 3Q + (\text{interpr\_coe})(|Trans\_prog_2| + 1)^2 \|\mathbb{S}_2\|.$$

With the field sizes above, this allows sufficient time for the above program to be carried out.

It is not difficult to find parameters satisfying the above inequalities since $\log Q \ll Q$.

Theorem 7.13 (Basic Block Simulation)**.** *There are strings Sim_prog_0, Sim_prog_1 such that the following holds. If $Trans\_prog_2$, $\|\mathbb{S}_1\|$, $\|\mathbb{S}_2\|$, Q , U satisfy the above inequalities then*

$$Param_0 = Sim\_prog_1, \qquad Param_1 = Trans\_prog_2,$$
$$Param_2 = \|\mathbb{S}_1\|, \qquad Param_3 = \|\mathbb{S}_2\|,$$
$$Param_4 = Q, \qquad Param_5 = U$$

is a rule program of a transition function Tr_1 such that $CA(Tr_1)$ has a block simulation of $CA(Tr_2, Q, U)$. Also, if $\|\mathbb{S}_1\|, \|\mathbb{S}_2\|, Q, U$ satisfy the above inequalities with $\mathbb{S}_1 = \mathbb{S}_2$ and $Trans\_prog_2 = Sim\_prog_0$ then

$$Param_0 = Sim\_prog_0, \qquad Param_1 = \textit{empty string},$$
$$Param_2 = \|\mathbb{S}_1\|, \qquad Param_3 = \|\mathbb{S}_1\|,$$
$$Param_4 = Q, \qquad Param_5 = U$$

is a rule program of a transition function Tr_1 such that $CA(Tr_1)$ has a block simulation of $CA(Tr_1, Q, U)$.

The given construction is essentially the proof. Its complete formalization would yield Sim_prog_0 and Sim_prog_1 explicitly.

8. Robust media

This section defines a special type of one-dimensional medium called *robust*. From now on, when we talk about a medium with no qualification we will always mean a robust medium. For help in understanding the definition can be compared to the primitive variable-period media in Subsection 6.3. The media defined in the present section have some features distinguishing them from cellular automata: non-lattice cells, damage, communication among non-nearest neighbors. The need for non-lattice cells was explained in Subsection 4.3.

8.1. **Damage.** Robust media have a special set of states $Bad \subset \mathbb{S}$. For a space-time configuration η we define the *damage set*

$$Damage(\eta) = \{\, (x,t) : \eta(x,t) \in Bad \,\}.$$

For a space configuration, the damage is defined similarly. The interpretation of the set $Damage(\eta)$ is that when $(x,t) \in Damage(\eta)$ then in the neighborhood of (x,t), we will not be able to make any predictions of η, i.e. in some sense, η behaves completely "lawlessly" there. When $\eta(x,t) \in Bad$ then it is irrelevant whether we regard site x occupied by a cell or not. For a cell x a time interval I is *damage-free* if $\eta(x,\cdot)$ is not in Bad during I.

Remark 8.1. In all cellular media concerned with our *results* we could require $Damage(\eta) = \emptyset$. The damage concept is necessary only in a trajectory η_2 of a medium M_2 obtained by simulation from a trajectory η_1 of some medium M_1. Such a simulation typically requires some structure in η_1. When noise breaks down this structure the predictability of η_2 suffers and this will be signalled by the occurrence of damage in η_2.

However, for convenience, we will define damage even in the media used on the lowest level, as a violation of a certain transition rule. ♢

According to the general definition of a medium, the set of trajectories will be defined by a pair $b(\cdot), g(\cdot,\cdot,\cdot)$ where $b(\alpha)$ give the probability bound belonging to type α and $g(\alpha, W, \eta)$ is the event whose probability is bounded. We formulate these in terms of "properties". For a medium with cellsize B, let

$$V = [-B/4, B/4-] \times [-T_\bullet/4+, 0]. \tag{8.1}$$

The Computation Property constrains the kinds of events that can occur under the condition that the damage does not intersect certain larger rectangles. One condition type $\alpha(restor)$ will be called the Restoration Property: it bounds the probability of occurrence of damage in the middle of some window $(x,t) + 2V$. The Restoration Property, which depends on a new parameter ε, says that the probability of the damage is small: even if other damage occurs at the beginning of a window it will be eliminated with high probability.

Condition 8.2 (Restoration Property). Let $b(\alpha(restor)) = \varepsilon$. Further, for any rational pair (x,t) let

$$g\big(\, \alpha(restor), (x,t) + 2V, \eta \,\big)$$

be the event function for the event that there is damage in $\eta((x,t) + V)$. ♢

The Restoration axiom says that damage, i.e. the occasional obstacle to applying the Computation Property, disappears soon after its occurrence: namely, it has very small conditional probability of occurrence in the inner half of any rectangle $(x,t) + 2V$.

The axiom will be automatically satisfied on the lowest level by the property of the medium that the transition rule is only violated with small probability.

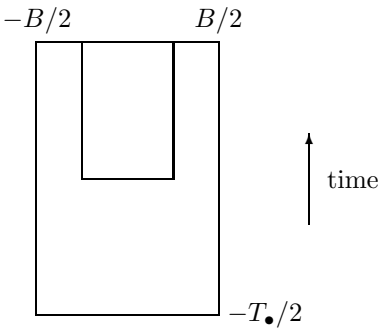

FIGURE 13. The Restoration Property

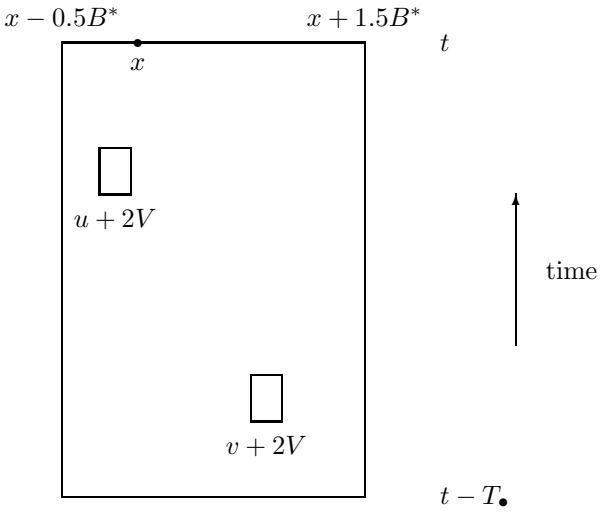

FIGURE 14. Cause for damage in point (x, t) of the simulated trajectory

Remark 8.3. In the model of [11], the restoration axiom is weaker. There, damage does not necessarily disappear in a short time but if it is contained in a certain kind of triangle at the beginning of the window then, with high probability, it is contained in a smaller triangle of the same kind later in the window. ♢

Damage helps us present a hierarchical construction as a sequence of simulations. When a large burst of faults destroys the fabric of these nested simulations, then η^{k+1} cannot be explained just in terms of the η^k from which it is decoded. The damage set will cover those lower-level details of the mess that we are not supposed to see as well as the mechanism of its removal. Damage points can be viewed as holes in the fabric of the lawful parts of a trajectory Consider a simulation $\eta^* = \Phi^*(\eta)$ between two robust media. Let us define the damage set of η^* to be used in almost all such simulations. As usual, let us mark by * the corresponding quantities in the simulated medium when some simulation is fixed. We will define $\eta^*(x, t) \in \mathit{Bad}$ iff $\mathit{Damage}(\eta)$ contains at

least two points u, v such that $u+2V$, $v+2V$ are disjoint and even $u+3V, v+3V$ are contained in $(x,t)+4V^*+(B^*/2,0)$.

The value of s, provided it is not in *Bad*, will be essentially defined by a block code φ as

$$\varphi^*(\eta(x+[0,QB-1],t)).$$

However, there will be some look-back in time for stabilization, i.e. the simulation will not be memoryless. The goal of the damage definition is to relieve us of the need of simulating a transition function if there is too much damage in the big rectangle $(x,t)+4V^*+(B^*/2,0)$.

Lemma 8.4 (Simulation Damage Probability Bound)**.** *Let M, M^* be media with parameters ε, ε^* whose local condition system is defined by the Restoration Property. Let a simulation Φ^* be defined between them which assigns damage in M^* according to the above damage map. If*

$$\varepsilon^* \geq 25((B^*/B)(T_\bullet{}^*/T_\bullet)\varepsilon)^2$$

then, Φ^* is a deterministic canonical simulation map as defined in 6.2.1.

This lemma says that small bursts of damage that are not too close to each other behave in the medium M as if they were independent, therefore the probability of the occurrence of two such bursts can be estimated by ε^2 times the number of such pairs in a rectangle V^*.

Proof. It is enough to show an expression of the form (6.12) for local conditions of type *restor* in M^*. Let x_1, t_1 be the sizes of the space and time projections of V. We define the following lattice of points:

$$\mathcal{V} = \{\, (ix_1, jt_1) : i,j \in \mathbb{Z} \,\}.$$

Then the rectangles $v+V$ for $v \in \mathcal{V}$ form a partition of the space-time. For each u in space-time, let $v(u)$ be the $v \in \mathcal{V}$ with $u \in v+V$. Let η be a trajectory of M. It is enough to consider the rectangle $2V^*$. If *Damage*(η^*) intersects V^* then there are some points u_1, u_2 in *Damage*(η) such that u_i+2V are disjoint and even u_i+3V are in $5V^*+(B^*/2,0)$. Then it is easy to see that even $v(u_i)+2V$ are disjoint from each other and are contained in $5V^*+(B^*/2,0)$. Let n be the number of pairs v_1, v_2 in $\mathcal{V}$ with this property. We found n pairs of rectangles $u_{i,j}+2V$ ($i=1,\dots,n$, $j=1,2$) such that $u_{i,1}+2V \cap u_{i,2}+2V = \emptyset$, and that if *Damage*$(\eta^*)$ intersects V^* then there is an i such that *Damage*(η) intersects $u_{i,j}+V$ for $j=1,2$. We found

$$g^*(\mathit{restor}, 2V^*, \eta^*) \leq \sum_i \prod_j g\big(\mathit{restor}, u_{i,j}+2V, \eta\big).$$

To complete the proof, observe that

$$\sum_i \prod_j b(\mathit{restor}) \leq n\varepsilon^2$$

Since counting shows $n < 25((B^*/B)(T_\bullet{}^*/T_\bullet))^2$ and the assumption of the lemma says $25((B^*/B)(T_\bullet{}^*/T_\bullet))^2\varepsilon^2 < \varepsilon^*$ both conditions of a deterministic canonical simulation are satisfied. □

8.2. **Computation.**

8.2.1. *Neighborhood structure.* Before giving the Computation Property, let us introduce some details of the structure of a robust medium. Cells sometimes have to be erased (killed) in a trajectory since cells created by the damage may not be aligned with the original ones. At other times, the creation of a live cell at a vacant site will be required. Most of the trajectory requirements of a robust medium will be expressed by a transition function Tr, desribed later. Killing and creation may be indicated by some special values of this function.

Although we permit non-lattice configurations we could still use a transition function that restricts the new state of a cell only as a function of adjacent cells. It is, however, convenient to allow nearby cells to see each other regardless of whether they are on the same lattice or not. We extend the notation $\vartheta_j(x)$ introduced in Subsection 7.3. Let us fix η. For a vacant site x at time t there is at most one cell at distance greater than 0 but less than B to the right of x: if it exists and there is no damage between it and x then we denote it by $\vartheta_{0.5}(x,t,\eta)$. Otherwise, $\vartheta_{0.5}(x,t,\eta)$ is undefined. The notation $\vartheta_{-0.5}$ is defined analogously. We will omit η, and sometimes even t, from the arguments when it is obvious from the context. By convention, whenever $\vartheta_j(x,t,\eta)$ is undefined then let

$$\eta(\vartheta_j(x,t,\eta),t) = \mathit{Vac}.$$

For a (nonvacant) cell x there is at most one cell at distance greater than B but smaller than $2B$ to the right of x: if it exists and there is no damage between it and x then we denote it by

$$\vartheta_{1.5}(x).$$

Otherwise, this value is undefined. The corresponding cell to the left is denoted by $\vartheta_{-1.5}(x)$. The medium depends on a parameter r called the *reach*, determining the maxiumum distance of the neighbors that a cell is allowed to depend on directly. (We will only use two possible reaches: $r = 1$ and $r = 3$.) Let

$$\mathit{Nb\_ind}_r = \{-1.5, 1.5\} \cup \{-r, -r+1, \ldots, r-1, r\}. \tag{8.2}$$

A function $\mathbf{u}\colon \mathit{Nb\_ind}_r \to \mathbb{S}$ will be called an *assignment*. For an assignment $\mathbf{u}$, we will write u_j for the value of $\mathbf{u}$ on j. The transition function in a robust medium has the form $Tr(\mathbf{u})$ where $\mathbf{u}$ runs over assignments. In analogy with (2.3), let

$$\overline{Tr}(\eta, x, t) = Tr(\mathbf{u})$$

where $\mathbf{u}$ is the assignment defined by $u_j = \eta(\vartheta_j, t)$ with $\vartheta_j = \vartheta_j(x,t,\eta)$ except that if ϑ_j is defined for some noninteger j, then then $u_{\lambda j} = \mathit{Vac}$ for all $\lambda > 1$. Thus, when a closest neighbor ϑ_j is not adjacent then the transition function does not depend on any farther neighbors in that direction. η will be omitted from $Tr(\eta, x, t)$ when it is obvious from the context.

8.2.2. *Properties of the transition function.* A robust medium will be denoted by

$$\mathit{Rob}(Tr^{(w)}, B, T_\bullet, T^\bullet, \varepsilon, \varepsilon', r).$$

The set $\mathbb{S}$ of states is implicit in the transition function. Similarly, the subset *Bad* of states will be defined implicity by the transition function: this will be the states such that if any argument of the transition function is in this set, the function value is not defined. The first bit of the state of a cell will be called *Rand* and we write $\mathit{Det} = \mathit{All} \setminus \mathit{Rand}$. There is also a subfield $\mathit{Color} \subset \mathit{Det}$ defined implicitly by the transition function: this is, say, the set of those first bits of *Det* that turn to 0 when *Tr* is applied to the vector with all *Vac* values. (Color plays role in self-organization.)

The transition function, a mapping from assignments to $\mathbb{S}$, describes the goal for the deterministic part *s.Det* of the value s of the state after transition. We will see that the field *Color* will not necessarily obey the transition function. We will assume that *Tr* is separable in the sense of 2.2.1 with some bandwidth w. Let the fields *Rand* and *Color* be part of *Inbuf*: thus, they can be seen by a neighbor cell in a single step. The separability requirement also defines a predicate $\mathit{legal}^{Tr}(u, v)$.

Our transition functions in robust media will also have the following properties:

Condition 8.5 (Time Marking). If $s \neq Vac$ then $legal(s, s) = 0$. ♢

This condition is similar to (6.17) and says that a nonvacant cell always tries to change its state. It is necessary for nontrivial media with a strict dwell period upper bound.

Let us call a pair $(j, v) \in \{-1, 1\} \times \mathbb{S}$, a *potential creator* of state $s \neq Vac$ if $s = Tr(\mathbf{u})$ provided $u_i = v$ for $i = j$ and Vac otherwise. In a space-time configuration η, at a switching time σ of cell x when $\eta(x, t)$ turns from vacant to non-vacant, we call $y = \vartheta_j(x, \sigma-)$ a *creator* of x for time σ if $(j, \eta(\vartheta_j(x, \sigma-)))$ is a potential creator of $\eta(x, \sigma)$ and further $\eta(x, t) = Vac$, $\eta(y, t) = \eta(y, \sigma-)$ for $t \in [\sigma - T_\bullet/2+, \sigma-]$.

Condition 8.6 (Creation).
Let $\mathbf{u}$ be an assignment such that $u_0 = Vac$, $Tr(\mathbf{u}).Det \neq Vac.Det$ and in which there is a j in $\{-1, 1\}$ with $u_j \neq Vac$. Then there is a $j \in \{-1, 1\}$ such (j, u_j) is a potential creator of $Tr(\mathbf{u})$. ♢

This condition says that if, for some $t_1 < t_2$, a vacant cell x comes to life at time t_2 due to "observation" at some time t_1 when there is a neighbor $\vartheta_j(x, t_1)$ nearby then an adjacent neighbor at time t_1 can be made completely responsible for the state of the new cell at time t_2: the state would be the same even if all other sites in the neighborhood were vacant.

If $\mathbf{u}$ is the neighborhood state assignment vector consisting of all vacant states then let $Newborn = Tr(\mathbf{u})$ be the *canonical newborn state*. A state s is a *newborn state* if

$$s.(Det \setminus Color) = Newborn.(Det \setminus Color).$$

Note that a newborn state is allowed to be somewhat less determined than others: this property will be used in self-organization.

Condition 8.7 (Cling to Life). If $Tr(\eta, x, t) = Vac$ then $\eta(\vartheta_j(x, t), t) \neq Vac$ for some $j \in \{-1.5, 1.5\}$. ♢

Thus a cell will only be erased if it may disturb a close non-aligned neighbor.

8.2.3. *The computation axiom.* The condition called the Computation Property has a form similar to the definition of primitive variable-period media in Subsection 6.3. For a space-time configuration η, cell x and rational number $a > 0$ let

$$\sigma_1, \sigma_2, \sigma_0$$

be defined as follows. σ_1, σ_2 are the first two switching times of x after a but defined only if $\eta(x, \sigma_1) \neq Vac$. On the other hand, σ_0 is the first switching time of x after $a + T^\bullet$ but defined only if $\eta(x, a + T^\bullet) = Vac$. Whenever we have an event function $g(\alpha, W, \eta)$ in whose definition σ_2 occurs, this function is always understood to have value 0 if σ_2 is not defined (similarly with σ_0). Let

$$\begin{aligned}
W_0(x, a) &= \{x\} \times [a - T_\bullet/2+, a + 2T^\bullet], \\
W_1(x, a) &= [x - (r + 1)B, x + (r + 2)B-] \times [a - T_\bullet/2+, a + 2T^\bullet], \\
f_j(x, a, \eta) &= \text{the event function for } \{\, Damage(\eta) \cap W_j(x, a) = \emptyset \,\} \qquad (j = 0, 1).
\end{aligned}$$

Condition 8.8 (Computation Property). This axiom consists of several condition types. For each type α used in this axiom except $\alpha(rand, j)$, we have $b(\alpha) = 0$, i.e. the corresponding events $g(\alpha(), W, \eta)$ are simply prohibited in the trajectory. On the other hand,

$$b(\alpha(rand, j)) = 0.5 + \varepsilon' \quad (j = 0, 1).$$

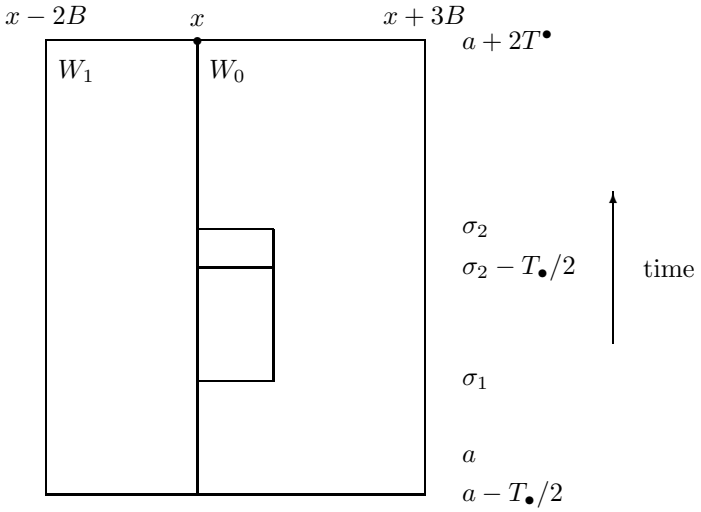

FIGURE 15. To the computation axiom. The large rectangle is $W_1(x,a)$. The rectangle of the same height but width 0 at point x is $W_0(x,a)$. The small rectangle between times σ_1 and σ_2 is the cell work period under consideration. Its lower part is the observation period.

(a) This condition requires coin-tossing, for $j = 0, 1$:

$$g\big(\,\alpha(\mathit{rand}, j),\ W_0(x,a),\ \eta\,\big) = f_0(x,a,\eta)\{\eta(x,\sigma_2).\mathit{Rand} = j\}.$$

(b) This prohibits dwell periods shorter than $T_\bullet$ or longer than $T^\bullet$. Let

$$g\big(\,\mathit{dw\_p\_bd},\ W_0(x,a),\ \eta\,\big) = f_0(x,a,\eta)h(\mathit{dw\_p\_bd}, x, a, \eta)$$

where $h(\mathit{dw\_p\_bd}, x, a) = 1$ if η has a dwell period shorter than $T_\bullet$ in $W_0(x,a)$ or has a dwell period longer than $T^\bullet$ there (and, as always, 0 otherwise).

(c) This says that whenever $W_0(x,a)$ is damage-free the transition at σ_2 is a legal one. Let

$$g\big(\,\mathit{legal\_comp},\ W_0(x,a),\ \eta\,\big) = f_0(x,a,\eta)(1 - \mathit{legal}(\eta(x,\sigma_2-), \eta(x,\sigma_2))).$$

(d) This says that whenever $W_0(x,a)$ is damage-free the transition in σ_0 is a legal one. Let

$$g\big(\,\mathit{legal\_birth},\ W_0(x,a),\ \eta\,\big) = f_0(x,a,\eta)(1 - \mathit{legal}(\eta(x,\sigma_0-), \eta(x,\sigma_0))).$$

(e) This says that whenever $W_1(x,a)$ is damage-free the transition function applies, at least as much as it can, based on observation at a certain time during the observation period ("atomicity"). Let

$$g\big(\,\mathit{trans},\ W_1(x,a),\ \eta\,\big) = f_1(x,a,\eta)h(\mathit{trans}, x, a, \eta)$$

where $h(\mathit{trans}, x, a, \eta) = 1$ unless there is a $t' \in [\sigma_1+, \sigma_2 - T_\bullet/2]$ with

$$\eta(x,\sigma_2).\mathit{Det} = \mathrm{Tr}(\eta, x, t').\mathit{Det}.$$

(f) This says that if x always has a neighbor during the whole interval of interest before its birth then it has a creator. Let

$$g\big(\,\mathit{creator},\ W_1(x,a),\ \eta\,\big) = f_1(x,a,\eta)h(\mathit{creator}, x, a, \eta)$$

where $h(\mathit{creator}, x, a, \eta) = 1$ unless either x has a creator for time σ_0 or there is a $t' \in [a+, \sigma_0]$ such that $\eta(\vartheta_j(x, t'), t') = \mathit{Vac}$ for all $j \in \{-1, 1\}$.

(g) This says that if x does not have a creator accounting for its birth then it is a newborn with vacant neighbors at some time shortly before birth. Let

$$g\big(\, \mathit{newborn},\ W_1(x, a),\ \eta \,\big) = f_1(x, a, \eta) h(\mathit{newborn}, x, a, \eta)$$

where $h(\mathit{newborn}, x, a, \eta) = 1$ if x has no creator for time σ_0, and either $\eta(x, \sigma_0)$ is not newborn state or there is no t in $[\sigma_0 - T_\bullet/2, \sigma_0-]$ with $\eta(\vartheta_j(x, t), t) = \mathit{Vac}$ for all $j \in \mathit{Nb\_ind}$.

(h) This says that x cannot stay vacant if it has would-be creators for a long time and has no neighbor blocking creation. Let

$$g\big(\, \mathit{no\_birth},\ W_1(x, a),\ \eta \,\big) = f_1(x, a, \eta) h(\mathit{no\_birth}, x, a, \eta)$$

where $h(\mathit{no\_birth}, x, a, \eta) = 1$ if

(a) $\eta(x, t) = \mathit{Vac}$ for all $t \in [a+, a + 2T^\bullet]$;

(b) for each $t \in [a+, a + 2T^\bullet]$ there is a $j \in \{-1, 1\}$ such that $(j, \vartheta_j(x, t))$ is a potential creator of some (non-vacant) state;

(c) there is no (y, t) with $0 < |y - x| < B$, $t \in [a+, a + 2T^\bullet]$, $\eta(y, t) \neq \mathit{Vac}$;

♢

Example 8.9 (Special cases). To obtain a constant-period medium as a special case of robust media set $T_\bullet = T^\bullet$, $r = 1$. To obtain a deterministic cellular automaton, set also $\varepsilon = 0$, require that the space-time configurations have empty damage and that the transition function does not give a vacant value.

The connection between primitive variable-period media and robust media will be set up using a trivial simulation. Let $M_1 = \mathit{Prim\_var}(Tr, B, T_\bullet, T^\bullet, \varepsilon)$. We define a simple simulation Φ^* by this medium of the robust medium

$$M_2 = \mathit{Rob}(Tr, B, T_\bullet, T^\bullet, \varepsilon, \varepsilon, 1).$$

M_2 has almost the same state set as M_1 except that it also has at least one extra state, making the set *Bad* nonempty. We give the definition of $s = \eta^*(x, t) = \Phi^*(\eta)(x, t)$ for each η. If x is not an integer multiple of B then $s = \mathit{Vac}$. Else $s = \eta(x, t)$ unless η violates the Computation Property of M_2 at x with $\sigma_2 = t$: in that case, it is in *Bad*. It can be verified that this is indeed a simulation. ♢

8.2.4. *Error-correction.* The error-correction property can be defined here just as in 3.6.2, replacing T_i with $T_{\bullet i}$. Then a lemma analogous to Lemma 3.7 can be formulated, and a proof for Theorem 6.13 can be provided that is analogous to the proof of Theorem 2.5.

8.3. **Simulating a medium with a larger reach.**

Theorem 8.10 (Reach Extension)**.** *For all r and all Tr_2 with reach r, for all $T_\bullet \leq T^\bullet$ let $\lambda = \lceil T^\bullet/T_\bullet \rceil$,*

$$U = 64\lambda((\lambda + 1)r + 1)(r + 2).$$

There is a set $\mathbb{S}_1$, a one-to-one mapping $s \mapsto s_*$ from $\mathbb{S}_2$ to $\mathbb{S}_1$ giving rise to the code $\varphi_*(\xi)(x) = (\xi(x))_*$, functions $\mathrm{Tr}_1()$ of reach 1, and for all $B, \varepsilon, \varepsilon'$, a decoding Φ^* such that (Φ^*, φ_*) is a simulation of

$$M_2 = \mathit{Rob}(\mathrm{Tr}_2, B, UT_\bullet, UT^\bullet, U\varepsilon, UR\varepsilon', r)$$

by

$$M_1 = \mathit{Rob}(Tr_1, B, T_\bullet, T^\bullet, \varepsilon, \varepsilon', 1).$$

The basic idea is of the proof is to replicate the state of a whole reach-r neighborhood of a cell of M_2 in a single cell of M_1. It will take then a work period size U that is at least proportional to r for a cell of M_1 to simulate a step of a cell of M_2 since the cell must learn about neighbors r steps away. In fact, due to asynchrony, the work period will be significantly larger. Some complication is due to the need to achieve the "atomicity" property, namely that a single observation time can be assigned to a transition of M_1 even though the observation occurs over an extended period of time. The solution for this is to make sure that the observed values coming from the neighbor cells do not come from near the end of the work period of those cells.

9. Amplifiers

In the present section, when we talk about media without qualification we understand robust media.

9.1. **Amplifier frames.** The eventual goal is to find an amplifier (M_k, Φ_k) with a fast decreasing sequence ε_k and we also want M_k to simulate a computation with transition function *Univ* and damage probability bound ε_k. In this subsection, we impose some conditions on the parameters of a sequence (M_k) of media that are sufficient for the existence of an amplifier. These conditions are similar to the inequalities in the conditions of Theorem 7.13.

9.1.1. *The rider fields.* All robust media M_k have, by definition, the fields *Det* and *Rand* whose behavior is governed by the computation and restoration properties. Only *Det* will be subdivided into subfields: therefore subfields $\eta.Det.F$ will be denoted simply as $\eta.F$, without danger of confusion.

In the media M^k we discuss further, there will be a shared field called $Rider^k$ with guard field $Guard^k$ (see Subsection 4.2). The useful computation of M_k will be carried out on the "active level" k, on the track $Rider^k$. The simulations will have sufficient freedom in the rule governing $Rider^k$ on the active level to regain the universality lost by having no "program" for the whole simulation (as discussed in Subsection 4.3). We will say that in a robust medium M, the transition function Tr, with state set $\mathbb{S}$, is *combined* from the rider transition function *Rd_trans* and simulation transition function *Sim_trans* with fields *Rider* and *Guard*, if the value $s = Tr(\mathbf{r})$ is defined in the following way. If $Sim\_trans(\mathbf{r}) = Vac$ then $s = Vac$. Otherwise, all fields of s disjoint from $s.(Rider \cup Guard)$ are defined by $Sim\_trans(\mathbf{r})$. Further

$$
\begin{aligned}
s.Rider &= \begin{cases} Rd\_trans(\mathbf{r}).Rider & \text{if } r_0.Guard = 0, \\ Sim\_trans(\mathbf{r}).Rider & \text{otherwise,} \end{cases} \\
s.Guard &= \begin{cases} Rd\_trans(\mathbf{r}).Guard & \text{if } r_0.Guard \geq 0, \\ Sim\_trans(\mathbf{r}).Guard & \text{otherwise.} \end{cases}
\end{aligned} \tag{9.1}
$$

Thus, the rider field is controlled by its own transition function only when the simulation does not command to eliminate the cell, and when we are on the active level according to the guard field. The guard field will be controlled by its own transition function in case we are not below the active level (where the simulation transition function will enforce the broadcast property for it).

9.1.2. *Broadcast amplifier frames.* The string *prog* will be called a *uniform program (with complexity coefficient c)* for the functions f_k if there is some constant c such that it computes on the universal computing medium *Univ* the value $f_k(\mathbf{r})$ from $k, \mathbf{r}$ with space- and time-complexities bounded by $c(\log k + |\mathbf{r}|)$ where $|\mathbf{r}|$ is the total number of bits in the arguments $\mathbf{r}$. If a sequence (f_k) of functions has a uniform program it is called *uniform*. As a special case, a sequence of constants c_k is uniform if it is computable with space- and time complexities $c \log k$ from the index k.

Here, and always from hence, when we refer to a natural number k as a binary string we always mean its standard binary representation. We will distinguish the simpler notion of a "broadcast amplifier frame" from the full-featured "amplifier frame". The simpler notion is sufficient for the basic non-ergodicity result. A broadcast amplifier frame *Frame* is given by sequences

$$Cap_k, Q_k, U_k,$$

and the constant R_0 satisfying certain conditions. Here, Q_k is the number of cells in a colony. For an appropriate sequence $\nu_k \geq 1$, the number of dwell periods in a work period will be allowed to

vary between U_k/ν_k and $U_k\nu_k$. Given a frame *Frame*, let us define some additional parameters as follows, with some arbitrary positive initial constants ${T^\bullet}_1 \geq T_{\bullet 1}$ and $\varepsilon = \varepsilon_1, \varepsilon'_1 < 1/R_0^2$.

$$\begin{aligned}
\mathbb{S}_k &= \{0,1\}^{Cap_k},\\
B_k &= \prod_{i<k} Q_i,\\
\nu_k &= 1 + R_0 Q_k/U_k \text{ if } T_{\bullet 1} \neq {T^\bullet}_1, \text{ and 1 otherwise.}\\
T_{\bullet k} &= T_{\bullet 1} \prod_{i=1}^{k-1} U_i/\nu_i,\\
{T^\bullet}_k &= {T^\bullet}_1 \prod_{i=1}^{k-1} U_i\nu_i,\\
\varepsilon_{k+1} &= 25(Q_k U_k \nu_k \varepsilon_k)^2,\\
\varepsilon''_k &= 4 Q_k U_k \nu_k \varepsilon_k,\\
\varepsilon'_k &= \varepsilon'_1 + \sum_{i=1}^{k-1} \varepsilon''_i.
\end{aligned} \tag{9.2}$$

The formula for ε_{k+1} is natural in view of Lemma 8.4. The definition of ε'_{k+1} takes into account the limited ability to simulate a coin-toss with the help of other coin-tosses. ε''_k is used for the error-correction property: it is essentially the probability that there is any k-level damage at all during the work period of a colony of k-cells. The requirements below defining an amplifier frame can be compared to the corresponding conditions for Theorem 7.13.

An object *Frame* given by the above ingredients is a *broadcast amplifier frame* if the conditions listed below hold.

Complexity Upper Bounds: All parameters in *Frame* are uniform sequences with complexity coefficient R_0;

Bandwidth Lower Bound: We should be able to deal with numbers comparable to the size of the colony and the work period within a cell:

$$Cap_k \geq R_0 \log U_k; \tag{9.3}$$

Capacity Lower Bound: The colony must represent the state of the big cell with redundancy:

$$R_0 Cap_{k+1} \leq Q_k Cap_k. \tag{9.4}$$

Work Period Lower Bound: There must be enough dwell periods in a work period to perform the necessary computations of a simulation, with some repetitions:

$$U_k \geq R_0(\log Q_k + \log k) Q_k. \tag{9.5}$$

The factor $\log Q_k + \log k$ is needed only for technical reasons, in the proof of self-organization.

Error Upper Bound: The following upper bound on $Q_k U_k$ implies, in view of the definition of the error probabilities, that these decrease exponentially:

$$\varepsilon_k^{0.2} \leq R_0/Q_k U_k. \tag{9.6}$$

Time Stability:

$$\frac{{T^\bullet}_k}{T_{\bullet k}} \leq 3. \tag{9.7}$$

This inequality (which is made this strong for simplicity) can be achieved e.g. with $U_k \geq ck^2 Q_k$ for some sufficiently large c.

The definition of ε_{k+1} and (9.7), implies

$$\varepsilon_{k+1} = 25(Q_k {T^\bullet}_{k+1}/{T^\bullet}_k)^2 \varepsilon_k^2 \leq 25(2Q_k U_k)^2 \varepsilon_k^2 \leq 32\varepsilon_k^{1.6}/(R_0)^2 \leq \varepsilon_k^{1.5}$$

hence

$$\varepsilon_k \leq \varepsilon^{1.5^{k-1}}. \tag{9.8}$$

From here, it is easy to see that ε_k'' also converges to 0 with similar speed.

Example 9.1. Let us choose for some $c \geq 1$

$$\begin{aligned} {T_\bullet}_1 &= 1, \\ {T^\bullet}_1 &= 2, \\ Q_k &= c^2 k^2, \\ U_k &= c^3 k^4, \\ Cap_k &= c(\log k \vee 1). \end{aligned} \tag{9.9}$$

Conditions (9.3) and (9.5) are satisfied for large c.

There is a slightly better bound on ε_k here than in (9.8).

$$\varepsilon_{k+1} \leq 25(2Q_k U_k)^2 \varepsilon_k^2 = 32c^{10}k^{12}\varepsilon_k^2. \tag{9.10}$$

Let us prove, by induction, for small enough ε,

$$\varepsilon_k \leq \varepsilon^{2^{k-2}+(k+1)/4}. \tag{9.11}$$

For $k = 1$, the statement gives $\varepsilon \leq \varepsilon$. For $k > 1$, using (9.10) and the inductive assumption,

$$\varepsilon_{k+1} \leq \varepsilon^{2^{k-1}+(k+1)/2} 32c^{10}k^{12} = \varepsilon^{2^{k-1}+(k+2)/4}(32\varepsilon^{k/4}c^{10}k^{12}).$$

For small enough ε, the last factor is less than 1. ♢

9.1.3. *Amplifier frames.* The above definitions must be modified for the notion of an amplifier frame, (not needed for the basic non-ergodicity result). An amplifier frame *Frame* is given by the same sequences as a broadcast amplifier frame, and some additional sequences

$$q_k, w_k, Rd\_trans_k(), Output^k, R_1(k).$$

Here, $Rd\_trans_k()$ describes a rider transition function with bandwidth $\leq w_k$, the field $Rider^k$ has subfield $Output^k$ on which this function has monotonic output, and the number $R_1(k)$ is a redundancy coefficient. Each cell will have space

$$B_k' = \prod_{i<k} q_i$$

for the represented information. An object *Frame* given by the above ingredients is an *amplifier frame* if the conditions listed below hold.

Complexity Upper Bounds: The sequence $Rd\_trans_k^{(w_k)}()$ of functions (see (2.5)) and all other parameters in *Frame* are uniform sequences with complexity coefficient R_0;

Bandwidth Lower Bound: We should be able to deal with numbers comparable to the size of the colony and the work period within the time and space bound w_k:

$$w_k \geq \log U_k; \tag{9.12}$$

Redundancy Lower Bound: The redundancy must help duplicate the content of at least a certain constant number of cells:

$$R_1(k) \geq R_0/Q_k. \tag{9.13}$$

Capacity Lower Bounds: The capacity Cap_k of a cell should accomodate all the original information $2B'_k$ (2 bits per primitive cell).

$$2B'_k \leq Cap_k. \tag{9.14}$$

The colony must represent the state of the big cell with redundancy. It also needs extra space for computing with a few bandwidths of information of the big cell.

$$(1 + R_1(k))Cap_{k+1} + R_0 w_{k+1} \leq Q_k Cap_k. \tag{9.15}$$

Work Period Lower Bound: There must be enough dwell periods in a work period to perform the necessary computations of a simulation:

$$U_k \geq R_0 Q_k (\frac{Cap_k}{w_k} + \log Q_k + \log k) \tag{9.16}$$

It costs $QCap_k/w_k$ to perform Q steps of simulated computation of *Univ* when each track of size w_k must be worked separately. The other terms in the parentheses are typically of lower order and are needed only for technical reasons, in the proof of self-organization.

Error upper bound, Time stability: These conditions are the same as the ones for broadcast amplifier frames.

The inequality (9.15) has the following consequences, similar to (4.32):

Lemma 9.2.

$$\sum_k R_1(k) \frac{Cap_{k+1}}{B_{k+1}} < \infty,$$
$$\sum_k \frac{w_{k+1}}{B_{k+1}} < \infty.$$

Thus, as expected, if we have constant space redundancy, i.e. Cap_{k+1}/B_{k+1} is bounded away from 0 then $\sum_k R_1(k) < \infty$, i.e. the redundancies of each level form a converging series. This will then impose, via (9.16), a lower bound on the time redundancy achievable by these amplifiers.

Proof. Inequality (9.15) can be rearranged as

$$R_1(k) \frac{Cap_{k+1}}{B_{k+1}} + R_0 \frac{w_{k+1}}{B_{k+1}} \leq \frac{Cap_k}{B_k} - \frac{Cap_{k+1}}{B_{k+1}}. \tag{9.17}$$

□

Let us remind the reader that the constructions involving the Rider and Guard fields are not needed for the simple non-ergodicity results.

Example 9.3. Let us choose for some $c \geq 1$

$$\begin{aligned} T_{\bullet 1} &= 1, \\ T^{\bullet}{}_1 &= 2, \\ Q_k &= c^2k^2, \\ q_k &= Q_k, \\ R_1(k) &= c/Q_k = 1/(ck^2), \\ U_k &= c^3k^4 = cQ_k/R_1(k), \\ w_k &= R_1(k)B_k, \\ Cap_k &= 2B_k(1+1/k). \end{aligned} \tag{9.18}$$

The transition function Rd_trans_k will be the aggregated function $Tr^{w_k}_{Univ}$ as defined in Example 3.4. This satisfies the required complexity bounds. Conditions (9.12) and (9.13) are satisfied with large c. Condition (9.14) is satisfied by the definition of Cap_k. It is easy to check that (9.15) also holds with large c. Condition (9.16) will be satisfied with large c. The error upper bound is satisfied just as for the broadcast amplifier example, and we have (9.11) again.

The parameters of this example are chosen for constant space redundancy, at the price of time redundancy that comes as an extra factor of Cap_k/w_k at every level.

For later reference, here is an explicit expression falling between $T_{\bullet k}$ and $T^{\bullet}{}_k$:

$$\prod_{i<k} U_i = c^{3(k-1)}((k-1)!)^4 = e^{4k\log k + O(k)}. \tag{9.19}$$

The expression for B_k is similar. ♢

9.2. **The existence of amplifiers.** We give a separate, simpler track of definitions for broadcast amplifiers. A *(uniform) broadcast amplifier* is given by a broadcast amplifier frame *Frame*, uniform sequences (Sim_trans_k), $(M_k, \Phi_k)_{k\geq 1}$ with $\Phi_k = (\Phi^*_k, \varphi_{k*})$ as defined in Subsection 3.6 and a uniform sequence of codes φ_{k**} with a sequence of fields (F^k) such that

(a)

$$M_k = Rob(Tr_k, B_k, T_{\bullet k}, T^{\bullet}{}_k, \varepsilon_k, \varepsilon'_k); \tag{9.20}$$

(b) (F^k) forms a broadcast field for both (φ_{k*}) and (φ_{k**}), as defined in Subsection 4.2;
(c) Simulation Φ_k has ε''_k-error-correction for φ_{k**};
(d) The damage map of the simulation Φ_k is defined as in Subsection 8.1.

(The last condition explains the definition of ε_{k+1}.) With Lemma 8.4, it will imply the Restoration Property for the simulated trajectory.

The following definition is not needed for the simple non-ergodicity result: A *(uniform) amplifier* is given by an amplifier frame *Frame*, uniform sequences (Sim_trans_k), $(Rider^k)$, $(Guard^k)$, a hierarchical code

$$\Psi = (\mathbb{S}_k, Q_k, \varphi_{k*}, Rider^k, q_k, \gamma_k, a_k)_{k\geq 1} \tag{9.21}$$

as in 4.13 and an amplifier $(M_k, \Phi_k)_{k\geq 0}$ where $\Phi_k = (\Phi^*_k, \varphi_{k*})$ is defined as in Subsection 3.6, and a sequence of codes φ_{k**} and fields $Guard^k$ such that

(a) $(Rider^k)$, and $(Guard^k)$ form a guarded shared field for (φ_{k*}), (φ_{k**}), as defined in Subsection 4.2;
(b) the damage map of the simulation Φ_k is defined as in Subsection 8.1;
(c) Tr_k is combined from Rd_trans_k and Sim_trans_k, with fields $Rider^k$, $Guard^k$;

(d) M_k is as in (9.20);
(e) Φ_k has ε''_k-error-correction for φ_{k**};

Most of our effort will be devoted to proving the following lemma.

Lemma 9.4 (Amplifier)**.** *Every (broadcast) amplifier frame Frame with large enough R_0 can be completed to a uniform (broadcast) amplifier.*

9.3. **The application of amplifiers.**

Proof of Lemma 3.7. We will actually prove the more general variable-period variant of the lemma, with $T_k = T_{\bullet k}$.

Let us define a broadcast amplifier frame *Frame* e.g. as in Example 9.1. Applying the Amplifier Lemma in the broadcast case, we obtain a broadcast amplifier with media M_k as in (9.20) with a hierarchical code having a broadcast field (F^k) for both (φ_{k*}) and (φ_{k**}). Let us denote the whole system of code and the broadcast field (with the additional mappings γ_k) by Ψ as in (4.11). For any value u_1 of the field F^1, we create an initial configuration $\eta^1(\cdot, 0) = \Gamma(u_1; \Psi)$ as in (4.12).

Then all properties but the initial stability property of an abstract amplifier (defined in Lemma 3.7) are satisfied by definition. For the initial stability property it is sufficient to note that each medium M_k is a robust medium, with work period lower bound $T_{\bullet k}$. Therefore, if η^k is a trajectory of M_k and $t < T_{\bullet k}$ then for each x the probability that $\eta^k(x,t) \neq \eta^k(x,0)$ is less than the probability that damage occurs in $\{x\} \times [0, T_{\bullet k}]$. This can be bounded by the Restoration Property. □

Proof of Theorem 6.14.

1. This proof starts analogously to the proof of Lemma 3.7.

   Let us define an amplifier frame *Frame* as in Example 9.3, with the rider transition function having the property

   $$Rd\_trans_k(\mathbf{r}) = r_0,$$

   i.e leaving all fields (in particular, the rider and guard fields) unchanged. Applying the Amplifier Lemma, we obtain a uniform amplifier with media M_k as in (9.20) with a hierarchical code having a guarded shared field $Rider^k$ with guard field $Guard^k$ as in (9.21). Let K be largest with $B_K \leq N$. For any infinite configuration $\varrho \in (\mathbb{S}_1.Rider^1)^{\mathbb{Z}}$, we create an initial configuration $\eta^1(\cdot, 0) = \Gamma(\varrho)$ with a guarded shared field $(F^k, Guard^k)$, with active level K. As mentioned after (4.30), if N is finite then $\Gamma(\varrho) = \Gamma(\varrho; K)$. Let the trajectories η^1, η^k be defined as before. Then we will have $\eta^k(x,0).Guard^k = -1$ for all x, for $k < K$ and $\eta^K(x,0).Guard^K = 0$.

   Let $x_1 = X(y)$ for a y in $Visible(K(N), N)$ (since the code was chosen such that $|Visible(K(N), N)| = N$, this is not really a restriction on y), and let $t_1 \geq 0$. Remember the definition of the aggregated input configurations ϱ^k in (4.21) and of $X(y, i; k)$ in (4.19). Eventually, we want to estimate the probability of $\varrho(y) = \eta^1(X(y), t_1).Rider^1$ but we will use a generalization corresponding to (4.26). Let

   $$x_k = o_k + X(y,k),\ x'_k = o'_k + X'(y,k)$$

   $1 \leq k \leq K$, then (4.20) shows that x_{k+1} is a cell of $\eta^{k+1}(\cdot, 0)$ whose body contains x_k with address y'_k where y'_k was defined before (4.19). Let $\mathcal{F}_k$ be the event that

   $$\eta^k(x_k, t).Rider^k = \varrho^k(x'_k) \tag{9.22}$$

   and $\eta^k(x_k, t).Guard^k = -1$ holds for t in $[t_1 - T_{\bullet k}/3, t_1]$.
2. We have $\mathrm{Prob}\{\, \mathcal{F}_{k+1} \cap \neg \mathcal{F}_k \,\} \leq \varepsilon''_k$.

Proof. Assume $\mathcal{F}_{k+1}$. Then, according to the error-correction property, except for an event of probability $\le \varepsilon''_k$, for every t in $[t_1 - T_{\bullet k}/3, t_1]$ there is a t' in $[t_1 - T_{\bullet k+1}/3, t]$ with

$$\eta^k(x_k,t).Rider^k \succeq \varphi_{k**}(\eta^{k+1}(x_{k+1},t'))(y'_k).Rider^k.$$

Since $(Rider^i)$ is a guarded shared field and $\eta^{k+1}(x_{k+1},t).Guard^{k+1} = -1$ for all t in $t_1 + [-T_{\bullet k+1}/3, 0]$, we can replace φ_{k**} with φ_{k*} in the above equation. $\mathcal{F}_{k+1}$ implies

$$\eta^{k+1}(x_{k+1},t').Rider^{k+1} = \varrho^{k+1}(x'_{k+1}).$$

The identification property and the aggregation property (4.22) implies

$$\varphi_{k*}(\eta^{k+1}(x_{k+1},t'))(y'_k).Rider^k = \varrho^k(x'_k).$$

Thus, except for an event of probability $\le \varepsilon''_k$, we have $\mathcal{F}_k$.

Let

$$n = \bigwedge\{\, k : t_1 < T_{\bullet k}/3 \text{ or } k = K \,\},$$
$$t_2 = t_1 - T_{\bullet n}/3.$$

3. Consider the case $t_2 < 0$.
   By the construction and Proposition 4.7 the relation (9.22) holds for $k = n$, $t = 0$. Since the duration of $[0, t_1]$ is less than $T_{\bullet n}$ the probability that $\eta^n(x_n, t)$ undergoes any change during this interval is less than ε_n, proving $\mathrm{Prob}\{\, \neg\mathcal{F}_n \,\} \le \varepsilon_n$.
4. Consider the case $t_2 \ge 0$: then $n = K < \infty$ and the space is finite.
   According to the definition of ξ^K_K in (4.27), and of $\eta^1(\cdot,0) = \xi^K_1(\cdot,0)$ in the same proof, the value $\eta^K(o_K + yB_K \bmod N, 0)$ is defined for y in $[0, \lfloor N/B_K \rfloor - 1]$. Thus, at time 0 the space is filled as much as possible with adjacent cells of η^K one of which is x_K. The set $R = \mathbb{Z}_N \times [0, t_2]$ can be covered by at most

$$r = \lceil N/B_K \rceil \lceil 2t_2/T_{\bullet K} \rceil.$$

   copies of the rectangle V_K. Since η^K is a trajectory of the robust medium M_K, the probability of damage on each of these rectangles is at most ε_K. Therefore the probability that there is any damage in η^K over R is at most $r\varepsilon_K$. Assume that there is no damage in η^K over R. Then the cling-to-life condition and the computation condition imply that all cells of $\eta^K(\cdot,0)$ remain nonvacant until time t_2. We defined $Rd\_trans_K()$ to leave the rider and guard fields unchanged, and $\eta^K(x,0).Guard^K = 0$. Hence, by the definition of "combined" in Subsection 9.1, for each of these cells z we have

$$\eta^K(z,t_2).Rider^K = \eta^K(z,0).Rider^K$$

   implying (9.22) for $k = n = K$. Hence the total probability upper bound is

$$\sum_{k=1}^{n-1} \varepsilon''_k + 3t_2\varepsilon_K Q_K$$

   where we used $N < B_{K+1} = B_K Q_K$. With the parameters of Example 9.3, we have $\varepsilon_k Q_k \le \varepsilon^{2^{k-2}}$ for small enough ε, and $N < B_{K+1} = e^{2K\log K + O(K)}$, hence $\varepsilon_K Q_K$ can be written as $\varepsilon^{h_2(N)}$ with

$$h_2(N) = N^{c_2/\log\log N}$$

   for some constant c_2.

□

Proof of Theorem 6.15: We proceed in analogy with the proof of Theorem 6.14 using the same notation wherever appropriate and pointing out only what is new or different. We will find out by the end what choice for the functions $h_0(t)$ and $h_1(t)$ works. One can assume that Tr is commutative in the sense of Subsection 7.1 since the methods of that subsection can be used to translate the result for arbitrary transition functions. The transition function $Rd\_trans_k().Rider^k$ on the rider track will be defined to be the aggregated transition function $Tr^{B'_k}$, as in Example 3.4, and the $Rd\_trans_k().Guard^k$ will leave the guard field unchanged. With this, an amplifier frame will be obtained as in Example 9.3. The Amplifier Lemma gives a uniform amplifier with media M_k as in (9.20) with a hierarchical code having a guarded shared field ($Rider^k$), ($Guard^k$) as in (9.21). The parameters of that example are chosen in such a way that $B'_k = B_k$ and therefore

$$|Visible(K, N)| = N. \tag{9.23}$$

Let ζ be any trajectory of Tr over $\mathbb{Z}$ with $\zeta(\cdot, 0)$ of the form $Init(\varrho)$ (see (5.4)), satisfying $2|Supp(\zeta, t)| \le s$, where $\varrho \in (\mathbb{S}_1.Rider^1.Input)^{\mathbb{Z}}$. For any s and $t_1 \ge 0$, let a sufficiently large K be chosen: we will see by the end of the proof how large it must be. We certainly need

$$Supp(\zeta, t) \subset Visible(K, N) \tag{9.24}$$

which, in view of (9.23), can be satisfied if $N \ge s/2$. Let $\eta^1(\cdot, 0) = \Gamma(Init(\varrho); K)$. The trajectories η^1, η^k are defined as before. Let $x_1 = X(y)$ for a y in $Visible(K, N)$. Let $\zeta^k(x, t)$ be the trajectory of the aggregated cellular automaton $CA(Tr^{B'_k}, B'_k, B'_k)$ with the aggregated initial configuration $\iota_{B'_k *}(\zeta(\cdot, 0))$. Let

$$u = |\lfloor t_2 / T^\bullet{}_K \rfloor - 1|^+. \tag{9.25}$$

Let $\mathcal{F}_k$ be the event that

$$\eta^k(x_k, t).Rider^k.Output \succeq \zeta^k(x'_k, u).Output \tag{9.26}$$

and $\eta^k(x_k, t).Guard^k = -1$ holds for t in $[t_1 - T_{\bullet k}/3, t_1]$.

1. We have $\mathrm{Prob}\{\, \mathcal{F}_{k+1} \cap \neg \mathcal{F}_k \,\} \le \varepsilon''_k$.
   Proof. Assume $\mathcal{F}_{k+1}$. According to the error-correction property, except for an event of probability $\le \varepsilon''_k$, for every t in $[t_1 - T_{\bullet k}/3, t_1]$ there is a t' in $[t_1 - T_{\bullet k+1}/3, t]$ with
   $$\eta^k(x_k, t).Rider^k \succeq \varphi_{k**}(\eta^{k+1}(x_{k+1}, t'))(y'_k).Rider^k.$$
   Again, we can replace φ_{k**} with φ_{k*} here. $\mathcal{F}_{k+1}$ implies
   $$\eta^{k+1}(x_{k+1}, t').Rider^{k+1}.Output \succeq \zeta^{k+1}(x'_{k+1}, u).Output.$$
   Hence, the identification property and the aggregation property (4.22) implies
   $$\varphi_{k*}(\eta^{k+1}(x_{k+1}, t'))(y'_k).Rider^k.Output \succeq \zeta^k(x'_k, u).Output.$$
   Thus, except for an event of probability $\le \varepsilon''_k$, we have
   $$\eta^k(x_k, t).Rider^k.Output \succeq \zeta^k(x'_k, u).Output.$$
2. Consider the case $t_2 < 0$.
   Then the statement of the theorem holds since the $\zeta(\cdot, 0).Output$ is filled with $*$'s.
3. Consider the case $t_2 \ge 0$: then $n = K < \infty$.

According to the definition of $\eta^1(\cdot,0) = \xi_1^K(\cdot,0)$, at time 0, the space is filled as much as possible with adjacent cells of η^K one of which is x_K. Let us define the intervals

$$I = \bigcup\{\, C_k(y) + [0, B_K - 1] : C'_K(y) \text{ is in any support of } \varrho^K \,\},$$
$$J = I \cap x_1 + B_K(t_2/T_{\bullet K} + 1)[-1, 1].$$

The set $R = J \times [0, t_2]$ can be covered by at most

$$r = \lceil |J|/B_K \rceil \lceil 2t_2/T_{\bullet K} \rceil < 2(t_2/T_{\bullet K} + 2)\lceil 2t_2/T_{\bullet K} \rceil.$$

copies of the rectangle V_K. Since η^K is a trajectory of the robust medium M_K, the probability of damage on each of these rectangles is at most ε_K. Therefore the probability that there is any damage in η^K over R is at most $r\varepsilon_K$. Assume that there is no damage in η^K over R. Then the cling-to-life condition and the computation condition imply that all cells of $\eta^K(\cdot,0)$ remain nonvacant until time t_2.

We defined $Rd\_trans_k()$ to leave $Guard^k$ unchanged, and to be the aggregated transition function $Tr^{B'_k}$, on $Rider^k$. We have $\eta^K(x,0).Guard^K = 0$. Hence, by the definition of "combined" in Subsection 9.1, each cell in each of its transitions during the damage-free computation, applies $Tr^{B'_K}$ to the field $Rider^K$, making at least $\lfloor t_2/T^{\bullet}{}_K \rfloor - 1$ steps. By (9.24), I contains the image of a support of ϱ^K, therefore the confinement to this interval does not change the result. Thus

$$\eta^K(x,t_2).Rider^K.Output \succeq \zeta^K(x,u).Output,$$

and thus $\mathcal{F}_K$ holds. This gives a probability upper bound

$$\sum_{k=1}^{\infty} \varepsilon''_k + 2\varepsilon_K(t_2/T_{\bullet K} + 2)\lceil 2t_2/T_{\bullet K} \rceil.$$

Using (9.25) and (9.7) (and ignoring integer parts and the additive 1 and 2) this can be upper-bounded by

$$\sum_{k=1}^{\infty} \varepsilon''_k + 36\varepsilon_K u^2.$$

For small enough ε, by (9.11), the second term is $\le u^2\varepsilon^{2^{K-2}}$, and hence

$$K \ge h_0(u) = \log\log u + O(1)$$

will do. This gives $T^{\bullet}{}_K = e^{4K\log K + O(K)} \le (\log u)^{5\log\log\log u}$ for large enough u showing that

$$t_1 \ge h_1(u) = u(\log u)^{5\log\log\log u}$$

will do.

□

Remark 9.5 (Non-ergodicity without the error-correction property). The paper [10] mentions no explicit error-correction property in the proof of Theorem 2.5. Rather, it uses the fact that the fields *Age* and *Addr* have a kind of automatic error-correction property. In this way, we indeed arrive at several different invariant measures. However, one of these two measures may just be a shifted version of the other one, allowing the medium to be "forgetful" in the sense of Subsection 5.1. ♢

10. Self-organization

10.1. **Self-organizing amplifiers.** Consider a robust medium

$$M = Rob(Tr^{(w)}, B, T_\bullet, T^\bullet, \varepsilon, \varepsilon', r). \tag{10.1}$$

In what follows, it is convenient to speak of a distinguished value of the field *Color*: let it be called "blue". But the definitions and statements hold for every value of *Color*.

When a cell turns from a vacant to a non-vacant state, we can distinguish two ways in which this can happen. The first one is if the cell has a time with no non-vacant neighbor within $T_\bullet/2$ before the event: let us call this *spontaneous birth*; the second one is when it does not: let us call this *creation*. Under certain conditions, spontaneous births are to be prevented, since they may give rise to the wrong kind of cell. One way to achieve this is to populate an area tightly with cells.

For an interval I not consisting of a point, let

$$\Gamma(I, d) = \{\, x : [x - d, x + d] \cap I \neq \emptyset \,\} \tag{10.2}$$

We will also write $\Gamma(x, d) = [x - d, x + d-]$. In a space-time configuration η of M, a point (x, t) is said to be *controlled* if $\Gamma(x, B) \times [t - 6T^\bullet, t]$ contains a cell. It is *blue* if all cells in this area are blue. A space-time set is controlled or blue if all of its points are. A medium has the *lasting control* property if for all of its trajectories, for all sites x_0 and times $t_1 < t_2$, with

$$I = \Gamma(x_0, ((t_2 - t_1)T^\bullet/T_\bullet + 4)B),$$

if $I \times \{t_1\}$ is blue and $I \times [t_1 - 2T^\bullet, t_2]$ is damage-free then (x_0, t_2) is blue. Let M_1, M_2 be robust media, and Φ a simulation between them. We say that this simulation has *control delegation* property if the following holds for all sites x_0 and time t_1 with

$$I = \Gamma(x_0, (8T^\bullet{}_2/T_\bullet{}_2 + 4)QB).$$

Let η be a trajectory of M_1. If $I \times \{t_1 - 8T^\bullet{}_2\}$ is blue in $\Phi^*(\eta)$ and the area $I \times [t_1 - 16T^\bullet{}_2, t_1]$ is damage-free in η then (x_0, t_1) is blue in η.

For some parameters $D, \sigma > 0$, we will say that the trajectory (μ, η) of medium M is (D, σ)-*blue* at time t if the following holds. Let $I_1, \ldots, I_n$ be any system of intervals of size D such that $\Gamma(I_j, D)$ are disjoint. Then the probability that each of the sets $I_j \times [t - 3T^\bullet, t]$ is non-blue is at most σ^n.

Let us be given an amplifier as in (9.21), constants C_1, C_2, κ_1, and a sequence $\sigma_k > 0$ with

$$\begin{aligned} &\sigma_k U_k \to 0, \\ &\sigma_{k+1} \geq R_0(U_k \varepsilon_k + Q_k^2 \sigma_k^2 + e^{-\kappa_1 U_k/Q_k}). \end{aligned} \tag{10.3}$$

We will call this a *self-organizing amplifier* if the following properties hold:

(a) all media in it have the lasting control property;
(b) each of its simulations has the control delegation property;
(c) for every k, every time t, every trajectory (μ, η) of M_k that is $(C_1 B_k, \sigma_k)$-blue at time t, the trajectory $(\mu, \Phi_k^*(\eta))$ of M_{k+1} is $(C_1 B_{k+1}, \sigma_{k+1})$-blue at time $t + C_2 T^\bullet{}_{k+1}$;

Example 10.1. To the choices of Example 9.3, let us add the choice $\sigma_k = c^{-2} e^{-k}$. Let us check (10.3). Its right-hand side is

$$c^3 k^4 \varepsilon_k + c^{-2} k^4 e^{-2k} + e^{-\kappa_1 c k^2}$$

Given the bound $\varepsilon^{1.5^{k-1}}$ on ε_k for $k > 1$, (see (9.8)), this is less then $c^{-2} e^{-k}$ if c is large enough and ε is small enough. ♢

Lemma 10.2 (Self-Organization)**.** *Each amplifier frame Frame with large enough R_0 and small enough ε can be completed to a self-organizing amplifier with the property that if $Rd\_trans(\mathbf{r}).Guard = -1$ for all $\mathbf{r}$ then $Sim\_trans(\mathbf{r}).Guard = -1$ for all $\mathbf{r}$.*

10.2. **Application of self-organizing amplifiers.**

Proof of Theorem 6.17. 1. This proof starts analogously to the proof of Theorem 6.14.

Let us define an amplifier frame *Frame* as in Example 9.3, with the rider transition function having the property

$$Rd\_trans_k(\mathbf{r}).Rider = r_0.Rider,$$
$$Rd\_trans_k(\mathbf{r}).Guard = -1.$$

i.e. leaving the *Rider* field unchanged and always setting the guard field to -1. It will have a broadcast field *Color* with m bits. Applying Lemma 10.2, we obtain a self-organizing amplifier with media M_k as in (9.20) with a hierarchical code having a guarded shared field $Rider^k$, $Guard^k$. Also, we will have $Sim\_trans(\mathbf{r}).Guard = -1$ for all $\mathbf{r}$. Let

$$K = \sup\{\, k : 5B_k \le N \,\}.$$

For a certain value of the field *Color* that we call *Blue*, we create an initial configuration $\eta^1(\cdot, 0)$ consisting of all blue latent cells covering the whole space, with $Guard = -1$. Let the trajectories η^1, η^k be defined as before. Let (x_1, t_1) be a space-time point. Eventually, we want to estimate the probability of $Blue = \eta^1(x_1, t_1).Color^1$.

Let

$$\begin{aligned} I_1 &= \{x_0\}, \\ v_1 &= t_1, \\ v_{k+1} &= v_k - 8T^\bullet{}_{k+1}, \\ I_{k+1} &= \Gamma(I_k, (8T^\bullet{}_{k+1}/T_{\bullet k+1} + 4)B_{k+1}), \\ u_1 &= 0, \\ u_{k+1} &= u_k + C_2 T^\bullet{}_{k+1}. \end{aligned}$$

If $k \le K$ then by the self-organizing property of the amplifier, the evolution η^k is $(C_1 B_k, \sigma_k)$-blue at time u_k for each k. Let $\mathcal{F}_k$ be the event that $I_k \times \{v_k\}$ is blue in η^k.

2. There is a constant κ_1 such that

$$\mathrm{Prob}\{\, \mathcal{F}_{k+1} \cap \neg \mathcal{F}_k \,\} \le \kappa_1 U_k Q_k \varepsilon_k.$$

Proof. The right-hand side is an upper bound on the probability of damage occurring in η^k in $I_{k+1} \times [v_{k+1}, v_k]$. If damage does not occur there and $\mathcal{F}_{k+1}$ holds then the control delegation property implies $\mathcal{F}_k$.

Let

$$\begin{aligned} n &= \bigwedge \{\, k : v_{k+1} < u_{k+1} \text{ or } k = K \,\}, \\ t_2 &= u_n. \end{aligned}$$

3. Consider the case $n < K$.

Then we have $u_n < v_n < u_{n+1} + (C_2 + 8)T^\bullet{}_{k+1}$. Let

$$J = \Gamma(I_n, ((C_2 + 8)T^\bullet{}_{n+1}/T_{\bullet n} + 3)B_n).$$

Let $\mathcal{B}$ be the event that $J \times \{u_n\}$ is blue in η^n. Since η^n is $(C_1 B_n, \sigma_n)$-blue at time u_n for each n, the probability of $\neg\mathcal{B}$ is at most $\kappa_2 U_n \sigma_n$ for some constant κ_2.

3.1. There is a constant κ_3 such that

$$\mathrm{Prob}\{\, \mathcal{B} \cap \neg \mathcal{F}_n \,\} \leq \kappa_3 U_n Q_n \varepsilon_n.$$

Proof. The right-hand side is an upper bound on the probability of damage occurring in η^k in $J_n \times [u_n, v_n]$. If damage does not occur there and $\mathcal{B}$ holds then the lasting control property implies $\mathcal{F}_n$.

Thus, the probability that (x_1, t_1) is not blue can be upper-bounded by

$$\kappa_2 U_n \sigma_n + \kappa_2 U_n Q_n \varepsilon_n + \kappa_1 \sum_{i=1}^{n-1} U_k Q_k \varepsilon_k.$$

4. Consider the case $n = K$: then the space is finite.
Let $\mathcal{B}$ be the event that the whole space $\mathbb{Z}_N$ is blue in η^n. Since η^n is $(C_1 B_n, \sigma_n)$-blue at time u_n for each n, and $5B_{n+1} \leq N$, the probability of $\neg\mathcal{B}$ is at most $\kappa_2 Q_n \sigma_n$ for some constant κ_2.

4.1. There is a constant κ_3 such that

$$\mathrm{Prob}\{\, \mathcal{B} \cap \neg \mathcal{F}_n \,\} \leq \kappa_3 U_n Q_n t_1 / T_{\bullet n} \varepsilon_n.$$

Proof. The right-hand side is an upper bound on the probability of damage occurring in η^k in $J_n \times [u_n, v_n]$. If damage does not occur there and $\mathcal{B}$ holds then by the lasting control property, $\mathcal{F}_n$ also holds.

Thus, the probability that (x_1, t_1) is not blue can be upper-bounded by

$$\kappa_2 U_n \sigma_n + \kappa_1 \sum_{i=1}^{n-1} U_k Q_k \varepsilon_k + \kappa_3 U_n Q_n t_1 / T_{\bullet n} \varepsilon_n.$$

Thus, the probability that (x_1, t_1) is not blue can be upper-bounded in both cases 3 and 4 (ignoring multiplicative constants) by

$$U_n \sigma_n + \sum_{i=1}^{n-1} U_k Q_k \varepsilon_k + U_n Q_n t_1 / T_{\bullet n} \varepsilon_n = A_1 + A_2 + A_3.$$

With the parameters of Example 10.1 we have $A_2 = O(\varepsilon)$ and A_3 can be written, as in the proof of Theorem 6.14 as $\varepsilon^{h_2(N)}$ with

$$h_2(N) = N^{c_2 / \log\log N}.$$

for some constant c_2. Also, $A_1 < e^{-\kappa_4 n}$ for some $\kappa_4 > 0$, If $n = K$ then this gives $A_1 < e^{-\kappa_4 K}$. Since $N = e^{2K \log K + O(K)}$ thus

$$A_1 = O(N^{-0.4}).$$

If $n < K$ then $t_1 = O(\sum_{k=1}^{n} T^{\bullet}{}_k)$. Hence $t_1 = e^{2n \log n + O(1)}$, giving

$$A_1 = O(t_1^{-0.4}).$$

Both of these bounds are poor if t_1 is small. But in this case there is a trivial upper bound $O(t_1^2 \varepsilon)$ on the total probability that (x_1, t_1) is not blue: this bounds the probability that there has been any damage since the beginning that could influence (x_1, t_1). □

11. General plan of the program

Our eventual goal is to prove Lemma 9.4 (Amplifier). The medium M_{k+1} to be simulated has reach 1. The simulation is the composition of two simulations. First, we simulate M_{k+1} by a medium M'_k having reach 2, and then we simulate M'_k by a medium M_k having the same cells but with reach 1. The latter simulation uses the construction in Theorem 8.10. The bulk of the work is in the first simulation, to which now we proceed. From now on, we fix the level k of the simulation hierarchy, refer to medium M'_k as M and to medium M_{k+1} as M^*. The subscript will be deleted from all parameters in M and a superscript $*$ will be added to all parameters of M^*. We will refer to cells of M as *small cells*, or simply *cells*, and to cells of M^* as *big cells*.

According to the definition of an amplifier frame, the medium M has two special fields called *Rider* and *Guard*, and can also have other fields. These two fields will be subfields of the field *Info* in our medium. In what follows we define the function *Sim_trans* only: the function *Rd_trans* is given in advance. The updating of *Rider* and *Guard* is determined by the rule (9.1). Thus, *Sim_trans* determines the next value of *Rider* only in case of $Guard \neq 0$, and the next value of *Guard* in case of $Guard < 0$. We will not point out always that what we define is only *Sim_trans* but this is to be understood. In particular, when our program requires to update *Info* then in the subfield *Rider* this will be actually carried out only if $Guard \neq 0$.

For simplicity, the proof will only be given for $q_k = Q_k$ (see 4.1.4). The general case is not more difficult, but would need extra notation. The track *Info* of a colony contains the string that is the state of the represented cell, encoded via an error-correcting code. *Info* consists of two subfields, *Info.Main* and *Info.Redun*. The track *Info.Main* contains the intended original information, including *Rider* and *Guard*. The track *Info.Redun* contains "parity checks" for the string on track *Info.Main*, but contains only 0 in cell 1 (due to the "controlling" property). Since *Rider* must control cell 1 of the colony in the sense of 4.1.3, the track *Redun* does not use cell 1. For the error-correcting code, the field *Info.Main* will be subdivided into "packets" such that the parity check bits for each packet will be computed separately. The fields $Rider^*$ and $Guard^*$ will occupy packets disjoint from the other fields and therefore the error-correction for the other fields will proceed without difficulty even if e.g. the error-correction for $Rider^*$ is prevented by $Guard^* = 0$.

The program will be described in a semi-formal way; the present section overviews it. Later sections restrict the program more and more by giving some rules and conditions, and prove lemmas along the way. The typical condition would say that certain fields can only be changed by certain rules. The language for describing the rules is an extension of the one given in Subsection 7.4. We will introduce a fair number of fields but they are all relatively small.

All fields but *Info.Main* are contained in *Buf* (see 2.2.1) and hence are visible by neighbor cells and changeable immediately even by the separable transition function. Cells have an address field which determines the only *colony* (Q-colony) to which the cell belongs. A colony $\mathcal{C}(y)$ has base y. All properties and relations defined for colonies are automatically defined for the sites of potential big cells at their bases and vice versa. The *Age* field of a cell, called its *age*, can have values in $[0, U-1]$, but the upper bound will typically not be reached.

11.1. **Damage rectangles.** The damage map of the simulation $\varphi = \varphi_k$ was defined in Subsection 8.1. Let us write

$$Damage = Damage(\eta), \qquad Damage^* = Damage(\eta^*).$$

Lemma 8.4 proves the Restoration Property for $\eta^* = \varphi^*(\eta)$ whenever η is a trajectory of M. In the Computation Property applied to a cell x of η^*, we are given a rational number a satisfying $f_1(x, a, \eta) = 1$, i.e. throughout the rest of the construction, *it is assumed that* $Damage^*$ *does not*

intersect the set

$$W_1^*(x,a) = [x - 2QB, x + 3QB-] \times [a - {T_\bullet}^*/2+, a + 2{T^\bullet}^*]$$

(where we used the fact that the reach of η^* is 1). Recall some definitions from Subsection 8.1. We had $V = [-B/4, B/4-] \times [-T_\bullet/4+, 0]$. We have $\eta^*(x,t) \in \mathit{Bad}$ iff $\mathit{Damage}(\eta)$ contains at least two points u, v such that $u + 2V$, $v + 2V$ are disjoint and even $u + 3V, v + 3V$ are contained in $(x,t) + 4V^* + (B^*/2, 0)$ (see Figure 14). Let

$$V' = \{\, u : u + 3V \in V^* + (QB/2, 0) \,\}. \tag{11.1}$$

The later definitions (13.4), (13.10) imply $Q > 10$, ${T_\bullet}^* > 6T_\bullet$. This guarantees that the space projection of V' is at least $1.5QB$ and the time projection is at least ${T_\bullet}^*/2$. We have $\eta^*(x,t) \notin \mathit{Bad}$ iff there is a rectangle of the form $(y,u) + 2V$ covering $\mathit{Damage} \cap (x,t) + V'$. It is easy to see that if Damage^* does not intersect W_1^* then there is a finite set of small rectangles of this latter form covering $\mathit{Damage} \cap W_1^*$ such that each rectangle of the form $(z,w) + V'$ intersects at most one of them. We will call these small rectangles the *damage rectangles*.

Let us say that the damage *affects* (x, t_0) *directly* unless one of the following occurs:

(1) $\{x\} \times [t_0 - 2T^\bullet - T_\bullet/2+, t_0]$ is disjoint from the damage rectangle;
(2) There is a dwell period $[t_1, t_2]$ of x with $t_0 - 2T^\bullet < t_2 \le t_0$ such that $\{x\} \times [t_1 - T_\bullet/2+, t_0]$ is disjoint from the damage rectangle;
(3) There is a switching time t_2 of x with $t_0 - 2T^\bullet < t_2 \le t_0$ such that $\{x\} \times [t_2 - T^\bullet - T_\bullet/2+, t_0]$ is disjoint from the damage rectangle;

This definition encompasses the cases when damage prevents the application of parts (a), (b), (c) or (d) of the Computation Property for concluding about (x, t_0).

We will say that (x, t_0) is affected *via neighbors* if in the above definition, one of the conditions does not hold with the interval $[x - 2B, x + 3B-]$ in place of $\{x\}$. This encompasses the cases when damage prevents the application of parts (e), (f), (g) or (h) (h) of the Computation Property. Thus the damage can affect more cells via neighbors but only during the same time interval.

By the definition of V, the (half-open) damage rectangle has width B and duration $T_\bullet/2$; therefore it can affect at most one cell directly, for less than $2T^\bullet + T_\bullet/2$ time units. For each damage rectangle we define a corresponding *extended* damage rectangle, an upper bound on the set of points affected directly. Generally, such a rectangle will be denoted by the Cartesian product

$$[a_0, a_1-] \times [u_0+, u_1-] \tag{11.2}$$

with $a_1 - a_0 = B$, $u_1 - u_0 = 2T^\bullet + T_\bullet$.

Lemma 11.1. *Suppose that (x,u) is not affected directly by damage but is affected via neighbors. Then x is not affected directly during $[u - 5T^\bullet, u]$ and not affected even via neighbors during $[u - 5T^\bullet, u - 3T^\bullet]$. If (x,u) is the end of a dwell period then x is not affected via neighbors during $[u - 5T^\bullet, u - 2T^\bullet]$.*

Proof. Some damage occurs near x but not in x at some time t_1 during $[u - 2T^\bullet - T_\bullet/2+, u]$. When (x,u) is the end of a dwell period then, since x is affected via neighbors, $t_1 \ge u - T^\bullet - T_\bullet/2$ Then it does not affect x directly at all. Clearly, no other damage rectangle affects directly x during $[u - 5T^\bullet, u]$, and this damage rectangle does not affect x before $t_1 - T_\bullet/2$. □

11.2. **Timing.** Let

$$\lambda = T^\bullet/T_\bullet. \tag{11.3}$$

Different actions of the program will be carried out with different speeds. We introduce some delay constants:

$$4\lambda < p_0 < p_1 < p_2 \tag{11.4}$$

whose value will be defined later. The slowdown uses a subdivision of the state into the following fields:

$$\mathit{Wait}, \mathit{Cur}, \mathit{Fut}.$$

Here, *Wait* takes values $1, \dots, p_2$, and $\mathit{Cur}, \mathit{Fut}$ have equal size. We will work almost exclusively with the field *Cur*, which represents the "current state". Its subfields $\mathit{Cur}.F$, will simply be written as F. An assignment will typically have the form

$$F :=_p v.$$

This means that $F.\mathit{Fut}$ gets value v and *Wait* gets value p. The default change of *Wait* is to decrease it by 1 if it is different from 1. When $\mathit{Wait} = 1$ then *Fut* will be copied into *Cur*. Missing subscript in the assignment means $p = 1$. Thus, all examined fields are understood to be the corresponding subfields of *Cur* and all changed fields are understood to be the corresponding subfields of *Fut*.

Condition 11.2 (Waiting).

(a) A field $\mathit{Cur}.F$ can only be changed when $\mathit{Wait}.F = 1$, in which case it is set to $\mathit{Fut}.F$.
(b) A field $\mathit{Wait}.F$ can only either decrease by 1 until it reaches 0 or stay 0, or be set to one of the values $1, \dots, p_2$.
(c) If $\mathit{Fut}.F$ changes then $\mathit{Wait}.F$ will be set to one of the values $1, \dots, p_2$.

♢

E.g., the proper reading of rule

```
cond {
   ?  Addr = 0 and Addr^1 = 0
   !  Kind :=_p Latent
   }
```

is that if $\mathit{Cur}.\mathit{Addr}(\mathbf{x}) = 0$ and $\mathit{Cur}.\mathit{Addr}(\mathbf{x} + B) = 0$ then $\mathit{Fut}.\mathit{Kind}(\mathbf{x})$ must be set to *Latent* and $\mathit{Wait}.\mathit{Kind}(\mathbf{x})$ must be set to p.

We will have $\mathit{Cur} \cap \mathit{Inbuf} = \emptyset$ where *Inbuf* was defined in 2.2.1. Thus, *Cur* will never be changed in one step in response to a change in a neighbor cell.

11.3. **Cell kinds.** The values of the address field *Addr* will vary in $[-Q, 2Q - 1]$. The cells whose colony is their originating colony, will be called *inner cells*, the other ones will be called *outer cells*. Cells will be of a few different kinds, distinguished by the field

$$\mathit{Kind}$$

with possible values (for $j \in \{-1, 1\}$) *Vac*, *Latent*, $\mathit{Channel}_j$, Growth_j, *Member*, *Germ*. The kind of a non-germ, non-latent cell is determined by *Age* and *Addr*. The kinds of cells are ordered by *strength*, as follows:

$$\mathit{Vac} < \mathit{Latent} < \mathit{Germ} < \mathit{Channel}_j < \mathit{Growth}_j < \mathit{Member}.$$

Stronger cells prevail over weaker ones in conflicts about who should create a neighbor. The relation $\mathit{Kind}(x, t) = \mathit{Vac}$ means that there is no cell at site x at time t.

Condition 11.3 (Latent Cells).

(a) A vacant cell can only turn into a latent one.
(b) A latent cell has $\mathit{Guard} = -1$ (else it is bad).

♢

A cell will be called *dead* if it is vacant or latent, and *live* otherwise. *Killing* a cell x means turning it latent: only when place must be vacated for a new nonadjacent cell whose body intersects the body of x will it be turned vacant. This will imply Condition 8.7 (Cling-to-Life). Members have addresses in $[0, Q-1]$. They are the strongest kind in order to maintain the integrity of colonies.

A right outer cell has (by definition) addresses $> Q$, and a left outer cell has addresses < 0. Germ cells have addresses in $[-Q, 2Q-1]$.

11.4. **Refreshing.** Since several damage rectangles can occur during a work period, a rule called *Refresh* will be performed several times during the work period and will attempt to correct information by global decoding-encoding. The last step of each invocation of *Refresh* will be called a *refreshing step*. The number of steps between invokations of *Refresh* will be

$$(\text{refresh\_time}) = T_\bullet^*/T^\bullet \tag{11.5}$$

making sure that at most one damage rectangle may occur between refreshings.

Certain fields, called *locally maintained fields*, will be kept constant over the colony, for most of the work period. Here are some of these. The field *Doomed* is 1 if the represented big cell must be removed (the site to become vacant), and 0 otherwise. Cells with *Doomed* $= 1$ are called *doomed*. Cells will have fields

$$\mathit{Creating}_j, \mathit{Growing}_j \ (j \in \{-1, 1\})$$

with values 0 or 1. $\mathit{Creating}_j = 1$ will mean that a cell is a creator in the sense of Condition 8.6 (Creation). $\mathit{Creating}_j$ of a big cell will be broadcast into $\mathit{Growing}_j$ of its small cells. $\mathit{Growing}_j = 1$ signifies the collective decision of the colony to grow a new neighbor colony in direction j. The track

$$\mathit{Control}_{0.1j}$$

controls the retrieval of information from neighbor colonies. It will be updated periodically by the subrule *Retrieve*.

The globally maintained field

$$\mathit{End}$$

will be locally updated by the rule

$$\mathit{Find\_end}.$$

The work period will end when $\mathit{Age} = \mathit{End}$. This will help setting the absolute duration of the work period more predictably. The default value of *End* is $U-1$ and we will always have $\mathit{End} \geq \mathit{Age}$.

Each locally maintained field F has some *update ages*: ages when F will be recomputed. An interval of size $4Q$ around each of these ages will be called an *update interval* for F.

11.5. **A colony work period.** The main stages are as follows:

Idle;
Extend;
Idle;
Retrieve;
Compute;
Idle;
Find_end;
Grow;
Shrink;
Finish;

The numbers

$$(\text{compute\_start}) < (\text{idle\_start}) < (\text{grow\_start}) < (\text{grow\_end}) < \mathit{End}$$

define the ages starting or ending important stages of this program and will be defined later. The idle stages make sure that

- the computation and communication times are positioned correctly;
- faults in one part have limited effect on other parts.

The numbers up to (idle_start) (i.e. (compute_start) and (idle_start)) are constant, and the numbers $\mathit{End}-(\text{grow\_start})$ and $\mathit{End}-(\text{grow\_end})$ are also constant. The difference $(\text{grow\_start})-(\text{idle\_start})$ is not constant since *End* will be a computed value. Here is a short description of the major stages.

Extension: The subrule *Extend* tries to extend some arms of the colony left and right, to use in communicating with a possible non-adjacent neighbor colony. In direction j, if it is not adjacent to another colony it will extend an arm of cells of kind $\mathit{Channel}_j$. In channel cells in the positive direction, the *Addr* field continues its values through

$$Q, Q+1, \ldots, 2Q-1.$$

Similarly in channel cells in the negative direction. Channel cells are weaker than member cells, so the growth of a channel does not normally damage another colony. The channels will be killed at the end of the computation.

Retrieval: Retrieval starts at age (compute_start). During this, the colony tries to retrieve the state represented by it and its neighbor colonies. The neighbor colonies will cooperate in this, having a dedicated mail track and control track for serving each neighbor. Atomicity will be guaranteed by waiting, similarly to the proof of Theorem 8.10 (Reach Extension).

Computation : The subrule *Compute* computes the output of the simulated transition function and stores it on the track *Hold*. It will put $\mathit{Doomed} = 1$ into each cell of the colony if the represented cell is to be erased. We will always set $\mathit{Doomed} = 0$ at $\mathit{Age} = (\text{compute\_start})$.

Idling: After the computation, some idling puts sufficient time between it and the end of the work period to make sure that the retrieval falls into the observation part of the work period. Procedure *Find_end* computes *End*.

Growth: If $\mathit{Growing}_j = 1$ then, between values (grow_start) and (grow_end), the colony tries to extend an arm of length at most Q in direction j. These cells are of kind Growth_j, or *growth cells*.

Birth: A latent cell x turns immediately into a germ cell with address $-Q$ and age 0. The germ thus started begins to grow to the right, trying to fill 3 colonies until age (germ_grow_end). At

$$\mathit{Age} = (\text{germ\_end}) - 1,$$

germ cells turn into member cells.

Germs will implement the self-organizing property of the medium.

Shrinking: When they reach the end of their growth period, growth and germ cells stop producing offshoot. In what follows, all edges whose existence is not justified by these processes (these edges will be called "exposed") will be subject to the process *Decay*. Therefore normally, a growth either disappears before the end of the work period or it covers a whole new colony by that time. Similarly, germ cells are required to cover 3 neighbor colonies.

Finish: Rule *Finish* will be called when $\mathit{Age} = \mathit{End}$. It reduces the addresses of growth cells mod Q. Outer cells and inner germ cells turn into members. If the colony is doomed it will be erased, (starting from the right end, since a doomed right endcell will be considered inconsistent with its neighbors). Otherwise, the information from the *Hold* track will be copied

into the corresponding locations on the *Info* track. This final rule will take only a single step, and will also be called the *cut*.

11.6. **Local consistency.** The basic structural soundness of a colony will be expressed by some local consistency conditions. Two cells locally consistent with each other will be called *siblings*. For $j \in \{-1, 1\}$, let

$$Sib(j) = 1$$

if cell $\mathbf{x}$ is a sibling of its adjacent neighbor in direction j, and $Sib(j) = 0$ otherwise. Sometimes, rules refer to the relation $Sib()$ but can be understood without going into the details of the definition of this relation itself.

11.6.1. *Siblings.* Two aligned cells x and $x + iB$ will be called *space-consistent* if $Addr(x + iB) - Addr(x) \equiv i \pmod{Q}$.

Let us consider cells x and $y = x + jB$ for $j = -1, 1$, with $|Addr(x)| < |Addr(y)|$. These two cells belong to the *same work period* if

$$0 \leq Age(x) - Age(y) \leq 1.$$

They *straddle a work period boundary* if $Age(x) = 0$, $Age(y) = End$. They are *siblings* if one of the following two properties holds.

(1) x, y belong to the same work period, originate at the same colony, either are both germ cells or neither of them is, and x is not a doomed right endcell with $0 < Age <$ (compute_start).
(2) x, y belong to the same colony, and straddle a work period boundary.

Of course, siblings are space-consistent. An interval of cells in which the neighboring cells are siblings will be called a *domain*. A domain of size n will also be called a *n-support* of its members. Let us call two cells *relatives* if they can be embedded into a common domain but are not necessarily in one. A colony with starting cell x will be called *full* if it is covered by a domain in such a way that $Addr(x + iB) \equiv i \pmod{Q}$.

11.6.2. *Age updating.* The rule for updating age is similar to the "marching soldiers" rule for updating *Age* in Subsection 7.1. As seen from the definition of siblings above, we impose some extra order on age: the age of all cells in the same extended colony must be non-increasing as they become more distant from the cell with $Addr = 0$. Also, there will be a bit

$$Frozen$$

with the property that when $Frozen = 1$ then *Age* will not be changed. Here is the basic updating rule for age:

```
rule March {
   cond {
      ?  Frozen = 0 and Age < End and ∀j ∈ {−1, 1}  ϑ_j(x)  is dead or the
         the increase of Age does not break the sibling relation with ϑ_j(x)
      !  Age :=_{p_1} Age + 1
      }}
```

Condition 11.4 (Address and Age).

(a) Only *Finish* can change *Addr* of a live cell;
(b) Only *March* and *Finish* change *Age* of a live cell;

♢

11.6.3. *Repairs.* The rule *Purge* eliminates isolated cells. The rule *Heal* repairs a small hole. An unrepaired hole will be enlarged by the rule *Decay*: this will eventually eliminate partial colonies.

The damage rectangle can destroy the information represented on its space projection, therefore the information representation on the *Info* track will be a redundant, error-correcting code. Information will be decoded before computation and encoded after it. The damage can also disrupt the computation itself therefore the decoding-computation-encoding sequence will be repeated several times. The result will be temporarily stored on the track *Hold*.

11.6.4. *New colonies.* The following condition helps enforce that newly created cells in the simulation are latent.

Condition 11.5 (Outer Info). Suppose that $\varphi^*(\eta)(x,t) \notin \mathit{Bad}$, and the colony with base x at time t is full and is covered with member cells belonging to the same work period, and is not intersected by the extended damage rectangle. Then $\varphi^*(\eta)(x,t)$ depends only on the *Info* track of this colony via some decoding function α^*. If the colony is covered with germ or outer cells then the state that would be decoded from this track is latent. ♢

11.7. **Plan of the rest of the proof.** In order to preserve intelligibility and modularity, rules and conditions belonging to the program will only be introduced as they are needed to prove some property.

The crucial Lemma 14.7 (Attribution) says that soon after the disappearence of the big damage Damage^*, all live non-germ cells not immediately arising from *Damage* can be attributed (via a path of ancestors) to some nearby colonies (all disjoint from each other). Thus, this lemma enables us to reason about the process over this area in terms of big cells. The lemma is important e.g. in seeing that a big cell can grow a neighbor if no other cell is nearby. In terms of colonies, this means that if no other colony is nearby, then a colony can grow and create a neighbor colony. This is not obvious since there could be "debris": earlier damage could have left bits and pieces of larger colonies which are hard to override locally. The Attribution Lemma will guarantee that those bits and pieces are not there anymore at the time when they could be an obstacle, since whatever is there is attributable to a big cell.

The Attribution Lemma also plays a role in healing. Most local healing is performed by the rule *Heal*. However, if the damage occurs at the end of some colony $\mathcal{C}$ then it is possible in principle that foreign material introduced by damage is connected to something large outside. The Attribution Lemma will imply that the foreign matter is weaker and can therefore be swept away by the regrowth of the member cells of $\mathcal{C}$.

The idea of the proof of the Attribution Lemma is the following. Suppose that (x_0, t_0) is a cell whose origin we want to trace. We will be able to follow a steep path (x_i, t_i) of "ancestors" backward in time until time $t_n = t_0 - mQ$ with some large coefficient m. Lemma 13.3 (Ancestor) shows that it is possible to lead a path around a damage rectangle. The attribution consists of showing that (x_n, t_n) belongs to a domain covering a whole colony. To prove this, we will show that the rule *Decay*, which eliminates partial colonies, would eventually cut through the steep path unless the latter ends in such a domain. In actual order, the proof proceeds as follows:

- Some of the simpler killing and creating rules and conditions will be introduced, and some lemmas will be proved that support the reasoning about paths and domains.
- We prove the Ancestor Lemma. Lemma 13.5 (Running Gap) says that if a gap is large enough then the process *Decay* propagates it fast, even in the presence of some damage.
- Lemma 14.2 (Bad Gap Inference) shows that (under certain conditions and in the absence of damage), if there is a gap at all then it is large enough in the above sense.
- The above lemmas are used to prove the Attribution Lemma.

Here is a summary of the roles of different delays:

p_0**:** Default;
p_1**:** Decay and computation;
p_2**:** Growth;

Here is a summary of the rest of the proof.

- We define those computation rules not dependent on communication with neighbor colonies.
- Lemma 16.5 (Legality) shows that the computation terminates gracefully independently of the success of communication.
- The development of colony $\mathcal{C}$ will be followed forward to the present in Lemma 16.7 (Present Attribution) .
- Finally, the retrieval rules will be defined and the remaining part of the Computation Property will be proved.

12. Killing and creation

The present section collects some rules, conditions and properties that will be needed in the rest of the construction. The material is somewhat heterogenous, but is related mostly to consistency, killing and creation.

12.1. **Edges.** Let

$$e_{-1} = 0,\ e_1 = Q - 1.$$

A cell x is a *colony endcell in direction* j if $Addr(x) \equiv e_j \pmod{Q}$. Outer or germ cells before the end of their growth ((grow_end) resp. (germ_grow_end)) are said to be in their *expansion period*, and are called *expansion cells*.

Suppose that cell x has no siblings in direction j. It will be called a *protected edge* in that direction if it is some legitimate boundary in that direction, in the sense described below; otherwise, it will be called an *exposed edge*. For each kind of cell listed below, the cell is defined to be protected if the statement listed after the colon is true; othewise, it is exposed.

Member: Colony endcell in direction j, except if it is a doomed right endcell with $0 < Age <$ (compute_start);

Expansion: In direction j if this is the direction of expansion, except when it is a channel cell with $Age \geq$ (idle_start);

Non-expansion, germ: Outer colony endcell in direction j if this is the extension direction and the cell is not outer with $Age >$ (germ_end) $- 2Q$;

Non-expansion, outer, non-germ: Any colony endcell in direction j if this is the extension direction;

In a rule, the condition that $\mathbf{x}$ is an exposed edge in direction j will be expressed by

$$Xposed_j = Xposed_j(\mathbf{x}) = 1.$$

An exposed edge is the sign of defect, or a call to eliminate an extension of a colony or a colony; the decay rule will kill an edge cell if it stays exposed too long.

Lemma 12.1. *If a left exposed edge dies and its right sibling was not a colony endcell then this neighbor becomes a left exposed edge. The same holds if we replace left with right.*

Proof. This is obvious in most cases. One case when it is not is when the exposed edge is a left outer cell that is not an expansion cell. It is imaginable namely that its right neighbor is still in the growth stage. However, our definition of siblings requires the ages of cells to be monotonically nonincreasing as we move away from the originating colony, therefore this is not possible.

The situation is similar when a doomed exposed right endcell dies. □

A *multidomain* is either a domain or the union of some adjacent space-consistent domains meeting in protected colony-endcells. From the above definitions it is clear that only a cut can turn a domain into a multidomain.

12.2. **Killing.** Generally, a cell will be "killed" by making it latent. Killing will be announced first by making a one-bit field

Dying

equal to 1. (The default value of this field is 0.) This feature will disable the cell from creating a live neighbor (just in the interest of continuity and ease of reasoning). So, whenever the program will kill a cell it will call the following subrule: its argument determines its speed.

```
subrule Die(p) {
    Dying :=_p 1;
    Kind :=_p Latent
    }
```

The killing of a cell is almost always a corrective action, except when the definition of "protected" makes certain cells exposed with the intention of killing off a whole channel or colony.

Condition 12.2 (Dooming). The only rules that change the field *Doomed* without killing the cell are the following.

(1) The subrules *Animate* and *Heal* when they create a non-doomed cell;
(2) At $Age = (\text{compute\_start})$, we set $Doomed = 0$;
(3) At $Age = (\text{idle\_start})$ we possibly set $Doomed = 1$.

♢

We call two cells *partners* if one of the following cases holds:

(1) They are relatives at distance $2B$ and the cell between them is not their sibling;
(2) They are adjacent and changing the age of one of them by 1 makes them siblings;

When a cell has a partner then it may happen that one of its neighbors will be corrected in a short while to become a sibling, therefore the cell will be frozen by the rule *Freeze* below. A frozen cell's age will not be advanced (seen the rule *March* in 11.6.2), making the neighbor's correction easier.

```
rule Freeze {
   cond {
      ?   x is exposed or has a non-dying partner
      !   Frozen := 1
      ?!   Frozen := 0
      }}
```

Condition 12.3 (Freeze). Only the rule *Freeze* can change the field *Frozen*. ♢

12.3. **Creation, birth, and arbitration.**

```
rule Create {
   pfor j ∈ {−1, 1} do {
      cond {
         ?   Kind = Vac and Creating_j^{−j} = 1
         !   Kind := Latent
         }}}
```

This rule is slightly different from the other ones since the site to which it is applied is vacant. Therefore in simulation, the cell $\mathbf{x}$ is not really there to "apply" this rule; it is applied implicitly by its creator neighbor $\vartheta_{-j}(\mathbf{x})$. We will call this neighbor the *mother* of the new latent cell.

```
rule Birth {
   cond {
      ?   Kind = Vac and Kind_j = Vac  (j = −1, 1)
      !   Kind := Latent
      }}
```

This rule tries to give rise to a newborn cell if its neighbors are vacant. As the Computation Property shows the birth rule will not be enforced by the trajectory property. (Still, birth will be realized for big cells when germ cells succeed in creating a new colony.)

Condition 12.4. *Create* and *Birth* are the only rules applicable to a vacant cell. ♢

Condition 12.4 implies that in all cases different from the one listed in the rules *Create* or *Birth*, the site is required by the transition function to remain vacant. Case (h) of Condition 8.8 (Computation Property) allows for the creation to be blocked by a cell whose body intersects the cell to be created.

Let us proceed to the definition of rule *Arbitrate* which controls the values of the field $Creating_j$. Consider the example of a cell x and its left nonadjacent neighbor $y = \vartheta_{-1.5}(x)$ that may want to create a cell in $y + B$, overlapping the body of x. Whether x will be erased will be decided not by whether y is stronger than x but by whether the new cell $y + B$ would be stronger than x.

Remark 12.5. This distinction matters when a colony wants to create an outer cell that would intrude into another colony. It is important in this case for the created cell to be weaker than the member cells of the other colony with whom it is competing for space. ♢

To simplify the expression of the rule, for a relation R, let

$$a \overset{R}{<} b$$

mean "$a < b$ or ($a = b$ and R holds)". The kind of the cell to be created must have been declared in field $Kind_{-j}$ of $\vartheta_j(\mathbf{x})$. This field is actually a function of two fields called

$$Kind_j.Grow,\ Kind_j.Heal,$$

which will be set by the rules *Grow_step.active* and *Heal* (see later), and is defined as

$$Kind_j = \begin{cases} Kind_j.Heal & \text{if } Dying = 0,\ Kind_j.Heal \neq Latent, \\ Kind_j.Grow & \text{if } Dying = 0,\ Kind_j.Grow \neq Latent, \\ Latent & \text{otherwise.} \end{cases}$$

Condition 12.6. The default value of $Creating_j$ is 1. The only rule changing $Creating_j$ is *Arbitrate*. ♢

```
rule Arbitrate {
   pfor j ∈ {−1, 1} do {
      cond {
(1)       ?  Kind = Latent and Kind_{−j}^{1.5j} >^{j=1} Kind_j^{−j} and Creating_j^{−j} = 0
          !  Kind := Vac
(2)       ?  Kind ≠ Latent and (Kind_{−j}^{j} >^{j=1} Kind or Kind_{−j}^{1.5j} >^{j=1} Kind)
          !  Die(p_0)
          ?  Kind^j ≠ Vac
(3)       !  Creating_j := 0
          ?!   Creating_j := 1
          }}}
```

Part (2) erases a cell if another cell must be put in its place (initiated by a non-dying cell) that is stronger or has the same strength but is initiated from right. Part (1) erases a latent cell fast. This part will be *stronger than any other rule* possibly conflicting with it (which, in other words, would just set some fields of the cell instead of erasing it). The role of the condition $Creating_j^{-j} = 0$ will be seen in the next lemma. According to (3), the rule turns off $Creating_j(x)$ as soon as the job of creating the neighbor $x + jB$ is done.

Remark 12.7. The rule also shows that the default value of $Creating_j$ is 1. Therefore even a latent cell creates a neighbor if nothing is in the way. This will be used for the self-organization properties. ♢

The following lemma shows that the rules *Arbitrate* and *Create* indeed succeed in creating a new cell. Here, cell $x - B$ will create a cell at site x. Creation from the right is analogous. Let

$$\tau_i = (p_i + 1)T^\bullet. \tag{12.1}$$

Lemma 12.8 (Creation)**.** *Assume the following, with* $I = [t_0, t_0 + \tau_0 + 4T^\bullet]$*:*
(a) $[x - B, x + 2B] \times I$ *is damage-free;*
(b) $\eta(x - B, t).Dying = 0$ *and*

$$\eta(x - B, t).Kind_1 \geq \eta(y, t).Kind \vee \eta(y, t).Kind_{-1}$$

for all (y, t) *in* $[x+, x + 2B-] \times I$*;*
Then $\eta(x, t) \neq Vac$ *for some* t *in* I*.*

Proof. Let us assume, on the contrary, that x is vacant during all of I, and we will arrive at a contradiction.
The conditions and the rule *Arbitrate* imply that we have $\eta(x - B, t).Creating_1 = 1$ at some $t_1 \leq t_0 + \tau_0$ and this stays so while x is vacant. Now the rule *Create* requires that x become a cell. However, Condition 8.8 (Computation Property) requires x to actually become a cell by time $t_1 + 2T^\bullet$ only if the body of no existing cell intersects with the body of x: see case (h). Suppose therefore that some cell y intersects the body of x during $[t_1, t_1 + 2T^\bullet]$.
1. Suppose that y does not disappear before $t_0 + \tau_0$.
Then part (3) of rule *Arbitrate* implies that during this time, $Creating_{-1}(y+B)$ becomes 0, hence $y + B$ loses the ability to recreate y fast. Part (2) of the same rule makes y latent within τ_0 time units. Part (1) then erases y within $2T^\bullet$ units of time.

Any new cell y' causing a similar obstacle to creating x could arise only if $y + B$ creates it. Indeed, the only other way allowed by the Computation Property is case (g) there; however, this case, reserved for the possible appearence of a latent cell out of "nothing", (indeed, out of lower-order germs) requires $\vartheta_j(y', t)$ to be vacant for all j, i.e. that y' have no (adjacent or non-adjacent) neighbors. But, $x - B$ would be such a neighbor, so y' will not appear.

Thus, x will be created within $2T^\bullet$ time units after the disappearence of y.
2. Suppose that y disappears before $t_0 + \tau_0$.
If it does not reappear within $2T^\bullet$ time units then x will be created as above. Suppose therefore that y reappears. When it reappears we necessarily have $Creating_{-1}(y + B) = 1$. This turns 0 within τ_0 time units. After it turns 0, the rule *Arbitrate* erases y within $2T^\bullet$ units of time and then x will be created in $2T^\bullet$ time units. Cell y will not be recreated to prevent this since $Creating_{-1}(y + B) = 0$ for at least $p_0 T_\bullet > 4T^\bullet$ time units.

□

12.4. **Animation, parents, growth.** A latent cell **x** can come to life by a rule in three ways: by the rules *Animate*, *Heal*, or by starting to develop as a germ cell. The subrule $Animate(j, p, \dots)$, $j \in \{-1, 1\}$, gives the new cell the appropriate field values.

The field *Becoming* (default value 0, reset when *Animate* is not applicable) is used to slow animation by its assignment parameter p. The condition $Dying = 0$ in the rule below makes sure that (in absence of damage) the mother is still alive when **x** becomes live. The subscript p_0 in the last assignments makes sure that once the decision is made to revive the cell, its age is set fast

enough in order that it does not stay much behind the age of the creating cell. This way, the created cell becomes a sibling of the creating one.

```
subrule Animate(j, p, F_1, v_1, F_2, v_1, ...) {
   cond {
      ?  Kind = Latent and Dying^j = 0 and (
         (ϑ_{2j}(x) is a sibling of ϑ_j(x) and Dying^j = Dying^{2j} = 0)
         or (Kind^j = Germ and Addr^j = −Q and Dying^j = 0 and
         there is no non-germ neighbor towards −j) )
      !  cond {
         ?  Becoming = 0
         !  Becoming :=_p 1
         ?! ‖ pfor i = 1, 2, ... do F_i :=_{p_0} v_i }
      ?! Becoming :=_{p_0} 0
      }}
```

The cell $\vartheta_j(\mathbf{x})$ used here is called the *mother cell*. In case the same result could also have arisen using cell $\vartheta_{-j}(\mathbf{x})$ then the mother cell is the one closer to the center of its colony.

The following lemma is immediate from the definition of the animation rule.

Lemma 12.9 (Animation Support)**.** *Suppose that*

(a) *a cell x has just been animated at time t by a non-germ neighbor $y = \vartheta_j(x)$ (this rule, with observation time t' being a possible explanation for its becoming live);*
(b) $x + [-3B, 4B] \times t + [-3T^\bullet, 0]$ *is damage-free;*
(c) *there is no colony-boundary between y and its sibling required by the rule ;*

Then y and its two siblings survive until after t.

Proof. The animation requires a sibling for y with both y and the sibling non-dying. Due to the minimum delay p_0 in dying which they did not even begin, these cells remain live till after t. Since there is no colony boundary between them, a cut will not break the sibling relation of these cells either. □

Remember that whether a cell has kind *Channel* or *Growth* can be determined from its age. Therefore it is sufficient to have one value

$$Ext_j$$

in place of $Growth_j$ and $Channel_j$. Rules *Extend* and *Grow* both rely on the following subrules.

```
subrule Grow_step.active(j) {
   cond {
      ?  Kind = Germ and j points away from the colony center
      !  Kind_j.Grow := Germ
      ?  (Addr = e_j or Kind = Ext_j) and Sib(−j)
      !  Kind_j.Grow := Ext_j
      ?! Kind_j.Grow := Latent
      }}
subrule Grow_step.passive(j) {
   cond {
      ?  Kind = Latent and Kind_j^{−j} ∈ {Germ, Ext_j}
      and Kind_j^{−j} ^{j=1}> Kind_{−j}^j
      !  if possible, make x consistent with ϑ_{−j}(x) using
```

$Animate(-j, p_2, Kind, Growth_j, \dots)$
}}

The rule *Extend* serves to extend the channel in direction j during the computation time of the colony.

```
rule Extend {
   pfor j = −1, 1 do {
      cond {
         ?  Age ∈ [0, (compute_start) − 1]
         !  Grow_step.active(j)
         ?  Age^{−j} ∈ [0, (compute_start) − 1]
         !  Grow_step.passive(j)
         }}}
```

The rule *Grow* depends on the fields $Growing_j$. The computation rule (to be defined below) turns this field to 1 in all cells of the colony iff the field $Creating_j$ in the big cell represented by the colony has value 1. Otherwise, it will be 0 in all cells. The healing rule, to be given later, keeps this locally maintained field constant throughout the extended colony.

```
rule Grow {
   pfor j = −1, 1 do {
      cond {
         ?  Age ∈ [(grow_start), (grow_end) − 1]  and  Growing_j = 1
         !  Grow_step.active(j)
         ?  Age^{−j} ∈ [(grow_start), (grow_end) − 1]  Growing_j^{−j} = 1
         !  Grow_step.passive(j)
         }}}
```

For germs, the growth rule is similar. However, growth will proceed always to the right and the time intervals in which it occurs will be defined later, in the germ program.

12.5. **Healing.** Let $F_1 := v_1, \dots, F_k = v_k$ be an assignment changing some fields of $\mathbf{x}$. This assignment is an *internal correction* if the following conditions hold:

(a) For $j \in \{-1, 1\}$, we have $Dying^j = 0$. Also, $Frozen^j = 1$ unless $\vartheta_j(\mathbf{x})$ is a sibling of $\mathbf{x}$.
(b) After the assignment, but not before, $\mathbf{x}$ and its two neighbors form a domain in which the following holds: for each locally maintained field F, if Age is not in the update interval of F (as defined in Subsection 11.4) then $F = F^j$ for $j \in \{-1, 1\}$.
(c) In each direction j, the domain, with constant values for the locally maintained fields, continues to the second neighbor unless the first neighbor is a protected colony endcell towards j.

If $\mathbf{x}$ is latent and one of its neighbors is an endcell of its colony protected in the direction away from $\mathbf{x}$ then the internal correction will be called a *near-end-correction*. An assignment to make a non-germ cell is an *end-correction to the right* if the following holds:

(a) $Dying^{-1} = 0$. Also, $Frozen^{-1} = 1$ unless $\vartheta_{-1}(\mathbf{x})$ is a sibling of $\mathbf{x}$.
(b) After the assignment, but not before, the following holds: $\mathbf{x}$ is the right-protected right endcell of its colony with $\vartheta_{-1}(\mathbf{x})$ in its domain, with $Age = Age^{-1}$, and for each locally maintained field F, if Age is not in the update interval of F (as defined in Subsection 11.4) then $F = F^{-1}$;
(c) The domain, with constant values for the locally maintained fields, continues to $\vartheta_{-2}(\mathbf{x})$.
(d) If $\mathbf{x}$ is a right outer cell then $Age \geq$ (grow_end);

End-corrections in the left direction are defined similarly. Note that end-correction does not create a germ cell. Each of the corrections above is called *mild* if it can be achieved by only changing some

locally maintained fields different from *Age* and *Addr*. Internal corrections are called *weak* if (c) is not required. Let us denote by

$$Int\_corr(x) = 1,\ End\_corr_j(x) = 1$$

the fact that an internal correction is possible in x or that an end-correction in direction j is possible in x. The fact that a weak internal correction is possible will be denoted by

$$Int\_corr'(x) = 1.$$

We need the notion of weak internal correction since cell $\mathbf{x}$ cannot check $Int\_corr(\vartheta_{-1}(\mathbf{x}))$ directly, only $Int\_corr'(\vartheta_{-1}(\mathbf{x}))$.

```
rule Heal {
   cond {
      ?  Int_corr(x) = 1 and
         if Int_corr'(ϑ_{-1}(x)) = 1
         then that correction would result in a weaker cell ϑ_{-1}(x)
      !  cond {
         ?  the correction is mild
         !  carry it out
         ?  Kind ≠ Latent
         !  Die(p_0)
         ?! correct using Animate(j, p_0, ...) for some j ∈ {-1,1}  }
(1)   ?  ∃j ∈ {-1,1} End_corr_j(x) = 1 and Int_corr'(ϑ_{-j}(x)) = 0 and
         x is no endcell in direction -j and End_corr_{-j}(x) = 0 and Kind_j^{-j} >^{j=-1} Kind_{-j}^{j}
      !  cond {
         ?  the correction is mild
         !  carry it out
         ?  Kind ≠ Latent
         !  Die(p_0)
         ?! correct using Animate(j, p_0, ...)  }
      ?  ∃j ∈ {-1,1} there is an end-correction in direction j in ϑ_j(x) (then j is unique)
(2)   !  Kind_j.Heal := Kind;
      ?! pfor j ∈ {-1,1} do Kind_j.Heal := Latent;
      }}
```

Part (1) uses the fact that our cells have a reach greater than 1, seeing their second neighbors. If an internal correction will be carried out then of the two neighbors of $\mathbf{x}$, the one closer to the center of the colony is called the *mother* of $\mathbf{x}$ while the one farther from the center is called the *father*.

```
rule Purge {
   cond {
      ?  x is isolated and is not an endcell of a near-end-correction
      !  Die(p_0)
      }}
```

Condition 12.10 (Animation).

(a) The only rules creating a live cell are *Animate* and *Birth*;
(b) *Animate* will be applied only by *Heal* or *Grow_step*, and it will never create an exposed cell;
(c) The birth rule will be applied to cells with no live neighbors;

♢

Condition 12.11 (Killing).

(a) The rule *Die*(p) is invoked always with $p \geq p_0$;
(b) A cell can only be made vacant by the rule *Arbitrate*;
(c) Only the following rules can kill a cell: *Heal*, *Arbitrate*, *Decay*, *Purge*;
(d) A non-exposed edge can be killed only if a neighbor has $Kind_j \neq Latent$. Setting $Kind_j \neq Latent$ happens only in the growth rules and in end-healing;

♢

12.6. **Continuity.** Our terminology turns out to be incestuous: a child cell can only be created if it also becomes a sibling.

Lemma 12.12 (Parent)**.** *Suppose (x, u) is not in any extended damage rectangle, and x becomes animated. Then there is a $j \in \{-1, 1\}$ and $t' \in [u - 3T^\bullet +, u - T_\bullet/2]$ with the following properties.*

(a) *Some rule is applicable at time t' invoking the animation of x with mother $x + jB$.*
(b) *$x + jB$ has a state at time t' that makes it a sibling of (x, u). If the internal correction case of the healing rule applies then $(x - jB, t')$ is a father of (x, u);*
(c) *For any time $t \in [u - T^\bullet, u]$, if x and its mother (resp. father) are not in any extended damage rectangle during $[t', t]$ then they are siblings at t;*
(d) *If x is not affected by the damage via neighbors at time u then we can choose $t' > u - T^\bullet$. Also, t' can be chosen anywhere before $u - T^\bullet$ if this is also before the time projection of the damage rectangle.*

Proof. Since x is not affected immediately by damage, legality implies that the change is the delayed result of the application of *Animate*.

1. Let us prove (a) and (b) first.

Suppose first that x is not affected by the damage via neighbors at time u. Then at the observation time t' corresponding to the switch (x, u), we have the situation described by (b).

Suppose now that x is affected by the damage via neighbors at time u. Then, according to Lemma (11.1), it is not affected via neighbors during $[u - 3T^\bullet, u - 2T^\bullet]$. Let t be a switching time of x in this interval (in a moment, we will see that t exists.) Then at the observation time t' corresponding to the switch (x, t), we have the situation described by (b). Indeed, there are at most 4λ steps of x between t and u; but the delay parameter of *Animate* is at least p_0, so since (11.4) implies $4\lambda < p_0$, the observation time t' must have occurred during the wait.

2. Let us prove (c).

According to Condition 12.10 (Animation), the applied rule was either *Heal* or *Grow_step*. In the healing case, the parents are frozen and non-dying, therefore they will not change their age for a while yet in any way but healing.

In the growth case, the age of the child is made equal to the age of the mother at a not much earlier time t'' since once animation has been decided the assignment during animation happens fast (with delay p_0). Therefore the mother has time for at most one increase (with delay p_2) of age in $[t''-, u]$, and this will not break the sibling relation. The mother (and father) does not die since the rule *Die* would have announced this at least $p_0 T_\bullet$ time units earlier (see Condition 12.11) via the field *Dying* and this would have turned off the animation or healing of x ((11.4) implies $p_0 T_\bullet > 2T^\bullet$).

□

Lemma 12.13 (Glue)**.** *Suppose that the adjacent cells $x, x + B$ are siblings at time t_0, and the damage does not intersect the rectangle*

$$(x, t_0) + [-2B, 3B] \times [-4T^\bullet, T^\bullet].$$

Suppose also that at the next moment that is a switching time of one of them, this is a switching time of x and this breaks the sibling relation. Then we have the following possibilities:

(1) *a cut;*
(2) *$x - B$ was not a sibling of x at the last observation time of x and the switch kills x by Heal, Decay or Arbitrate;*

Proof.

1. If a cell x breaks a sibling relation by a rule then one of the cases listed in the statement of the lemma holds.

   This follows from the definition of siblings, Conditions 11.4 (Address and Age), 12.11 (Killing) and the healing rule.

We will show that if neither of the possibilities listed in statement of the lemma holds then the cells remain siblings.

2. Suppose that x or $x + B$ were animated at some time in $[t_0 - T^\bullet, t_0]$; without loss of generality, suppose the latest such time was t_1 and the cell was $x + B$.
   Then $x + B$ will be without an age change for at least $p_0 T_\bullet$ time units which, due to the fact that (11.4) implies $p_0 \geq 4\lambda$, is longer than the whole period under consideration. If x also underwent animation during this interval then the same is true for it, hence the two cells remain siblings. Suppose therefore that x has been live during $[t_0 - T^\bullet, t_0]$. The rule *Animate* implies that $x + B$ is not a germ unless x is one of the parents. If x is a parent of $x + B$ then Lemma 12.12 (Parent) implies that the two cells remain siblings for at least $T^\bullet$ time units; after this, both cells have seen each other as siblings and therefore Condition 11.4 (Address and Age) shows that they remain siblings until a cut or a death.

   Suppose that x is not a parent of $x + B$. If it has changed its age within the last $4T^\bullet$ time units then it will not change the age for a long time after, and the two cells remain siblings. If it has not changed its age within this time then for at least $2T^\bullet$ time units before the observation time before the animation, it already is a partner of the mother $x + B$. The rule *Freeze* of Subsection 12.1 implies then that x is frozen which keeps x and $x + B$ siblings.
3. Suppose now that both cells have been live during $[t_0 - T^\bullet, t_0]$.
   If x changes its age within this time then it will not change its age soon and therefore remains a sibling. Suppose therefore that x does not change its age during this time. If $x + B$ was a sibling all the time during $[t_0 - T^\bullet, t_0]$ then x sees that $x + B$ is a sibling and will not break the sibling relation. Suppose therefore that $x + B$ changes its age within this interval and becomes a sibling of x this way. Then x had ample time before this age to observe that $x + B$ is a partner. Therefore x is frozen and will not change its age at the next switch.

□

Remark 12.14. The lemma essentially also holds if we exchange left for right and $x + B$ for $x - B$, with the following modification:

In this case there is yet another possibility for x to break the sibling relation by a rule: namely, when x is a doomed right end membercell of its colony turning to $Age = 1$. Indeed, case (1) of the definition of siblings says x is not a sibling of its left neighbor anymore if this happens. ♢

Lemma 12.15 (Exposing)**.** *An edge turns into an exposed one by a rule only in the following cases:*

(1) *doomed right end membercell of its colony turning to $Age = 1$;*
(2) *channel cell, $Age = (\text{idle\_start})$;*
(3) *outer germ, $Age = (\text{germ\_end}) - 2Q$;*

(4) *growth, non-end,* $Age = (\text{grow\_end})$*;*
(5) *germ cell,* $Age = (\text{germ\_grow\_end})$*, except when it is an outer edge of an outer colony endcell;*

Proof. Direct consequence of the definition of siblings and exposed edges and Conditions 11.4 (Address and Age), 12.11 (Killing). □

13. Gaps

The main lemma of this section is the Running Gap Lemma, saying that if a sufficiently large gap is found in a colony then this gap will not be closed, but will sweep through it, essentially eliminating a partial colony, and serving as a preparation to the Attribution Lemma of the next section. We start by collecting here some constants and inequalities, for later reference. For clarity, we omit the notation $\lfloor\cdot\rfloor$ for integer part. Recall that by definition, $End-$(grow_start) and $End-$(grow_end) are constant. We use a constant K_1 that can be computed from the program, see (16.7) below. Recall the definition $\tau_i = (p_i+1)T^\bullet$ in (12.1).

$$p_0 = 4\lambda + 1, \tag{13.1}$$

$$p_1 = 4\lambda p_0. \tag{13.2}$$

$$p_2 = 2\lambda p_1, \tag{13.3}$$

$$(\text{split\_t}) = 14\tau_1 + T_\bullet, \tag{13.4}$$

$$Q > 250 > 12(\text{split\_t})/\tau_1, \tag{13.5}$$

$$(\text{compute\_start}) = 3Qp_2/p_1 = 6Q\lambda, \tag{13.6}$$

$$(\text{compute\_time}) = K_1 QCap/w, \tag{13.7}$$

$$(\text{idle\_start}) = (\text{compute\_start}) + (\text{compute\_time}); \tag{13.8}$$

$$(\text{synch\_start\_lb}) = (\text{idle\_start})(1+\lambda), \tag{13.9}$$

$$U \geq ((\text{synch\_start\_lb}) + Q)p_1, \tag{13.10}$$

$$(\text{end\_period}) = 6Qp_1\lambda, \tag{13.11}$$

$$(\text{grow\_start}) = End - (\text{end\_period}) + 4Q, \tag{13.12}$$

$$(\text{grow\_end}) = (\text{grow\_start}) + 6Q\lambda, \tag{13.13}$$

$$(\text{crit}) = 3Q\tau_1, \tag{13.14}$$

$$(\text{germ\_grow\_end}) = 12Qp_1\lambda, \tag{13.15}$$

$$(\text{germ\_end}) = (\text{germ\_grow\_end}) + (\text{end\_period}).$$

Inequality (13.9) holds in view of (9.16), if R_0 is large enough. Before, (synch_start_lb) is a lower bound on the value of *Age* when *End* will be computed. Similarly, (synch_start_lb) + (end_period) is a lower bound on *End*.

13.1. **Paths.** Suppose that $t < u$, cell x is live at t and is not in any extended damage rectangle during the interval $[t,u]$, and there are no switching times in $[t+,u-]$. Then we say that (x,t) is connected by a *vertical link* to (x,u).If one end of a vertical link is not a switching time then the link is called *short*.If cells $x, x+B$ are siblings not in any extended damage rectangle at time t or $t-$ then the points (x,t), $(x+B,t)$ are said to be connected by a *horizontal link* (of size 1).Also the pair $(x,t),(x+2B,t)$ is said to be connected by a *double horizontal link* if there is a near-end-correction for $(x+B,t)$.If (y,t') is, according to case (b) of Lemma 12.12 (Parent) a mother or father of (x,u), we will say that the point (y,t') is connected by a *parental* (maternal or paternal) link to point (x,u).By this lemma (under the appropriate damage-free condition), the parent survives until the birth of the child, and therefore the parental link can be replaced by a horizontal link and some vertical links. A *link* is a link of one of these kinds. A link is *steep* or, equivalently, *slow* if it is a non-short vertical link or a parental link.(Since time is the second coordinate, steepness of a line is synonymous to slowness of the movement of a point along it.) A sequence (x_0,t_0), ..., (x_n,t_n) with $t_i \leq t_{i+1}$ such that subsequent points are connected by links, is called a *path*. A *forward* (n,k)*-path* is a path of length n whose number non-steep links is at most k. The adjective "forward" will be

omitted when it is obvious from the context. A $(n, 0)$-path is *steep*, or *slow*. A *backward path* is the reversed reading of a forward path, backward in time. Notice that a point (x_i, t_i) on a path can actually be dead, if it has just died: indeed, it can be connected e.g. to (x_{i+1}, t_{i+1}) by a horizontal link such that $t_i = t_{i+1}$ and x_i, x_{i+1} are siblings at t_i-.

For a path $P = (x_0, t_0), \dots, (x_n, t_n)$ and $t \in [t_0, t_n]$, let

$$P(t)$$

be x_i with the smallest i such that $t \in [t_i, t_{i+1}]$.

The following statement follows immediately from the definition of paths.

Lemma 13.1. *For the time projection d of an (n, k)-path we have*

$$d \geq (n - k)T_\bullet/2.$$

The following lemma says that if two paths cross then they intersect.

Lemma 13.2 (Crossing)**.** *Let $(x_1, s_1), \dots, (x_m, s_m)$ and $(y_1, t_1), \dots, (y_n, t_n)$ be two paths with $s_1 = t_1$, $s_m = t_n$, $x_1 \leq y_1$, $x_m \geq y_n$. Then there are i, j such that $x_i = y_j$ and either $t_j \in [s_i, s_{i+1}]$ or $s_i \in [t_j, t_{j+1}]$.*

Proof. Parental links can always be replaced with horizontal and vertical links. Horizontal links of size 2 jump over a latent cell only. So, paths cannot jump across each other. □

According to the Parent Lemma, a steep path can be continued backward in time until it hits some extended damage rectangle. Moreover, occasionally we have a choice between continuing to the mother or to the father. Let

$$(\text{wake}) = \tau_0 + 2T^\bullet. \tag{13.16}$$

If $[a_0, a_1-] \times [u_0+, u_1-]$ is an extended damage rectangle then let

$$u_2 = u_0 + (\text{wake}). \tag{13.17}$$

The rectangle

$$[a_0, a_1-] \times [u_0+, u_2-] \tag{13.18}$$

will be called the *wake* of the damage rectangle in question. The lemma below says that unless a path started backward in the wake of a damage rectangle, it can be diverted and continued back past a damage rectangle.

Lemma 13.3 (Ancestor)**.** *Let (x_0, t_0) be a live point not in* (13.18)*, with x_0 in $[a_0 - QB/4, a_0 + QB/4-]$ and t_0 in $[u_0, u_0 + T_\bullet{}^*/2]$.*

Then, there is a path going backward from (x_0, t_0) and ending either in u_0 or in a birth. It is an $(n, 2)$-path for some n with at most 1 horizontal link. In constructing the path backwards, we are free to choose between maternal and paternal links at all times but possibly once, when moving out of $[a_0, a_1]$. These times, as well as the horizontal link, may only occur during $[u_0+, u_2-]$.

Proof. We call the path to be constructed a *desired path*. Let us start constructing a steep path $c_0, \dots, c_n$ with

$$c_i = (x_i, t_i)$$

backward from (x_0, t_0). If we get to u_0 then we are done, otherwise, we stop just before we would hit $u_1 + T^\bullet-$, with c_k being the last element. Then $t_k < u_1 + 2T^\bullet$ and $x = x_k \in [a_0, a_1]$. Indeed, if this is not so then we could continue the path either by a vertical or by a parental link. The vertical link would be shorter than than $T^\bullet$, and the parental link would lead to a damage-free cell, so either of them would be allowed.

Let us now go back on the path for $i = k, k-1, \ldots$ until the first i (counting from k) such that either c_i is a parent of c_{i-1} or $x_i = x$ can have a horizontal link at some time during $[t_i, t_{i-1}]$. There will be such an i, since otherwise the cell x would be isolated throughout $[u_1 + 2T^\bullet, u_2]$ with no near-end-correction in a neighbor, and the rule *Purge* would kill it by the time u_2.

Suppose that c_i is not a parent: then let $y_0 = x$ and let w_0 be the earliest time in $[t_i, t_{i-1}]$ when x has a sibling $y_1 \neq x$. Let $w_1 = w_0$. Suppose that c_i is a mother: then it has a sibling $y_1 \neq x$. Let $w_1 = t_i$ in this case. Suppose that c_i is a father: then let $y_1 \neq x$ be the corresponding mother with $w_1 = t_i$.

Let P_1 be the part of the original path until c_{i-1}, and P_2 the new part ending in (y_1, w_1). Let us build a steep path (y_j, w_j), $j = 1, 2, \ldots$, backwards until either $w_j < u_0$ or $y_j = x$ and let P_3 be the part of this path ending with (y_{j-1}, w_{j-1}). If $w_j < u_0$ then P_1, P_1, P_3 combine to a desired path, suppose therefore that $w_j \geq u_0$, hence $y_j = x$. Then (x, w_j) is a parent of (y_{j-1}, w_{j-1}), hence by the Parent Lemma, x is a sibling of y_{j-1} at time t_{j-1}. By definition, if t is the first time after t_k when x has a sibling then $t \geq t_i$; hence $w_{j-1} < t_k < u + 2T^\bullet$. By the Parent Lemma, $(x, u_0 - T^\bullet)$ is also parent of (w_{j-1}, t_{j-1}) (by the same animation): choosing this as (y_j, w_j), we are done. □

13.2. **Running gaps.** The rule *Decay* attempts to achieve that if some "relevant gap" was not closed in reasonable time then it becomes wider.

```
rule Decay {
   cond {
      ?  ∃j ∈ {−1, 1}Xposed(j)
      !  Die(p1)
      }}
```

Remark 13.4. Most killing will be done by this rule. The rule *Purge* is important only when the damage erases a colony endcell and creates an isolated cell, intersecting its former body. The decay rule (in this simple form) would take too long to eliminate the new cell which could in the meantime widen the gap (and make it unhealable). ♢

Consider an interval $G = [l+, r-]$ where $0 < r - l$ is divisible by B. Assume that if the wake of a damage rectangle with space projection $[a_0, a_1-]$ intersects G then $l < a_0 < a_1 < r$. We call G a *gap with right-age* n if n is the smallest number k such that every cell in G space-consistent with r and not in the wake of a damage rectangle is a germ cell with age $< k$. The right age is infinite if there is no such number k (these cases will not be relevant). The *size* of G is $r - l - B$. If G is contained in the colony of r then it is called an *interior gap*. As we see, G is assumed to be a gap only in the set of cells that are space-consistent with r and are not young germ cells. (It is probably not important that we exclude only cells space-consistent with r.)

Suppose that in a time interval $[v_0, v_1]$, the gap $G(t) = [l(t)+, r(t)-]$ is defined for all t, in such a way that all cells $r(t)$ are space-consistent with each other, and for all t_1, t_2 with $|t_2 - t_1| \leq 3T^\bullet$, we have $r(t_2) - l(t_1) > B$. Then $G(t)$ is called a (right) *gap path*, and the *right age* of the gap path is the maximum of the right ages of the gaps in it. By this definition, if a path space-consistent with $r(t)$ has the same time projection $[v_0, v_1]$ as the right gap path $G(t)$ and has no germ cells younger than the right age of the gap path then it cannot cross $G(t)$ since no parental link can jump through it. Though the wake of a damage rectangle in $G(t)$ may contain some cells that are not young germ cells these do not live long enough to become parents (due to the wait of animation) and therefore also cannot assist in the jumping of a path.

The lemma below says that the *Decay* rule causes a large enough gap to move right rather fast. The gap is assumed to be connected, via a series of horizontal links, to a forward path. This excludes irregular and unimportant cases when the gap would have to travel through all kind of debris.

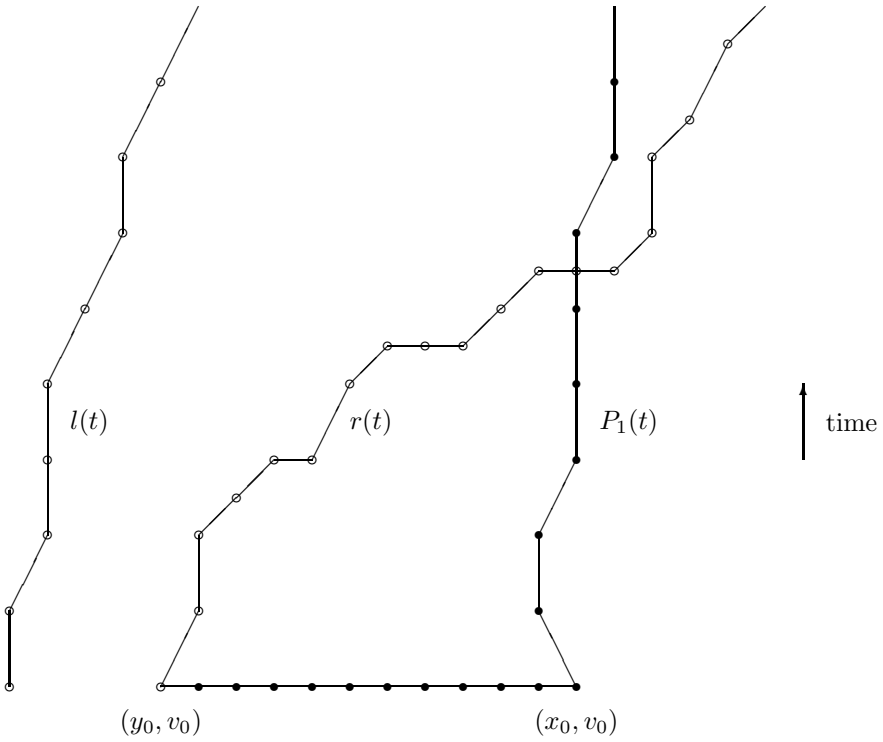

FIGURE 16. Running Gap Lemma

Lemma 13.5 (Running Gap)**.** *Let $P_1 = (x_0, v_0), \ldots, (x_n, v_n)$ be a forward path with at most one double horizontal link, let L, k be positive integers with*

$$L \le 3Q,$$
$$k < (\text{synch\_start\_lb}) + (\text{end\_period}) - 5Lp_1\lambda.$$

Assume the following:

(a) *(x_n, v_n) is not a germ cell younger than $< k + 10L\lambda p_1$;*
(b) *(y_0, v_0) is to the left of (x_0, v_0) in the same domain;*
(c) *if y_0 is a left outer cell then P_1 is in the same colony;*
(d) *No damage rectangle affects $[y_0+, x_0-] \times \{v_0\}$, even via neighbors;*
(e) *(y_0, v_0) is a left exposed edge (see Subsection 12.1);*
(f) *y_0 has been the right end of an interior gap of right-age k and size $> 5B$ during $[v_0 - T^\bullet, v_0]$;*
(g) *$v_n - v_0 \le 2L\tau_1$;*

Then during $[v_0, v_n]$, a right gap path $G(t) = [l(t)+, r(t)-]$ can be defined, with $r(v_0) = y_0$,

$$(r(v_n) - y_0)/B \ge (v_n - v_0 - (\text{wake}))/\tau_1 - 4, \tag{13.19}$$

$r(t) \ge y_0 - B$ and the right ageof the gap path is $< k + 5L\lambda p_1$. If no damage rectangle occurs on the whole interval during the time considered then the right-hand side of (13.19) *can be replaced with $\lfloor (v_n - v_0)/\tau_1 \rfloor$.*

Before proving this lemma, let us prove a corollary saying that if P_1 is long there is no large young gap next to its beginning (say, on the left).

Corollary 13.6 (Running Gap)**.** *Assume the conditions of the Running Gap Lemma, with $L = Q$, and assume also that the path P_1 remains in the colony of y_0. Then we have*

$$v_n - v_0 \le Q\tau_1. \tag{13.20}$$

Proof. It is easy to see that the length of the path is at most $5L\lambda p_1$. By the definition of the gap path $G(t)$ above, path P_1 starts on its right. Since age varies by at most 1 per link along a path (except for a double link) and since (x_n, v_n) is not a germ cell younger than $k+10L\lambda p_1$, no cell on P_1 is a germ cell with age $< k+5L\lambda p_1$, while according to Lemma 13.5, all cells in $G(t)$ space-consistent with $r(t)$ are germ cells with such age. Therefore P_1 never crosses the gap path from right to left. The inequality (13.19) gives a lower bound on how fast $r(t)$ moves right. Since P_1 stays in the colony of y_0,

$$(v_n - v_0 - (\text{wake}))/\tau_1 - 3 < Q - 5,$$
$$v_n - v_0 < \tau_1(Q-2) + (\text{wake}) < \tau_1 Q$$

(see (13.1) and (13.16) for the last inequality). □

Proof of Lemma 13.5. Let $r(v_0) = y_0$, and let $l(v_0)$ be the leftmost cell such that the gap $[l(v_0)+, r(v_0)-]$ has right-age $\le k$.

1. Let $t_1 > v_0$. Assume that for all $t \le t_1$, a gap path $G(t)$ is defined with the desired properties and in such a way that $(r(t_1), t_1)$ is not a germ cell younger than $k + 10Lp_1\lambda$ and is not in the wake of a damage rectangle. Then $r(t_1)$ is a edge space-consistent with $r(t_0)$ that is either exposed or is to become exposed in one step, in one of the cases of Lemma 12.15 (Exposing). This last case occurs only if there is a damage rectangle during $[v_0, t_1]$.

   Proof. Since $(r(t_1), t_1)$ is not in the wake of a damage rectangle we can build a backward $(m, 2)$-path

   $$P_2 = ((z_0, w_0) = (r(t_1), t_1), \dots, (z_m, w_m))$$

   according to the Ancestor Lemma. Lemma 13.1 and the bound (g) gives

   $$m \le 2 + 2(v_n - v_0)/T_\bullet \le 2 + 4Lp_1\lambda < 5L\lambda p_1.$$

   The backward path P_2 ends in at time v_0. Indeed, if it ended in a birth then $(r(t_1), t_1)$ would be a germ cell younger than $5Lp_1\lambda$ contrary to the assumption. P_2 does not cross the gap since otherwise $(r(t_1), t_1)$ would be a germ cell younger than $k + 5L\lambda p_1 + 5L\lambda p_1$ which was excluded. Therefore defining $x = P_2(v_0)$ we have $x \ge y_0$. Without loss of generality, we can suppose $x \le x_0$. Otherwise, Lemma 13.2 (Crossing) implies that path P_2 crosses P_1 and we can switch from P_2 to P_1 at the meeting point. Thus, x is on the interval $[y_0, x_0]$, aligned with x_0.

   Combine P_2 and the horizontal path from (x, v_0) to (y_0, v_0) into a single chain of links connecting siblings. This chain has at most $3Q$ horizontal links from the left edge (y_0, v_0) to (x, v_0), and then at most $2 + 2(v_n - v_0)/T_\bullet$ links on P_2, giving fewer than

   $$3Q + 2(v_n - v_0)/T_\bullet + 2$$

   links. The assumption that the gap path satisfies the requirements of the Lemma implies $r(t) \ge y_0 - B$, hence P_2 is to the right of the left end of the colony of y_0.

   In the number of steps available, a protected left edge cannot occur on such a path unless y_0 is a left outer cell and the path leaves the colony of y_0 on the right. Indeed, suppose first that (y_0, v_0) is a member or right outer cell. For a path of ancestors starting from such a cell to turn

into a left outer cell this path would first have to walk right to participate in the creation of a colony and then walk back through the growth process from that colony, which is impossible due to (13.8).

Suppose now that (y_0, v_0) is a (exposed) left outer cell. The age of cells on the path is locally monotonically nondecreasing, except when it crosses into another work period. Indeed, since the age of siblings is required to be monotonically nondecreasing towards the originating colony, it is nondecreasing on the horizontal part of the path; it is also nondecreasing on the part P_2 constructed by the Ancestor Lemma, except on the horizontal link allowed by that lemma. Therefore the age of $(r(t), t)$ cannot be smaller by more than 1. If the age is not smaller at all then $(r(t), t)$ is also exposed. If decreasing the age by 1 makes it protected then we must have one of the cases in Lemma 12.15.

Suppose that (y_0, v_0) is a germ cell. Left germ cells inside a colony, as well as member cells they can turn into on the path P_2 are always exposed. We should not rely on this asymmetry in the definition of "exposed" for germs, however, since we want to apply the lemma also when changing left to right. Let us assume therefore for a moment that left and right is interchanged in the definition of "exposed" for germs. Then a germ cell that is a left edge is exposed to the left only if its age is $\geq$ (germ_grow_end). In this case, the same argument works as for left outer cells above.

2. Consider a time interval $[f_0, f_1]$, assume that the gap $G(t)$ with the desired properties was defined up to time f_0, where $G(f_0)$ has size $> 2B$ and right-age $\leq k$. Assume that for $t \in [f_0, f_1]$, no wake of a damage rectangle intersects the area where we define the path further.

   Then the gap path $G(t)$ can be defined further in $[f_0, f_1]$ in such a way that $G(t)$ has right-age $\leq k + 2(t - f_0)/T_\bullet + 1$ and size

$$\geq r(f_0) - l(f_0) - B + (t - f_0)B/(p_2 T_\bullet),$$

   with

$$(r(t) - r(f_0))/B \geq \lfloor (t - f_0)/\tau_1 \rfloor - 1.$$

   Proof. Let us define $G(t)$ as follows. Suppose that it was defined up to time t_1 and let t_2 be the next time that is a switching time of either $l(t_1) + B$ or $r(t)$. We distinguish the following cases.

   (1) t_2 is a switch of $l(t_1) + B$. If the switch is an animation resulting from *Grow* creating a sibling of $l(t_1)$ then $l(t_2) = l(t_1) + B$, else $l(t_2) = l(t_1)$.
   (2) t_2 is a switch of $r(t)$. If $r(t_1)$ dies then $r(t_2)$ is the closest cell to the right of $r(t_1)$ that is not a germ cell younger than $k + 2(t_2 - f_0)/T_\bullet + 1$ or $k + 10L\lambda p_1$, else $r(t_2) = r(t_1)$.
   (3) In all other cases, we leave $G(t)$ unchanged.

   2.1. $G(f_1)$ has right-age $\leq k + 2(f_1 - f_0)/T_\bullet + 1$.

   Proof. Let z_1 be a live cell in $G(f_1)$ space-consistent with $r(f_1)$, and let us build a sequence (z_1, w_1), (z_2, w_2), ... with $f_1 = w_1 \geq w_2 \geq \cdots$ as follows. Without loss of generality, assume that w_1 is a switching time of z_1. If z_1 is live at w_1- then $z_2 = z_1$ and w_2 is the previous switching time of z_1. Otherwise, either (z_1, w_1) is a newborn germ cell, in which case the sequence ends at (z_1, w_1), or it has a parent (x, u).

   If $x \in G(u)$ then $z_2 = x$, $w_2 = u$. Let us show that otherwise, z_1 was created by the internal correction part of the healing rule. If $x \leq l(u)$ then it could not have created z_1 by the growth rule since by the definition of $l(t)$, this would have increased $l(t)$, bringing (z_1, w_1) outside the

gap. If $x \geq r(u)$ then it could not have animated z_1 by the growth rule since $r(u)$ is an exposed edge. The end-correcting part of the rule *Heal* could not be involved. Indeed, the left end in case could only be the left end of the right neighbor colony of y_0. In this case, $r(u)$ would be a member cell and hence y_0 would be a left outer cell. For that case, however, we assumed in condition (c) of the lemma that the path (hence also the gap path which is to its left) stays in the colony of y_0. (This is the part of the Gap Lemma where we use the fact that we upper-bound only the right-age, i.e. the age of cells space-consistent with $r(u)$: thus, the end-healing of some other colony inconsistent with $r(u)$ can be ignored.)

If the internal correction case of healing created z_1, let (z_2, w_2) be the father of (z_1, w_1), this will be inside the gap. By the parent construction, $w_2 \leq w_1 - T_\bullet$ and $Age(z_1, w_1) \leq Age(z_2, w_2) + 1$. We see that the sequence (z_i, w_i) steps back at least $T_\bullet/2$ time units in every step and the age of (z_i, w_i) can decrease by at most 1 in each such step. Since it can only end in a birth or a germ cell in $G(f_0)$ this and (g) proves the age bound.

2.2.

(a) Edge $r(t)$ does not move left during $[f_0, f_1]$. During the same period, $l(t)$ moves right at most once.

(b) For all $n > 0$ with $t = f_0 + n\tau_0 < f_1$, we have $r(t) \geq r(t_1) + (n-1)B$.

Proof. Let us prove (a) first. By property 2.1, the gap can contain only latent and germ cells. The size of $G(t)$ does not allow *Heal* to decrease it: indeed, it follows just as as in 2.1 above that end-healing cannot operate. Since $G(t)$ is a left gap, the rule *Grow* can decrease it only on the left. After one such decrease, the size is still $> B$. The next application of *Grow* is away by a waiting period of length $p_2 T_\bullet$.

Let us prove (b). Since the conditions of 1 are satisfied, $y = r(f_0)$ is an exposed left edge or is a cell whose age is just one step before the applicability of one of the cases in Lemma 12.15 (Exposing). If y is exposed then the rule *Decay* kills it within τ_1 time units. In the other case, nothing prevents the age of y to increase within τ_1 time units. From now on, the rule *Decay* applies.

To conclude, note that according to 2.2 and (13.2), every $p_2 T_\bullet$ time units, the left edge $l(t)$ moves at most one cell width to the right but the right edge $r(t)$ moves at least two cell widths. Since the size started from $> 2B$, it will remain $> B$. This proves the lower bounds on the size of $G(t)$ and on $r(t)$.

3. Let $[a_0, a_1-] \times [u_0+, u_2-]$ be the wake of the damage rectangle and assume that $u_0 \in [v_0, v_n]$. Assume also that $G(u_0)$ has size $> 4B$ and right-age $\leq k$ at time u_0.

Then the gap path $G(t)$ with the desired properties can be defined for $t \in [u_0+, u_2-]$; moreover, $G(u_2)$ has size $> 2B$ and right-age $\leq k + 2(\text{wake})/T_\bullet + 1$.

We also have $r(t) > r(u_0) - 2B$ for all $t \in [u_0+, u_2-]$.

Proof. Let $k(t) = k + 2(t - u_0)/T_\bullet + 1$.

3.1. Assume that the interval $[a_0, a_1]$ is closer to $l(u_0)$ than to $r(u_0)$.

Then $r(t)$ will be defined just as in the damage-free case.

3.1.1. Let $t \in [u_0+, u_2-]$.

Then we set $l(t)$ to be the first site from the left that, at time u_0, is $\geq l(u_0) \vee a_1$ and is aligned with $l(u_0)$. This defines a gap $G(t)$ of size $> B$ for this time interval. This gap will not decrease during $[u_0+, u_2-]$ since it is too large for *Heal* and *Grow* the damage has insufficient time to trigger *Grow*. So, the age bound reasoning of point 2.1 above applies and the right-age of $G(t)$ will be $\leq k(t)$.

3.1.2. Let $t = u_2$ and $a_0 \leq l(u_0) + B$.

Then $a_1 \leq l(u_0) + 2B$, and we set $l(u_2)$ to be the greatest cell $\leq l(u_0) \vee a_1$ and aligned with $l(u_0)$. Then $l(u_2) \leq l(u_0) + 2B$, hence $G(u_2)$ has size $> 2B$. Again, the reasoning of point 2 above applies to show that $G(t)$ has age $\leq k(t)$. If $l(u_2) > l(u_0)$ then no growth animation could have been pending in $l(u_2)$ from time before u_0 and this completes the proof of the claim in 3.

3.1.3. Let $t = u_2$ and $a_0 > l(u_0) + B$.

Then we define $l(u_2) = l(u_0) + B$. At most one cell can be added to $l(u_0)$ by growth during $[u_0, u_2-]$. Lemma 13.3 (Ancestor) shows that any live cell in $[a_0, a_1-]$ can be traced back to a birth within $G(t)$ or to a time in $G(t)$ before u_0 and therefore the same reasoning as for the other cases gives that $G(t)$ has right-age $\leq k(t)$.

3.2. Assume now that $[a_0, a_1]$ is closer to $r(u_0)$ than to $l(u_0)$.

This case is similar to case (3.1), so we point out only the differences. Now, the gap on the left of a_0 may decrease by one during $[u_0+, u_2-]$, to size > 0, if a growth step occurs on the left. For $t = u_2$, now the cases we distinguish are:

3.2.1. Assume $a_1 \geq r(u_0)$.

Then $a_0 \geq r(u_0) - B$. We set $r(u_2)$ to be the smallest cell $\geq r(u_0) \wedge a_0$ and aligned with $l(u_0)$. Then $r(u_2) \geq r(u_0) - B$, hence $G(u_2)$ has size $> 2B$, even if a growth step occurred at $l(t)$ during $[u_0+, u_2-]$.

3.2.2. Now assume $a_1 < r(u_0)$.

Then we define $r(u_2) = r(u_0)$. Since no growth could have occurred on the right-hand side, by applying the earlier reasoning, we will find that $G(t)$ has right age $\leq k(t)$.

Let us construct $G(t)$ for $t \in [v_0, v_n]$. In the space-time rectangle considered, at most one damage rectangle occurs. Indeed, $v_n - v_0 \leq T_\bullet{}^*/2$ follows from (g) and (13.10).

Let $f_0 = v_0$ and let f_1 be the supremum of the those $t > f_0$ until which the damage-free construction in the proof of 2 is applicable: thus, if there is no damage involved then $f_1 = v_n$, else $f_1 = u_0$. Applying this construction, we get to f_1 with gap size $> 4B$. Applying 2, knowing there is no damage before, we find the age upper bound $k + 2(t - v_0)/T_\bullet + 1$ and the lower bound $\lfloor (t - v_0)/\tau_1 \rfloor$ on $(r(t) - y_0)/B$. If $f_1 = u_0 \leq v_0$ then we apply the construction of 3 to get to $f_0' = v_m \wedge u_2$. Now, the gap has age upper bound $k + 2(t - v_0)/T_\bullet + 2$, and lower bound $(t - v_0 - (\text{wake}))/\tau_1 - 2$ on $(r(t) - y_0)/B$.

Let now f_0' take the role of f_0 and repeat the damage-free step to $f_1' = v_n$. Now, we have age upper bound $k + 2(t - v_0)/T_\bullet + 3$ and lower bound $(t - v_0 - (\text{wake}))/\tau_1 - 4$ on $(r(t) - y_0)/B$. □

14. Attribution and progress

14.1. **Non-damage gaps are large.** A *right* (as opposed to left, not as opposed to wrong) *bad gap* is a gap of right-age $\leq 2(\text{split\_t})/T_\bullet$ and size $> 5B$. The Bad Gap Opening Lemma says that if a left exposed edge persists too long (while possibly moving) then a right bad gap develops on its left. This lemma will be used to prove the Bad Gap Inference Lemma, saying that, under certain conditions, a left exposed edge automatically has a right bad gap next to it.

Let us call two siblings *strong* if either they have been siblings for at least $2T^\bullet$ hours or one cell is a parent of the other. A cell is called e.g. a *weak left exposed edge* if it has no strong left sibling and if it would be a left exposed edge in case it had no left sibling. A sequence $R_1 = (y_0, v_0), \ldots, (y_m, v_m)$ of cell-time pairs will be called a *left boundary path* if it has the following properties:

(a) y_i is a weak left exposed edge during $[t_i+, t_{i+1}-]$;
(b) cell y_{i-1} dies at time t_i and is a strong sibling of $y_i = y_{i-1} + B$ at time t_i-;

Lemma 14.1 (Bad Gap Opening)**.** *Let $\mathcal{C}$ be a colony and let $R_1 = (y_0, v_0), \ldots, (y_m, v_m)$ be a left boundary path in $\mathcal{C}$. Assume that the damage rectangle does not intersect $[\bigwedge_i y_i - B, \bigvee_i y_i + B] \times [v_0, v_m]$. Then y_i has at most $p_1 + 1$ switching times during $[v_i, v_{i+1}]$. If $m \geq 7$ then at time v_m-, there is a right bad gap on the left of $R_1(t)$. The same statement holds if we interchange left and right.*

Proof. Let us show that (y_i, t) is a left exposed edge for t in $[v_i + 2T^\bullet+, v_{i+1} - 2T^\bullet-]$.

Indeed, y_i can be a weak left exposed edge that is not a left exposed edge only if it has a left sibling that is not strong. Now, if y_i does not have a left sibling and it gets one then it follows from Lemma 12.13 (Glue) (and the exclusion of cut, since the edge does not become exposed) that the only way to lose this sibling is if the sibling dies again, which it cannot do before making at least p_0 switches, becoming a strong sibling in the meantime. From this, it is easy to see that y_i can have a left sibling only during a time interval adjacent to either v_i or v_{i+1}. Since the sibling stays weak these time intervals must be at most $2T^\bullet$ long.

Due to the above observation, since y_i is a left exposed edge after its first complete work period following v_i, either *Purge* or *Decay* will kill it within the following p_1 steps. In 7 repetitions of this, a gap of width $7B$ will be created. During this time, by (13.2), at most one growth step can occur on the left, leaving still a gap of size $6B$. □

Lemma 14.2 (Bad Gap Inference)**.** *Let $c_0 = (x_0, t_0)$ be a left exposed edge,*

$$D = [x_0 - 8B, x_0 + B],$$
$$I = [t_0 - (\text{split\_t}) + T_\bullet, t_0]$$

and let z_0 be the starting cell of the colony $\mathcal{C}$ of (x_0, t_0). Suppose that if c_0 is an outer cell then $D \cap ([z_0 - 2B, z_0 + QB-]) \times I$ is damage-free, else $D \times I$ is damage-free. Then one of the following holds:

(1) *There is a bad gap on the left of (x_0, t_0) inside the colony of c_0;*
(2) *There is a bad gap on the left of (x_0, t_0), and a backward path $(x_i, t_i)_{0 \leq i \leq n}$ of length $\leq 7(p_1 + 1)$ with the property that one of the path cells, x_i, is in the left neighbor colony. It is a weak left exposed edge during $[t_i+, t_{i-1}-]$. Also, either it is an expanding germ cell closer than c_0 to the origin of expansion or it is closer than c_0 to the originating colony.*
(3) *There is a backward path of length $\leq 7(p_1 + 1)$ leading from c_0 to a cell undergoing one of the changes listed in Lemma 12.15 (Exposing);*
(4) *There is a backward path of length $\leq 7(p_1 + 1)$ leading from c_0 to a protected left colony endcell, just being killed by a left neighbor cell and exposing a right neighbor.*

The same statement holds if we replace left with right.

Proof. Let us construct a backward path (x_i, t_i) made up of horizontal and vertical links such that cell x_i is a weak left exposed edge during every nonempty time interval $[t_i+, t_{i-1}-]$, and the backward path never passes into the right neighbor colony. Suppose that (x_i, t_i) has already been constructed.

(1) The construction stops in any of the following cases:
- the path has reached length $7(p_1 + 1)$;
- the path moved 7 steps to the left;
- (x_i, t_i) belongs to the left neighbor colony;
- (x_i, t_i-) is not a weak left exposed edge;

(2) If x_i has a switching time t' immediately before t_i such that (x_i, t) is a weak left exposed edge during $[t', t_i]$ then let $(x_{i+1}, t_{i+1}) = (x_i, t')$.

(3) Otherwise, let t' be the lower bound of times t such that (x_i, t) is a weak left exposed edge. If $t' < t_i$ then let $(x_{i+1}, t_{i+1}) = (x_i, t')$.

(4) Assume now that $t' = t_i$. Then at time t_i, cell x_i became a weak left exposed edge without taking an action: hence, it must have lost a strong left sibling. (Animation does not produce any exposed edge, see Condition 12.10 (Animation).) According to Lemma 12.13 (Glue), this can only happen in one of the ways listed there. It is easy to check that of these, only the killing of the strong left sibling produces an exposed edge in x_i. Each of the rules *Heal*, *Decay* and *Arbitrate* that could have killed $x_i - B$ presupposes that this cell did not have a left sibling at the observation time t'' of $x_i - B$; thus, it did not have a strong left sibling at time t_i-. On the other hand, as a strong left sibling of (x_i, t_i-), it has been alive for at least $2T^\bullet$ hours. Let $(x_{i+1}, t_{i+1}) = (x_i - B, t_i)$. Such an i will be called a *right jump*.

By the construction, each jump is surrounded by vertical links. For each jump i, cells x_i and x_{i+1} are strong siblings at time t_i-. Let (x_{i_k}, t_{i_k}) for $i_1 < \cdots < i_m$ be the endpoints of vertical links on the backward path with the property that $(i_k, i_k + 1)$ is not a vertical link; let (x_n, t_n) be the last point of the path. Let us number these points forward in time: $(y_0, v_0) = (x_n, t_n)$, $(y_1, v_1) = (x_{i_m}, t_{i_m})$, $(y_2, v_2) = (x_{i_{m-1}}, t_{i_{m-1}})$, etc., creating a left boundary path.

If the path has moved 7 cells to the left then we are done. If $v_0 \le t_0 - 7\tau_1$ then Lemma 14.1 (Bad Gap Opening) is applicable, and it shows that there is a bad gap on the left of (x_0, t_0). If the path has stopped for some other reason then we have the following cases.

(1) (x_n, t_n) is a weak left exposed edge belonging to a neighbor colony.
(2) (x_n, t_n-) is not a weak left exposed edge.

In case 1, the death of (x_n, t_n) must have created the weak left exposed cell (x_{n-1}, t_{n-1}) that was a left colony endcell. This is possible only if either (x_n, t_n) is an expanding germ cell closer than c_0 to the origin of expansion or it is some other kind of cell in the originating colony of c_0. Continue the construction of the backward path for another $7(p_1 + 1)$ steps (it is easy to see that now it will not be stopped earlier) and apply the Bad Gap Opening Lemma, showing that the present lemma is true for this case.

In case 2 we have either one of the cases in Lemma 12.15 (Exposing), or (x_n, t_n-) dies at time t_n as a left, non-exposed edge. □

The following lemma shows that a colony always makes some kind of progress in the absence of damage.

Lemma 14.3 (Small Progress)**.** *Let $\mathcal{C}$ be a colony with starting cell z_0 and $t_0 < t_1$. If it is covered by a domain whose originating colony it is at time t then let $E(t)$ be the maximal such domain. Suppose that no damage rectangle intersects $[z_0 - 8B, z_0 + (Q + 7)B]$ during $[t_0 - (\text{split\_t}), t_1]$ and $E(t)$ exists during this time.*

Then one of the following statements holds:

(1) *The minimum value of Age in $E(t)$ increases at least $(t_1 - t_0 - 2(\text{split\_t}))/\tau_1 - 3$ steps during $[t_0, t_1-]$ (the minimum must be defined "locally", taking into account the possible crossing of work period boundary).*
(2) *an exposed edge appears in $E(t')$ at some time $t' \in [t_0, t_1]$ and moves towards decreasing $E(t)$ either until it reaches the end of $\mathcal{C}$ or at least by $B(t_1 - t')/\tau_1 - B$;*

Proof. If both endcells of $E(t)$ are protected during $[t_0, t_1]$ then the only thing preventing the increase of age in a protected edge, according to Condition 11.4 (Address and Age), is when x is frozen. According to Condition 12.3 (Freeze), this x can become frozen only when it has a non-dying partner. But Lemma 14.2 (Bad Gap Inference) would be applicable to this partner as an edge turned toward x and it is easy to verify that in the present case, this lemma implies an impossibly large gap between the partners. Therefore the minimum age of $E(t)$ increases every τ_1 steps.
Suppose now, without loss of generality, that the right end of $E(t)$ is exposed at some time $t' \in [t_0, t_1]$. Lemma 14.2 (Bad Gap Inference) implies the existence of a bad gap on the right of $E(t_0)$ unless one of the cases (3) or (4) occurs. If none of these cases occurs then Lemma 13.5 (Running Gap) will widen the gap as predicted in (2).

1. Suppose that case (4) of the Bad Gap Inference Lemma occurs.
   Thus, there is a backward path of length $\leq 7(p_1 + 1)$ leading from c_0 to a protected right colony endcell x, just being killed by a right neighbor cell and exposing a left neighbor. There are only two ways this can happen. One: when x is the endcell of a growth and is killed by the end-healing of member cells of another colony. This case does not really occur. Indeed, let y be the left exposed cell trying to do the end-healing. The Bad Gap Inference lemma can be applied to the y and would imply now, when none of the distracting cases applies, the existence of a large gap between x and y.

   The other case is when x is the endcell of a germ. Then the exposed edge thus created will never become protected, and the Bad Gap Opening lemma implies the creation of a bad and growing gap within (split_t) hours.
2. Suppose that case (3) of the Bad Gap Inference Lemma occurs.
   The cases in Lemma 12.15 fall into two categories. In all cases but (4) and (5), Lemma 14.1 (Bad Gap Opening) applies to the development forward in time from this event, showing that again, the exposed right edge moves left, creating an unhealable gap.

   In case (4), a non-end growth cell at $Age = (\text{grow\_end})$ becomes exposed. The edge may later disappear by healing, but only by end-healing. Indeed, for internal healing a close partner would be needed but the Bad Gap Inference Lemma would show (without the distracting other cases, this time) that the partner cannot be close. In order to prevent a bad gap from opening, the end-healing must succeed within (split_t) hours. Therefore this kind of exposed edge will exist at most for a time interval of length (split_t) on the right and for a similar interval on the left. The same reasoning applies to case (5).

We found that in three intervals outside the possibly two exception intervals, the minimum age of $E(t)$ increases every τ_1 steps. □

14.2. **Attribution.**

Lemma 14.4 (Cover)**.** *Let P_1 be a backward path contained in colony $\mathcal{C}$, starting at time t_0, and passing through the nonempty time interval $I = [v_1, t_0 - Q\tau_1]$. There is a union A_1 of three intervals of size* (split_t) *such that we have one of the following cases, with $c_0 = (P_1(t_0), t_0)$.*

(1) *At all times v_0 in $I \setminus A_1$, the cell $P_1(v_0)$ is in a domain that has no exposed edges in $\mathcal{C}$.*

(2) *There is a time v_0 in $I \smallsetminus A_1$ at which P_1 can be redirected using at most $2Q$ links, to reach the originating colony of c_0;*

Proof. Let A_0 be the set of elements t of I such that a damage rectangle intersects $\mathcal{C}$ during $[t - (\text{split\_t}) + T_\bullet, t + (\text{split\_t})]$. Then A_0 is covered by an interval of size $2(\text{split\_t})$. At time v_0 in $I \smallsetminus A_0$, assume the domain of $P_1(v_0)$ has exposed edges in $\mathcal{C}$. If the colony is not an originating colony we can choose the edge to look towards the originating colony. Indeed, if there is no exposed edge pointing in this direction then we can redirect the path by a horizontal stretch to the originating colony, as in case (2) of the lemma.

Without loss of generality, assume that our edge is a left edge y_0. The conditions of the Lemma 14.2 (Bad Gap Inference) are satisfied for (y_0, v_0) as the cell (x_0, t_0) in that lemma. Case (1) of the conclusion of the lemma does not happen since it would satisfy the conditions of the Running Gap Corollary for (y_0, v_0): hence $t_0 - v_0 \leq Q\tau_1$ would follow, contrary to the definition of I. Case (2) implies case (2) of the present lemma and it is also easy to see that then c_0 is not a member cell.

As in the proof of Lemma 14.3 (Small Progress), we can show that case (3) of the Bad Gap Inference Lemma does not occur. Consider case (3) of that lemma. As in the proof of the Small Progress Lemma, we can conclude that cases other than (4) and (5) of Lemma 12.15 (Exposing) do not occur. These remaining cases result again, in at most one additional time interval J f size (split_t) to be excluded (not two, since the event in question can occur at only one end of the colony $\mathcal{C}$). Thus, let $A_1 = A_0 \cup J$. □

If the Cover Lemma is applied with $t_0 - v_1 \geq 2Q\tau_1$ then the set $I \smallsetminus A_1$ is not empty: indeed, see (13.4).

Remark 14.5. In case c_0 is a germ cell with $Age < (\text{germ\_end}) - 2Q$, further I is defined as $I = [v_1, t_0 - 3Q\tau_1]$ and then we can conclude similarly that there is no weak left exposed edge even outside $\mathcal{C}$. Indeed, then the domain in question is of size $\leq 5Q$ and the Lemma 13.5 (Running Gap) would erase it in $5Q\tau_1$ hours. ♢

Lemma 14.6 (Cover Ancestor)**.** *Suppose that $\mathcal{C} \times [v_0, v_1]$ is not intersected by any damage rectangle, and at time v_1, colony $\mathcal{C}$ is covered by a domain with no exposed edges, consisting of member cells or of internal germ cells older than* (germ_grow_end)*. Then, defining A_0 as in the proof of Lemma 14.4 (Cover), at all times u in $[v_0, v_1] \smallsetminus A_0$, the colony is covered by a domain that has no exposed edges.*

Proof. We can follow the proof of the Cover Lemma for the special case considered here: that the domain is only over $\mathcal{C}$, consisting of member cells or of internal germ cells. It is easy to verify that applicable cases of the Bad Gap Inference Lemma remain that lead to a widening gap. □

We will say that cell (x_0, t_0) is *attributed* to colony $\mathcal{C}$ if there is a path P_1 going back to time $t_0 - 5Q\tau_1$ and a union E of at most 3 intervals of length (split_t) such that $P_1(t) \in \mathcal{C}$ for t in

$$I = [t_0 - 4Q\tau_1, t_0 - 3Q\tau_1], \tag{14.1}$$

and $\mathcal{C}$ is covered by a domain without exposed edges for all times in $I \smallsetminus E$. In view of (13.4), this is the majority of times in I and therefore if (x_0, t_0) is attributed to colony $\mathcal{C}_0$ and (x_1, t_0) is attributed to colony $\mathcal{C}_1$ then $\mathcal{C}_0$ and $\mathcal{C}_1$ either are disjoint or are the same.

Lemma 14.7 (Attribution)**.** *Assume that the live cell $c_0 = (x_0, t_0)$ with colony $\mathcal{C}$ is not a germ and is not in the wake of a damage rectangle. Then we have:*

(a) *It c_0 is a member cell then it can be attributed to its own colony.*
(b) *If c_0 is an outer cell then it can be attributed to its originating colony.*

(c) *If c_0 is an outer cell older than* (grow_end) $+ 2Q\lambda p_1$ *and not adjacent to the originating colony, then it can be attributed to its own colony.*

Proof. Without loss of generality, assume that c_0 is not to the left of the center of its originating colony.
Let us build the path $P_1 = (c_0, c_1, \dots)$ with $c_i = (x_i, t_i)$ backward in such a way that vertical, parental and horizontal links are chosen in this order of preference. Horizontal links are chosen only in the case of the application of Lemma 13.3 (Ancestor). Whenever we have a choice between father and mother we choose the parent towards the center of the originating colony of c_0 or towards the center of the current colony, as the construction requries.

1. Assume that c_0 is not an outer cell younger than (grow_end) $+ 2Q\lambda p_1$ and one of the following holds:

   (1) P_1 does not leave $\mathcal{C}$ during $[t_0 - 2Q\tau_1, t_0]$;
   (2) c_0 is not adjacent to its originating colony;
   (3) $[t_0 - (\text{split\_t}), t_0]$ is damage-free;

   Then it can be attributed to its own colony.

   Proof. Let us direct the path always towards the center of the current colony. If the path did not leave $\mathcal{C}$ during $[t_0 - 2Q\tau_1, t_0]$ one finds, applying the Cover Lemma, some time v_0 during this period when the colony is covered by a single domain. At time v_0, we can extend the path along horizontal links to the center of the colony $\mathcal{C}$ of c_0. Continuing backward from there by the original method, the path will never reach the edges of the colony, since the only way for it to move away from the center is by one horizontal link near a damage rectangle, and this can only happen once. Hence an application of the Cover Lemma finishes the proof.

   If c_0 is not adjacent to its originating colony then the backward path constructed by our method never leaves $\mathcal{C}$ during $[t_0 - 2Q\tau_1, t_0]$. Indeed, there is at most one horizontal link in the path during this time, and the parental links do not move the path away from the center.

   If neither of the above conditions holds but $[t_0 - (\text{split\_t}), t_0]$ is damage-free then c_0 is an endcell of $\mathcal{C}$ during this time. It must have a sibling in $\mathcal{C}$ sometime during this interval since otherwise it would be killed by *Decay*. We can redirect the path into the sibling and continue from there.
2. If c_0 is an outer cell then P_1 can be redirected to the originating colony during $[t_0 - 2Q\tau_1, t_0]$ and then continued there without leaving it again, as above.
   Proof. Without loss of generality, suppose that c_0 is a right outer cell. The Cover Lemma implies that for $J = [t_0 - 2Q\tau_1, t_0 - Q\tau_1]$, at all times v_0 in $J \setminus A_1$ with A_1 as defined there, the cell $P_1(v_0)$ is in a domain that has no exposed edges in $\mathcal{C}$. At any such time v_0 the leftmost cell of $\mathcal{C}$ belongs to the domain since otherwise the left edge of the domain would be exposed. If it has a sibling in $\mathcal{C} - QB$ then let us then direct P_1 to this sibling. According to 1, P_1 can be continued from here without having to leave $\mathcal{C} - QB$ again.

   If the left end has no left sibling then it must be already a member cell in a new work period. Lemma 14.2 shows that during $J \setminus A_1$ there are no cells to the right end of the domain that could prevent the increase of age as in Condition 11.4 (Address and Age). Therefore according to Lemma 14.3 (Small Progress), the minimum age of cells in the domain keeps decreasing along the backward path. But when it decreases by more than Q the left edge of the colony is also an outer cell and has a left sibling in $\mathcal{C} - QB$ to which the path can be redirected.
3. Assume that c_0 is a member cell, the interval $[t_0 - (\text{split\_time}), t_0]$ is not damage-free and P_1 leaves $\mathcal{C}$ during this time. Then c_0 can still be attributed to its own colony.

Proof. It follows from the cases already considered in 1, that we can assume that the path leaves $\mathcal{C}$ on the left and c_0 is the left endcell of $\mathcal{C}$, that before leaving $\mathcal{C}$ the path is vertical and the cells on it do not have any right sibling. The path leaves $\mathcal{C}$ along a horizontal link (c_{p-1}, c_p). Indeed, suppose it is a parental link. The rule animating x_{p-1} could not be the growth rule since c_0 is a member cell and due to (13.10), the path is not long enough to get the age from the expansion period to the end of the work period. The healing rule does not act across colony boundaries. The horizontal link across the colony boundary is created as in the proof of the Lemma 13.3 (Ancestor), therefore the extended damage rectangle is near the colony boundary, near time t_p.

If c_p is an outer cell then 2 implies that the path can be redirected within the desired time to $\mathcal{C}$. Let us show that c_p is not member cell. It could be one only if P_1 crosses into $\mathcal{C} - QB$ at an age when $\mathcal{C}$ is a newly created colony covered with outer cells and c_p is in the originating colony. Let $i < p$ be such that c_{i-1} is still a member cell but c_i is already an outer cell. Then the transition from c_i to c_{i-1} involves a transition from outer to member status at the end of the work period. This cannnot happen, however, under our assumption that c_i has no right sibling.

□

14.3. **Progress.** The lemma below says that under certain conditions, the age of a colony increases.

Lemma 14.8 (Large Progress)**.** *Let $c_1 = (x_1, v_1)$ be of colony $\mathcal{C}$ with starting cell z_0, within B of the center of $\mathcal{C}$ and not in the wake of any damage rectangle. Assume that either c_1 is a member or it is an internal germ cell with Age $>$ (germ_grow_end) $+ 10Q\lambda p_1$. There is a v_0 and a forward path P_1 in $\mathcal{C} \times [v_0, v_1]$ such that defining*

$$x_i = P_1(v_i),\ c_i = (x_i, v_i), \tag{14.2}$$

$$\Delta = Age(c_1) - Age(c_0), \tag{14.3}$$

we have $v_1 - v_0 \le 5Q\tau_1$, further either $Q \le \Delta$ or the path crosses a work period boundary. Cell c_0 is within distance B of the center of $\mathcal{C}$, the latter is not in the wake of any damage rectangle, and $\mathcal{C}$ is covered by a domain at time v_0.

Proof. The proof is a repeated application of Lemma 14.3 (Small Progress), with some case distinctions and delays due to damage. Let us build the path P_1 backward like in the proof of Lemma 14.7 (Attribution), till

$$v_0 = v_1 - 3Q\tau_1.$$

This path does not leave $\mathcal{C}$. Indeed, it could only leave if there was a number of steps in which the path moves towards the edge of the colony. Each such step but one must be a parental link. If the animation was via healing then the healing would be an internal correction and the parent towards the center would be chosen, so this is excluded in all cases but one. The animation cannot be via growth since due to (13.10), the age of cells along the path P_1 cannot decrease enough for them to become growth cells.

Let $E(t)$ be the maximal domain containing $P_1(t)$, and let $m(t)$ be the "local" minimum value of age in $E(t)$ (taking into account the possible crossing of work period boundary). Let $I = [v_0, v_1 - Q\tau_1]$. According to the Cover Lemma, there is a set A_1 coverable by 3 intervals of size (split_t) such that for all t in $I \setminus A_1$ the colony $\mathcal{C}$ is covered by $E(t)$. If $v_0 \in A_1$ then let us change v_0 to a little larger value, so that $v_0 \notin A_1$. Then P_1 can be led back to the center at time v_0. It remains to show $\Delta > 2Q$.

Let us represent $I \setminus A_1$ as a union of at most 4 disjoint closed intervals $[s_i, f_i]$ $(i = 1, 2, \dots)$. In each interval $[s_i, f_i]$, according to Lemma 14.3 (Small Progress), $m(t)$ keeps increasing to

$$(t - s_i - 2(\text{split\_t}))/\tau_1 - 3$$

until an exposed edge (say, a right edge) occurs at some time t_1. By the same lemma, for $t \in [t_1, f_i]$, this edge moves at least $(t - t_1)/\tau_1 - 1$ cell widths till time t towards $\mathcal{C}$ unless it disappears earlier by reaching $\mathcal{C}$. The damage can stop this process and move right the end of $E(t)$ by at most 2 cells. Therefore except for a union J_1 of two time intervals of a total length of $(Q+2)\tau_1$ accounting for right exposed edges and another exception set J_2 accounting for left ones, part 1 of the Small Progress Lemma is applicable.

The set $I \setminus (A_1 \cup J_1 \cup J_2)$ breaks up into a disjoint union of at most 6 intervals $[s_i', f_i']$. The domain $E(t)$ covers $\mathcal{C}$ and has no exposed edges during these intervals, so $m(t)$ increases in them as in the Small Progress Lemma, and the total increase during I is at least

$$(v_1 - v_0 - (3Q+4)\tau_1 - 7(\text{split\_t}))/\tau_1 - 14 > Q. \tag{14.4}$$

□

15. Healing

15.1. **Healing a gap.** Given a set E of cells at time t, let us call a cell x' a *successor* of E at time $t' > t$ if (x', t') is reachable by a forward path from some (x, t) for some x in E.

Lemma 15.1 (Healing)**.** *Let $[a_0, a_1] \times [u_0+, u_1-]$ be an extended damage rectangle. Let E be a domain at time u_0 with the property that for each locally maintained field F such that the age of no cell of $\mathcal{C}$ is in an update interval for F (let us call these fields "relevant"), the latter is constant over $\mathcal{C}$ at this time. Let $\mathcal{C} = \{ z_0 + iB : i = 0, \dots, Q-1 \}$ be a colony with which the addresses in E are aligned, with $a_0 \in [z_0, z_0 + QB/2-]$. Then the following holds.*

(a) *At time $w_0 = u_0 + 4\tau_1$, some multidomain in $[a_0 - QB/2, a_0 + QB/2-]$ contains all successors of $E \cap [a_0 - QB/4, a_0 + QB/4-]$ from time u_0;*

(b) *If E covers $\mathcal{C}$ at time u_0 and all cells of $\mathcal{C}$ are member cells then a domain covers $\mathcal{C}$ at time w_0 and each relevant locally maintained field is constant over $\mathcal{C}$ at this time;*

Proof.

1. Let us prove (a).

   The damage can change only a single cell x of E. If this cell is at an end of E then there is nothing to prove. Otherwise, $x - B$ and $x + B$ are also in E. Assume first that both $x - B$ and $x + B$ survive until w_0. Then we have a case of internal correction; let us show that this correction succeeds. If an internal correction of $x - B$ is also possible before internal correction starts in x (it cannot become possible after the correction of x started because either x was dead or the correction of x starts with killing x) then the reasoning for x can be applied to $x - B$. Suppose therefore that internal correction of $x - B$ is not possible. The correction begins by killing x (in case it is not mild). Let us show that it succeeds by reanimating it. If $x - B$ is not a left edge any time before w_0 then the internal correction of x will succeed since the animation, needing a neighbor that is not isolated, can use $x - B$. Similarly if $x + B$ is not a right edge any time before w_0.

   Let us show that if each of these cells is an edge pointing away from x then one of them dies before w_0 contrary to our assumption. Certainly, one of them is an exposed edge, suppose that e.g. $x - B$ is a left exposed edge at some time t in $[u_0, w_0 - \tau_1]$. Lemma 14.2 (Bad Gap Inference) implies that either there is a bad gap on the left of $x - B$ or there is a backward path of length $\le 7(p_1 + 1)$ leading from $(x - B, t)$ to a cell in the same colony that changes, at age (grow_end), from a protected to an exposed left edge, or undergoes a planned kill. If there is a bad gap of the indicated size then $x - B$ is an isolated cell during $[t, w_0]$. It is easy to see that it is not an endcell of a near-end correction and therefore it will be purged by the time w_0, contrary to the

assumption that it stays alive. Suppose now that a backward path of length $\leq 7(p_1+1)$ leads from $(x-B,t)$ to a cell in the same colony that changes, at age (grow_end), from a protected to an exposed left edge. Then it is easy to see that $x+B$ (which also is a right edge at some time before w_0) is an exposed cell for which the Bad Gap Inference Lemma really infers a bad gap on its right, resulting in its purge.

Suppose now that e.g. $x-B$ dies before w_0. If there will be no cells after w_0 that can be traced back to E only bey tracing them through the left or x, then we are done. If there are at any time $t > u_0$, then they must be connected by parental and horizontal links to $x-B$, therefore they form a domain from $x-B$ left. This domain can only be killed from the edges. $x-B$ cannot be the last element to be killed since then the whole domain disappears by w_0. Therefore there must be moment t' when $x-B$ and $x-2B$ are alive and $x-B$ will be killed. The only rule accomplishing this is *Decay*, if $x-B$ is a right exposed edge and the gap in x cannot be healed. Since $x+B$ is still there this could only happen if no internal correction is possible in x, since otherwise this internal correction would have succeeded already before the killing. The only obstacle to the internal correction is if $x+B$ has no right sibling and is not a right protected endcell.

It is easy to see using the Lemma 14.2 (Bad Gap Inference) that there is no domain on the right of E with which the successor of E could merge in the given time period. Also healing the gap caused by the death of $x-B$ is not possible since (it was created due to the impossibility of healing to begin with). Therefore the the whole domain containing $x+B$ (and possibly containing x and even $x+2B$ if the latter arose by growth in the meantime) decays within further $3\tau_1$ hours.

2. Let us prove (b) now.
Suppose that E covers $\mathcal{C}$ at time u_0 and all cells of $\mathcal{C}$ are member cells. Let x be the cell changed by the damage. If it is not an endcell or next to an endcell of $\mathcal{C}$ then (given the absence of planned kill) the change caused by the damage will clearly be corrected by the rule *Heal*.

Suppose that $x-B$ is a left endcell. Then the only obstacle to healing x is if there is a competing internal correction in $x-B$ that would create a member cell in $x-B$. This cell could only belong to a colony $\mathcal{C}'$ different from $\mathcal{C}$ since $x-B$ is an endcell in $\mathcal{C}$. Since there is a correction in $x-B$ the cell $x-2B$ must belong to $\mathcal{C}'$. It is clearly exposed to the right at the observation time of the correction (since the correction would create a member cell, a protected edge is excluded). It must have been there as an exposed cell for a significant time. Indeed, according to Lemma 14.6 (Cover Ancestor), $\mathcal{C}$ has been full for a long time, the growth could not have created $x-2B$ during quite a long interval before u_0, and healing could not have created it while $x-B$ was an endcell of $\mathcal{C}$. But it could not have been there long since the Decay rule or the Purge rule would have killed it.

Suppose now that x is a left endcell. Then it is part of an end-correction. According to the healing rule, one possible obstacle to carrying out the end-correction is that there is an internal correction in $x+B$. In this case, x is the endcell of a near-end correction, therefore it will not be purged, and the near-end correction will be carried out. All other possible obstacles listed in the healing rule imply that $x-B$ has been right exposed cell for such a long time before u_0 that it would have died.

If the healing rule tries to correct x it still must be proved that it succeeds in doing so. If x is not latent then first x will be made latent, then line (2) of the rule *Heal* in Subsection 12.5 applies to cell $x+B$ to make a member cell in x by setting $Kind_{-1}.Heal := Member$ and triggering thereby rule *Animate* of Subsection 12.4. We must show that the animation of the latent x succeeds. The only obstacle to the animation would be an end-healing of member cells of some other colony $\mathcal{C}'$

from the left. But this cannot happen since then the rightmost live cell of $\mathcal{C}'$ must have been exposed before the damage, and Lemma 14.2 (Bad Gap Inference) would imply that it has a large gap on its right, excluding right end-healing.

If x is vacant then first it must be made latent by rule *Create* of Subsection 12.3: for this, Lemma 12.8 (Creation) can be used. Again, no obstacle will be posed by any opposite end-healing.

Adding the time upper bound in the Creation Lemma to τ_0, we get the upper bound $4\tau_1$ for the healing time. Due to to (13.1), this is still smaller than the time needed for the decay rule to kill any cell made exposed by the damage.

□

16. Computation and legalization

16.1. **Coding and decoding.**

16.1.1. *Main track and redundancy track.* Assume that the transition function Tr_k and has bandwidth $w = w_k$ as defined in 2.2.1, and similary the simulated transition function Tr_{k+1} has bandwidth w_{k+1}.

The state of each small cell is partitioned using $Buf = Inbuf \cup Outbuf$, etc. Also, the field *Info* is partitioned into subfields *Info.Main* and *Info.Redun*. Each of the fields of a small cell mentioned in the construction of the previous sections, with the exception of *Info.Main*, will be part of *Buf*. Let

$$P = Cap_{k+1}/w_{k+1}.$$

Assume w.l.o.g. that this number is integer. We partition the state of the represented cell into P fields called *packets* of size w_{k+1}. We also break up the field *Info.Main* into fields $Info.Main_i$, $i = 0, \dots, P-1$ of equal size. Assume without loss of generality that

$$\frac{Cap_k}{w_k} \le \frac{Cap_{k+1}}{w_{k+1}} \le 2\frac{Cap_k}{w_k}. \tag{16.1}$$

The first inequality implies that the width of track $Info.Main_i$ is less than the bandwidth of a small cell. The second inequality will be used in deriving (16.7). (In case the first inequality was violated, we could distribute a packet over several tracks whose width is the bandwidth, and otherwise the algorithm would be similar. In case the second inequality was violated, we could put several packets onto each tracks whose width is the bandwidth, and otherwise the algorithm would be similar.)

Let us also break up the track *Info.Redun* into locations (segments) $\_Info.Redun_i$, $i = 0, 1, \dots, P-1$. Our error-correcting code will distinguish, similarly to Example 4.8, information symbols and error-check symbols. Each packet of the represented state will be encoded separately. Track $Info.Main_i$ will contain the information symbols, and location $\_Info.Redun_i$ contains the error-check bits of packet i. Let the location

$$\_Info_i$$

denote the union of track $Info.Main_i$ and location $\_Info.Redun_i$. These together form the error-correcting code of packet i. The first three packets represent $Inbuf^*$, $Outbuf^*$ and $Pointer^*$.

We will work with a 12-error-correcting code

$$\alpha \tag{16.2}$$

as in Example 4.6, with Q_k information symbols. Since that code can correct errors in t symbols with just $2t$ error-check symbols, so for each i, the location $\_Info.Redun_i$ is only 24 cells long. When *loc* is a location representing a string s then we will use the notation

$$\alpha^*(loc) = \alpha^*(s)$$

for the value decoded from it.

The computation result will be stored temporarily on the track *Hold*. This has, similarly to *Info*, also subfields *Main* and *Redun*. This computation result corresponds to the output of $Tr^{w_{k+1}}$. Recall that the latter consists of 5 segments of size w_{k+1}. The first three segments determine the new values of $Inbuf^*$, $Outbuf^*$ and $Pointer^*$. The fourth segment determines the new value of a part of the cell state whose position is determined by the fifth segment. The output of the simulating computation, on the *Hold* track, will correspond to the first four of these values, in four subfields. The fifth one will be kept in a locally maintained field called

$$Target\_addr$$

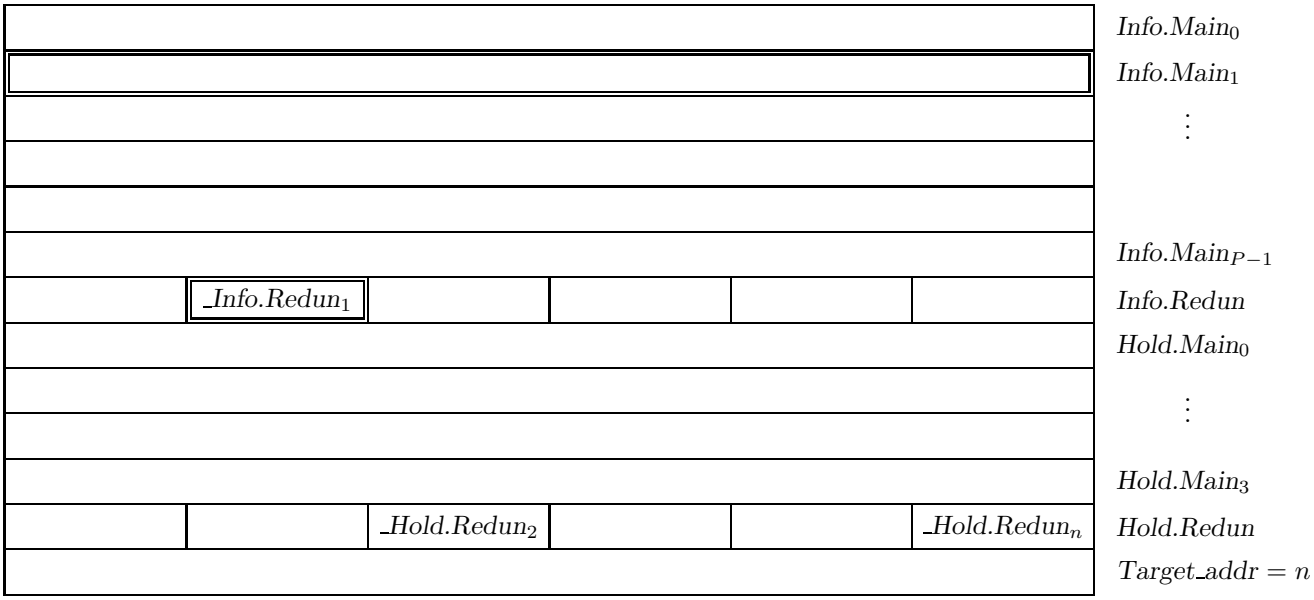

FIGURE 17. The *Info* and *Hold* tracks] subdivided into packets, with $P = 6$ and $n = 5$. The location $\_Info_1$ is the union of the framed locations $Info.Main_1$ and $\_Info.Redun_1$.

which is supposed to have a constant value throughout the colony. Track *Hold.Redun* is split into locations $\_Hold.Redun_i$ for $i = 0, \dots, P-1$, most of which will be unused. For $i = 0, 1, 2$ we define again $\_Hold_i$ as the union of the track $Hold.Main_i$ and $\_Hold.Redun_i$, similarly to how $\_Info_i$ was defined. The redundancy bits for $Hold.Main_3$ will be put, however, not into $\_Hold.Redun_3$ but into $\_Hold.Redun_n$ where $n = Target\_addr$ (here, n has the same meaning as in 2.2.1). Correspondingly, we define

$$\_Hold_3 = Hold.Main_3 \cup \_Hold.Redun_n.$$

16.1.2. *The code of the amplifier.* Recall (13.13). If $\varphi^*(\eta)(y,t) \notin Bad$ then let us define

$$\eta^*(y,t) = \varphi^*(y,t) = Vac$$

unless there is a time v_1 in $[t - (\text{crit}), t]$ such that the colony $\mathcal{C}(y)$ is covered with member cells belonging to the same work period at time v_1 and is not intersected by the extended damage rectangle at this time. In case there is such a time v_1 let t^* be the last time before t with this property that is also the switching time of a small cell (just to make sure there is such a last time). Let

$$s = \eta(\mathcal{C}(y), t^*).Info$$

(the meaning of this notation should be clear). If s contains at most 2 errors then (see (16.2))

$$\eta^*(y,t) = \alpha^*(s),$$

else it is *Vac*. The look-back to an earlier time in the definition avoids some oscillation between the different states assigned to the large cell that could occur as the result of damage and asynchronous updating.

The encoding φ_* takes the state of a cell, applies α_* and copies the result to *Info* track of a colony. The other parts of the colony are set in such a way as they are to be found at the beginning of a work period. The encoding φ_{**} is like φ_* except that all tracks but *Info* will be filled with the don't care symbol $*$.

16.1.3. *Coding and decoding rules.* Let

$$Vacant\_str$$

be a string representing a vacant state in our code. Let

$$Code\_size(k) = \lceil R_0/R_1(k) \rceil \tag{16.3}$$

where R_0 and $R_1(k)$ were defined in 9.1.2 and 9.1.3. Let *Decode_prog* be a program for the medium *Univ* such that location *_Decode_output* on track *Cpt.Output* after applying

$$Univ(Decode\_prog \sqcup S; Q, 2Q)$$

contains $\alpha^*(S)$ (see (16.2)) if $\alpha_*(\alpha^*(S))$ differs from S in at most 12 symbols: otherwise, it contains *Vacant_str*. The decoding rule takes a string from location loc_1, decodes it and copies the result to location loc_2. The rule *Write* was described in Subsection 7.4.

```
rule Decode(loc_1, loc_2) {
    Cpt.Input := *;
    Write(Decode_prog, _Prog)
    Copy(0, loc_1, _Decode_arg);
    apply the rule Univ 2Q times to Cpt
    Copy(0, _Decode_output, loc_2);
    }
```

(Recall the assumption made in 7.2.1 that *Univ* is commutative, so we do not have to worry about the order of execution in different cells.) The rule

$$Encode(loc_1, loc_2)$$

performs encoding. The locations loc_i in the encoding and decoding rules will be allowed to be given indirectly, as explained in Remark 7.10.

16.2. **Refreshing.** In each rule of the present section, the operation

$$\operatorname*{Maj}_{i=0}^{2} s_j$$

when applied to strings s_0, s_1, s_2 is the result of taking the bitwise majority of these three strings.

We will have a general type of rule

$$Check_0(prop, F(I), X_1, X_2, \dots)$$

checking some global property of some parameters $X_1, X_2, \dots$ each of which is either an explicit string or a location. Here, *prop* is some program for the universal computing medium such that after we run the rule of that medium for Q steps the bit b representing the outcome of the check will be broadcast into field F of every cell of interval I. We will also need an error-checked version of $Check_0$ which uses some fields $Vote_i$ for $i = 0, 2, 3$:

```
   rule Check_1(prop, F(I), X_1, X_2, ...) {
      for i ∈ {0,1,2} do {
(1)       Check_0(prop, Vote_i(I), X_1, X_2, ...)   }
      cond {
         ?   Addr ∈ I
```

```
        !   F := Maj^2_{i=0} Vote_i
        }}
```

The rule

$$Check_0(nearly\_equal, F(I), loc_1, loc_2, d)$$

checks whether the strings in locations loc_1 and loc_2 differ in at most d code symbols. The subrule *Refresh* attempts to clean the information on the *Info* and *Hold* tracks by decoding and re-encoding it. It also attempts to make sure that these tracks represent at least something, even if this is only the vacant state.

Rule *Update_loc_maint* updates the locally maintained fields

$$Doomed, Growing_j \ (j \in \{-1, 1\}).$$

(It need not update the fields $Control_{0.1j}$ to be introduced later.) Rule

$$Check_0(recheck\_1, F(I), G, d) \tag{16.4}$$

checks whether track G is covered with 1's with possibly d cells as exceptions. As usual, the argument F means $F([0, Q-1])$. The rule uses some additional locations and fields: $\_Decoded_0$ $\_Encoded_n$ for $n = 0, 1, 2$, Ck_res_n for $n = 0, 1, 2$.

```
    subrule Refresh {
        Ck_res := 1;
        for i = 0 to P − 1 do {
            for n = 0 to 2 do {
                Decode(_Info_i, _Decoded_0);
                Encode(_Decoded_0, _Encoded_n);
                Check_0(nearly_equal, Ck_res_n, _Encoded_n, _Info_i, 8);
                Idle Q steps   }
            _Encoded := Maj^2_{n=0} _Encoded_n;
            Ck_res := Ck_res ∧ Maj^2_{n=0} Ck_res_n;
            Copy(_Encoded, _Info_i);
            Idle Q steps   }
(1)     Check_1(recheck_1, Ck_res, Ck_res, 2);
        cond {
            ?   Ck_res = 0
            !   Write(Vacant_str, _Info)   }
        Repeat the whole procedure with _Hold in place of _Info, "mutatus mutandis".
        }
```

Let us say that a string s of symbols that can be the argument of a certain error-correcting decoding *has d errors* if, when applying the decoding and encoding to it, the result differs from s in d symbols.

Lemma 16.1 (Refresh)**.** *Assume that we have a full colony in which all cells have ages before the starting point of Refresh. Then we have:*

(a) *At the end of Refresh, for all i, locations $\_Info_i$, have at most 4 errors;*

(b) *Suppose that at the beginning of Refresh, for all i, locations $\_Info_i$, have at most 4 errors. Then during the whole Refresh, the value $\alpha^*(\_Info_i)$ does not change and $\_Info_i$ has at most 8 errors.*

The same statements hold for location $\_Hold$.

Proof. We perform the analysis only for *_Info*.

1. For any i, the "immediate" effect of the damage can affect at most 4 symbols of $\_Info_i$.

Proof. The damage rectangle can have immediate effect in the cell where it happens, and its healing can change possibly one more cell next to it. If the affected cells happens to contain the symbols from $\_Redun_i$ then this can affect two information symbols and two error-check symbols.

2. Assume that the damage rectangle occurs before point (1) of the rule.
Let us look at the damage-free operations in $Check_1$ after (1). The first such operation turns *Ck_res* into all 0 or all 1. If $Ck\_res = 0$ then the last step writes *Vacant_str* into *_Info*, and thus the latter will have no errors.

Suppose therefore that if $Ck\_res = 1$. Then before $Check_1$, we have $Ck\_res = 1$ in all but two cells. Let us look at any cell x not affected immediately by damage (the large majority is such). Since the majority step computing *Ck_res* is correct in x, for all i there must have been two values of n for which $Check_0$ wrote the value 1 into $Ck\_res_n(x)$. For at least one of these n, say n_1, the computing period (i, n_1) is damage-free. That period wrote an error-free string into $\_Encoded_{n_1}$, and also determined that this string does not differ from $\_Info_i$ by more than 8 symbols. There is another damage-free period (i, n_2). Its $\_Info_i$ could have been different at its beginning (if damage occurred between the two periods), though according to 1 by not more than 4 symbols. Therefore, since our code corrects 12 errors, we have $\_Encoded_{n_1} = \_Encoded_{n_2}$. Except for the immediate effect of new damage, this is going to be the new value of $\_Info_i$, therefore there can be at most these 4 new errors at the end of *Refresh*.

Suppose that at the beginning, each $\_Info_i$ has at most 4 errors. By 1, the damage rectangle affects at most 4 cells of $\_Info_i$ immediately. Therefore during the whole *Refresh*, the value $\alpha^*(\_Info_i)$ does not change and $\_Info_i$ has at most 8 errors.

3. Assume that the damage rectangle occurs on line (1).
Then the damage did not occur before and therefore the track *Check_res* contains either all 0's or all 1's.

Suppose the damage has occurred before line (1) of rule $Check_1$, in iteration n of that rule. Then only $Vote_n(I)$ will be affected, and possibly two more cells, the cell where the damage occurred and a cell next to it due to healing. In the 2 other iterations, the track *Check_res* still contains either all 0's or all 1's with these exceptions, and the result of these 2 iterations is the same as it was without the damage in the third iteration. The majority vote brings out the same result, except for possibly 2 immediately affected cells.

If the damage occurs after line (1), in rule $Check_1$, i.e. in the last step of the rule, then again only the cells immediately affected will not have the correct result.
If the damage occurs after line (1) in rule *Refresh* then again, this being a one-step action, only the cells immediately affected by the damage would show up as errors. □

16.3. **Computation rules.** After retrieval from neighbor colonies (to be discussed later), the string returned from neighbor m will be in some location

$$\_Retrieved_m.$$

These strings are inputs to the simulated transition function but still in encoded form. After decoding, the result will appear in $\_Decoded_m$

16.3.1. *Evaluation.* As seen in (2.6), only the field Buf^* of the neighbors is found in $\_Retrieved_m$ for $m \neq 0$. For $m = 0$, the location $\_Info_a$ is also needed, where a is determined by $Pointer^*$, which according to 16.1.1 is found in $\_Info_2$. Thus, first we broadcast this number a into a locally maintained field

$$Source\_addr$$

of each cell of the colony, and then use indirect copying as in Remark 7.10 to copy $\_Info_a$ into the appropriate part of $\_Retrieved_0$.

Then, the subrule *Eval* finds the intended output of $Tr^{(w)}_{k+1}$, as defined in 2.2.1, using the universal computation, as in *Decode* above. Here are the details.

We are proving Lemma 9.4 (Amplifier), so what is given is an amplifier frame *Frame*, and the number k, and we are building the transition function of the medium M_k. Assume that *Frame* is described in a string

$$Param_2 = Frame,$$

the parameter k is described in a string

$$Param_1 = k,$$

and the program we are just writing is $Param_0$. First *Eval* computes $Tr^{(w)}_{k+1}$. It uses several new locations, some of which were introduced in 7.4.3: $\_Interpr$, $\_Param_i$ for $Param_i$, $\_Arg_m$. The result will be deposited in $\_Sim\_output_i$ for $i = 1, 2$.

$Cpt.Input := *$; $Write(Interpr, \_Interpr)$;
$Write(Param_0, \_Param_0)$;
$Write(Param_1 + 1, \_Param_1)$;
$Write(Param_2, \_Param_2)$;
`for` $m \in Nb\_ind$ `do` {
  $Copy(0, \_Decoded_m, \_Arg_m)$ }
apply Q times the rule *Univ* to *Cpt*
The result is in $\_Sim\_output_1$.
$Copy(0, \_Sim\_output_1, \_Sim\_output_2)$

As seen in (2.8), in the output string of $Tr^{(w)}_{k+1}$, only the first segment of size w_{k+1} (corresponding to field $Inbuf^*$) depends on the neighbors. To make sure that really this is the case, the above sequence of operations will in fact be performed twice, with the following differences.

- The first time, for all $m \neq 0$, the string written in $\_Arg_m$ will not be a copy of $\_Retrieved_m$ but simply the corresponding part of *Vacant_str*. Thus, the result does not depend on retrieval from neighbor colonies;
- The second time, only field $Inbuf^*$ of $\_Sim\_output_1$ will be copied into the appropriate part of $\_Sim\_output_2$;

According to 9.1.1, the complete transition function is combined from *Sim_trans* and *Rd_trans*. To make sure this is the case, we recompute Rd_trans_{k+1} and perform the combination. Thus, now the rule *Eval* finds out from $\_Sim\_output_2$ whether the new big cell is going to be vacant. If not then Rd_trans_{k+1} is computed somewhat similarly to Sim_trans_{k+1} above. The main difference is in what we copy into $Param_0$. According to the complexity upper bounds in Subsection 9.1, the sequence Rd_trans_i of functions is uniformly simple. Therefore *Frame* contains a program *Rd_trans_prog* computing Rd_trans_{k+1} on the universal medium *Univ* from $k+1$ and its arguments in time $R_0\|\mathbb{S}_{k+1}\|$ and space $R_0\|\mathbb{S}_{k+1}\|$. We will therefore copy *Rd_trans_prog* into $\_Param_0$. The result will be stored in $\_Rd\_output_2$.

Finally, we combine $\_Sim\_output_2$ and $\_Rd\_output_2$ as required in 9.1.1, into a location $\_Eval\_output$. In accordance with (2.7), this location can be broken up into segments $\_Eval\_output_i$ for $i = 0, 1, 2, 3, 4$, of size w_{k+1}. Locations $\_Eval\_output_i$ for $i = 0, 1, 2$ are supposed to be the first three packets of the new big cell state. $\_Eval\_output_3$ is also supposed to be the new value, to replace some packet $Info.Main_n$ of the new cell state, where $n = \_Eval\_output_4$. To use this information, the number n will be broadcast into a field *Target_addr* of each cell.

Remark 16.2. It seems wasteful to compute Rd_trans_{k+1} twice: once as part of Tr_{k+1} and once separately. To avoid this, in the first computation we can replace *Rd_trans_prog* with a trivial program that does not do anything. ♢

This ends the definition of the rule *Eval*.

16.3.2. *The computation rule.* In the rule *Compute*, the evaluation process will be repeated 3 times: in repetition n, we code *_Eval_output* onto track $Hold\_vote_n$, and finally *Hold* will be obtained by majority vote from $Hold\_vote_n$ $(n = 0, 1, 2)$.

In the rule *Randomize*, the *Rand* bit of the first cell of the colony is taken and an α_*-code of it is placed into the appropriate location on the *Hold* track. This is done only once, without any attempt of error-correction.

Rule *Update_loc_maint* will be discussed below.

```
subrule Compute {
    Refresh;
    for n = 0 to 2 do {
        for m ∈ Nb_ind do {
            Decode(_Retrieved_m, _Decoded_m);   }
        Eval;
        for i = 0, 1, 2, 3 do {
            Encode(_Eval_output_i, _Hold_i);
            For i = 3, use indirect copying with Target_addr = _Eval_output_4.      }
        Hold_vote_n = Hold;
        Target_addr_n := Target_addr;   }
    Hold := Maj^2_{n=0} Hold_vote_n;
    Target_addr := Maj^3_{n=1} Target_addr_n;
    Randomize;
    Refresh;
    Update_loc_maint
    }
```

16.4. **Finishing the work period.** The present subsection is devoted to achieving good upper and lower bounds on the length of the colony work period, i.e. on ${T^\bullet}_{k+1}$ and ${T_\bullet}_{k+1}$ in terms of ${T^\bullet}_k$ and ${T_\bullet}_k$. Though it seems likely that these bounds can be proven if each cell of the colony simply runs until the same maximum age $U_k - 1$, we did not find such a proof. Therefore we introduce some special rules.

The *March* rule updates the age of a cell only when its neighbors allow it, and this is what makes the time estimates difficult. We will therefore define a different rule, $March_1$, which allows updating a different field, Age_1, independently of any neighbors. The idea is to end the work period when a certain cell reaches $Age_1 = U$, since then the (multiplicative) uncertainty in the length of the colony work period will not be much greater than the uncertainty in the length of the individual cell dwell period. We will use the Age_1 of more than one cell, for robustness, but then we will need to reach a consensus among these about when the end comes. To leave clearly enough time for the consensus computation, and to make sure the work period ends in a coordinated way, what will be determined by consensus is some future value *End* of *Age* when the work period will end.

16.4.1. *Finding the work period end.* We will also use the auxiliary field

$$End_0 \in [0, U - 1].$$

At the beginning of the work period, we start with $Age_1 = 0$, $End = End_0 = U - 1$. (These values are extremely large for End, End_0.) Recall the definition of (end_period) in (13.10). The field End_0 is required to have the property

$$End_0 = U - 1 \text{ or } End_0 \le Age + (\text{end\_period}) \tag{16.5}$$

which will be enforced by a trivial rule. The update rule of Age_1 is trivial.

```
rule March_1 {
   cond {
      ?   Age_1 < U − 1
      !   Age_1 :=_{p_0} Age_1 + p_0
      ?!  cond {
          ?   End_0 = U − 1
          !   End_0 :=_{p_0} (U − 1) ∧ (Age + (end_period))
          }}}
```

Thus, each cell updates its Age_1 independently of all other cells, and the delays are also counted in. When the upper bound $U - 1$ of Age_1 is reached each cell puts its estimate of the end of the work period into End_0. The subrule *Find_end* will last at most

$$(\text{synch\_time}) = 16Q \tag{16.6}$$

steps and will be run every 4(synch_time) steps after $Age >$ (synch_start_lb) (see (13.8)). It uses votes $End_i \in [0, U-1]$ $(i = 1, 2, 3, 4)$ to achieve consensus of the End_0 values and to store it in End. The subrule

$$Check_0(synch\_consensus, End\_legal, End_4)$$

checks whether there is a c such that $End_4(x) = c$ for all but 3 of the cells x in the colony.

```
   subrule Find_end {
      for i ∈ {1, 2, 3} do {
         cond {
(1)         ?   There is a c < U − 1 such that End = c
            for all but 2 cells in the colony
            !   End_i :=_{p_1} c
(2)         ?   All but Q/4 cells have End_0 < U − 1
            !   End_i :=_{p_1}  the maximum of the Q/2 smallest values
            of End_0 in the colony.;   }};
      cond {
         ?   There are two equal values among { End_i : i ∈ {1, 2, 3} }
         !   End_4 :=_{p_1} Maj_{i=1}^{3} End_i   };
      Check_1(synch_consensus, End_legal, End_4)
      cond {
         ?   End_legal = 1
         !   End := End_4
         }}
```

16.4.2. *Finish.* At $Age = End$ for non-germ cells and $Age =$ (germ_end) for germ cells, a final computation takes place in the rule *Finish*. The information from the *Hold* track will be copied into the corresponding locations on the *Info* track. This is done in accordance with (2.7): locations $\_Hold_i$ for $i = 0, 1, 2$ go to $\_Info_i$. On the other hand, location $\_Hold_3$ goes to $\_Info_n$ where $n = Target\_addr$. Still, due to the positioning done earlier, the whole operation involves only copying within each cell and takes therefore only a single step. In particular, $Outbuf^*$ remains unchanged until the last step.

In the same step, addresses of growth cells will be reduced mod Q. Outer cells and inner germ cells turn into members, outer germ cells turn latent. End, End_0, Age, Age_1 become 0.

16.4.3. *Updating the locally maintained fields.* The rule

$$Broadcast(loc, F(I))$$

takes the information found in location *loc* and writes it into the F field of every cell in interval I. The rule $Check_0(Check\_vacant, F(I), loc)$ checks whether the string represented in *loc* is Vac^*. Let $\_Creating_j$ be the known location of the field $Creating_j^*$ of the represented cell inside $\_Decoded_0$ after decoding. The value in this represented field will be broadcast reliably to the track $Growing_j$.

```
rule Update_loc_maint {
   for n = 0 to 2 do {
      Decode(_Hold, _Decoded_0);
      Check_0(Check_vacant, Vote_n, _Decoded_0)   }
   Doomed := Maj^2_{n=0} Vote_n
   for j ∈ {−1, 1} do {
      for n = 0 to 2 do {
         Decode(_Info, _Decoded_0);
         Broadcast(_Creating_j, Vote_n);   }
      Growing_j := Maj^2_{n=0} Vote_n
      }}
```

Remark 16.3. Defining the rules this way wastes many operations since we do not extract all information from its decoded form at once but rather decode repeatedly. But it is easier to follow the analysis in this form. ♢

Lemma 16.4. *Assume that we have a full colony in which all cells have ages before a starting point of Update_loc_maint. Then at the end of this rule we have the same conditions as at the end of Refresh, and in addition the following:*

(a) *For all but at most 2 cells, the locally maintained fields Doomed and $Growing_j$ have constant values in the colony;*
(b) *We have $Doomed = 1$ almost everywhere iff $\_Hold$ represents a vacant state;*
(c) *For $j \in \{-1, 1\}$, for almost all cells in the colony, $Growing_j$ is equal to $Creating_j^*$ of the cell state represented by the colony;*

Proof. The proof is similar to, only simpler than the proof of Lemma 16.1: simpler, since at the starting point of *Update_loc_maint*, we can already assume that the conditions asserted by that lemma hold. □

By choosing Q sufficiently large (R_0 times the bandwidth), the time complexity of each of the *Univ* computations can be bounded by some constant times Q and therefore the total number of steps taken by *Compute* can be estimated as

$$K_0 QP = K_0 Q \frac{Cap_{k+1}}{w_{k+1}} \le 2K_0 Q \frac{Cap_k}{w_k} \tag{16.7}$$

where K_0 is some absolute constant, which is consistent with (13.6).

16.5. **Legality.** Let us prove parts (c) and (d) of Condition 8.8 (Computation Property) for big cells.

Lemma 16.5 (Legality)**.** *Assume that, for some rational number a, Damage* does not intersect $\{x\} \times [a - T_\bullet{}^*/2+, a + 2T^{\bullet *}]$.*

(a) *If σ_1, σ_2 is defined in the interval $[a+, a + 2T^{\bullet *}]$ Then*

$$legal_{k+1}(\eta^*(x, \sigma_2-), \eta^*(x, \sigma_2)) = 1 \tag{16.8}$$

and $T_\bullet{}^* \le \sigma_2 - \sigma_1 \le T^{\bullet *}$.

(b) *If σ_0 is defined then* (16.8) *holds for σ_0 in place of σ_2.*

Proof. The discussion in Subsection 11.1 has shown that $\eta^*(x,t) \notin Damage^*$ iff the damage can be covered by a single damage rectangle in $(x,t)+V'$ where V' is defined in (11.1). The space projection of V' is at least $1.5QB$ and the time projection is at least $T_\bullet{}^*/2$. Thus, our assumption implies that during every time interval of size $T_\bullet{}^*/2$ in $[a-T_\bullet{}^*/2, a+2T^{\bullet *}]$, the interval $[x-0.25QB, x+1.25QB]$ is intersected by at most one damage rectangle.
Consider a time v_2 such that $\eta^*(x, v_2) \neq Vac$. If σ_0 is defined then let $v_2 = \sigma_0$, while if σ_2 is defined then let it be some time very close from below to σ_2.
According to 16.1.2, there is a time v_1 in $[v_2 - (\text{crit}), v_2]$ such that the colony $\mathcal{C}$ is covered with member cells belonging to the same work period at time v_1 and is not intersected by the extended damage rectangle at this time. Lemma 14.8 (Large Progress) says that, by setting $u_1 = v_1$, there is a time $u_0 \in [u_1 - 5Q\tau_1, u_1]$, and a path P_1 in $\mathcal{C} \times [u_0, u_1]$ with $d_i = (P_1(u_i), u_i)$ within distance B of the center of $\mathcal{C}$ such that $\mathcal{C} \times \{u_0\}$ is not in the wake of a damage rectangle, is covered by a domain, and $Q \le Age(d_1) - Age(d_0)$. If d_0 is a member cell let us apply the Large Progress Lemma (backward) repeatedly until either the path crosses a work period boundary, or the age along the path decreases by

$$L = 6(\text{synch\_time}) + (\text{end\_period}).$$

Suppose the cell we arrived at is not a member cell and has $Age \ge End - Q$. Then we perform one more repetition, but stop along the path as soon as the age decreases below $End - Q$. Lemma 14.6 (Cover Ancestor) shows that we can stop at a time when the colony is covered by a domain. Let us now denote by $d_0 = (y_0, u_0)$ the cell at which we arrive this way.
1. Assume that d_0 is an outer cell.
Then we have

$$Age(d_1) - Age(d_0) \le L + Q \tag{16.9}$$

and at time u_0, the whole colony is covered by outer cells. Lemma 14.7 (Attribution) shows that d_0 can be attributed to its own colony and that this colony could not have been occupied by member cells for at least (crit) time units before u_0. Therefore $\eta^*(x, u_0) = Vac$.

In this case we have $v_2 = \sigma_0$. The outer cells at time u_0 encode a latent cell of medium M^*. Therefore eventually, when all outer cells turn into member cells, the new colony will encode a latent cell. Now, a vacant-to-latent transition is legal by Condition 11.3.

The reasoning is analogous in case d_0 is a germ cell. Assume now that d_0 is a member cell in the same work period as c_1: then $Age(d_1) - Age(d_0) \ge L$. Then we have the case when σ_1, σ_2 are defined.

2. The Large Progress Lemma can extend the path repeatedly until we arrive at a cell $c_0 = (x_0, v_0)$ in the same work period, with

$$\begin{aligned} v_0 &> \sigma_1, \\ \mathit{Age}(c_0) &\le 2Q, \\ v_1 - v_0 &\le UT^\bullet + 5\tau_1 L. \end{aligned} \tag{16.10}$$

Proof. One can find a time t_1 in $[u_0, v_1]$ such that denoting $e_1 = (P_1(t_1), t_1)$ we have

$$\mathit{Age}(e_1) \le \mathit{Age}(d_0) + 6(\text{synch\_time}),$$

and there is a complete damage-free execution of *Find_end* during $[\mathit{Age}(d_0), \mathit{Age}(e_1)]$.

2.1. This execution of *Find_end* finds more than $Q/4$ cells in $\mathcal{C}$ with $\mathit{End}_0 = U - 1$.

Assume that this is not true. Then this execution of *Find_end* sets the common value of *End* to a value that was $\le \mathit{End}_0(y, t)$ for one of the cells y at the time t when it examined y. By (16.5), then $\mathit{End} \le \mathit{Age}(y, t) + (\text{end\_period})$. It is easy to see that subsequent executions of *Find_end* do not change this value of *End* (except for the spot of damage, which will be corrected). Therefore the work period will have to end at an age before $\mathit{Age}(e_1) + (\text{end\_period})$ which is not possible given our definition of e_1, since we came back L steps within the work period to $\mathit{Age}(d_0)$ and went forward at most 6(synch_time) steps.

Let us continue applying the Large Progress Lemma from d_1 and extending the path backward from d_0 until a cell $c_0 = (x_0, v_0)$ with $Q < \mathit{Age}(x_0) \le 2Q$ or $v_0 = \sigma_1$, whichever comes first. Look at the cells y which were found by the above-mentioned run of *Find_end* at some time t to have $\mathit{End}_0(y, t) = U - 1$ and hence $\mathit{Age}_1(y, t) < U - 1$. There are so many of them that some of them will be damage-free during the whole work period. Take one such cell y. Rule March_1 keeps increasing $\mathit{Age}_1(y)$ between times v_0 and t, hence $t - v_0 \le UT^\bullet$, and

$$v_1 - t \le v_1 - u_0 \le UT^\bullet + 5\tau_1 L.$$

Let us show that $\mathit{Age}(c_0) > 2Q$ is not possible. Indeed, this would mean $v_0 = \sigma_1$. Then the big cell y would go through a switch of a state in σ_1 but our backward path has never changed a colony work period.

3. Under the assumptions, the switch in σ_2 is legal.

Proof. Let us follow the development of the colony $\mathcal{C}(x)$, starting from the time v_0 found above. Since the big cell is non-vacant during this, location *_Info* encodes a string with at most 2 errors. The rule *Compute*, begins, at age (compute_start), with an application of *Refresh*. As the corresponding lemma shows, at the end of this, location *_Info* will still encode the same string with possibly fewer errors. Thereafter, the *Info* track will not change until the next work period, except for the local changes caused by the damage and its corrections by new applications of *Refresh*.

The following computation can be analyzed similarly to the first part of the proof of Lemma 16.1 (Refresh). According to the definition of rule *Eval*, the value put into the part corresponding to Memory^* of *_Hold* is obtained without using the result of *Retrieve*, so it is certainly a value legally obtainable from *_Info*.

Any subsequent change of *_Info* or *_Hold* caused by a damage rectangle will be corrected by a later *Refresh*, and any new inhomogeneity of *Doomed* will be corrected by *Heal*. If $\mathit{Doomed} = 1$ then the whole colony dies, since the exposed right end is impossible to heal. In the case $\mathit{Doomed} = 0$, the rule *Finish* installs the changes on the track *Info* in one step, leaving again no

place for more than 2 errors. In both cases, the state at time σ_2 will be a legal consequence of the state at time σ_2-.

4. Let us lower-bound now the distance between the two switching times of η^* in x. (The proof of the upper bound is similar but simpler.)
The usual analysis of *Find_end* shows that this rule changes *End* either in all cells (but possibly two) or in none of them (but two). Suppose first that $End = U - 1$ stays that way during the whole work period. Then the end would come only at $Age = U - 1$. Measuring the time for a cell that has never been affected by damage, this would give a lower bound $(U - 2Q)p_1T_\bullet > T_\bullet{}^*$.

Suppose now that *End* becomes smaller than $U - 1$. Consider the first execution of *Find_end* when this happens. Then there must have been some damage-free cell x such that by the time t_1 this execution inspects x it has $Age_1(x, t_1) = U - 1$. Let t_0 be the time when $Age_1(x, t_0) = 0$ at the beginning of the work period: then $t_1 - t_0 \geq UT_\bullet$. From here, the desired lower bound follows easily.

□

The lemma below follows easily from the above proof and from Condition 8.5 (Time Marking).

Lemma 16.6. *Assume that, for some rational number a, Damage* does not intersect $\{x\} \times [a - T_\bullet{}^*/2+, a + 2T^{\bullet *}]$. If η^* has no switching time during $[a+, a + 2T^{\bullet *}]$ then $\eta^*(x, a+) = Vac$.*

The following lemma infers about the present, not only about the past as the Attribution Lemma. For an extended damage rectangle $[a_0, a_1] \times [u_0, u_1]$, we call the rectangle

$$[a_0, a_1] \times [u_0, u_0 + 4\tau_1]$$

its *healing wake*.

Lemma 16.7 (Present Attribution)**.** *Assume that the live cell $c_0 = (x_0, t_0)$ in colony $\mathcal{C}$ with base cell z_0 is not a germ, and is to the left of the center of its originating colony. Assume also that $\mathcal{C}$ at time t_0 does not intersect the healing wake of any damage rectangle. Then one of the following cases holds.*

(1) *c_0 is a member cell, attributed to $\mathcal{C}$ which is covered by a domain of member cells at time t_0. If $Q < Age < End - Q$ then the Info track of this colony has at most 2 errors;*
(2) *c_0 is a member cell from which a path of time projection at most $Q\tau_1$ leads back to to a growth cell in $\mathcal{C}$ and this growth cell can be attributed to $\mathcal{C}+QB$. At time t_0, if $\mathcal{C}+QB$ does not intersect the healing wake of a damage rectangle then $[x_0, z_0 + QB-]$ is covered by a domain;*
(3) *c_0 is an outer cell, attributed to its originating colony. At time t_0, if $\mathcal{C} + QB$ does not intersect the healing wake of a damage rectangle then $[x_0, z_0 + QB-]$ is covered by a domain;*
(4) *c_0 is a member cell and there is in $\mathcal{C}$ a backward path from c_0, with time projection $\leq 2(\text{split\_t}) + (Q + 1)\tau_1$ going back to a domain of doomed member cells covering $\mathcal{C}$;*

Proof.

1. Suppose that c_0 is a member cell and $\mathcal{C}$ was covered by member cells at some time in the set $I \setminus E$ (as defined around (14.1)).
Using the Large Progress lemma repeatedly, we can go back to a time before age (compute_start) in $\mathcal{C}$, just as in the proof of the Legality Lemma above. Then we can follow the development of the colony forward and see that it forms a continuous domain together with its extension. The *Info* track has at most 2 errors after the first application of *Refresh*, as seen by Lemma 16.1 (Refresh).

If the computation results in a nonvacant value for the represented big cell then we have case (1) of the present lemma. The represented field $Creating_j$ of the colony will be broadcast into the field $Growing_j$ of its cells, as shown in Lemma 16.4. The homogeneity of this latter field will be

maintained by *Heal*. Thus, depending on the value of $Creating_j$ of the big cell, growth will take place and the growth forms a continuous domain with the originating colony.

Suppose that the computation results in a vacant value. Then $Growing_j$ will be 0 everywhere but in the healable wake of the damage. Growth cannot start accidentally by a temporary wrong value $Growing_j = 1$ in an endcell since there is enough time to correct this value during the long waiting time of *Grow_step*. Also, all cells become doomed. After $Age = 1$, the doomed right end becomes exposed by definition, and the whole colony decays within $Q\tau_1$ time units. Before that, the colony is full. After that, we have case (4) of the present lemma.

2. If c_0 is an outer cell then we have case (3) of the present lemma.
   Proof. The Attribution Lemma attributes c_0 to the originating colony which is covered by member cells during $I \setminus E$. It forms a continuous domain with its extension, until the age *End*. From that age on, they form a multidomain. This could only change if the originating colony goes through a next work period and kills itself; however, there is not enough time for this due to (13.8).
3. Suppose that c_0 is a member cell but during $I \setminus E$, the colony is never covered by member cells. Then we have case (2) of the present lemma.
   Proof. Let c_1 be a non-member cell in $\mathcal{C}$ during $I \setminus E$. Then it is an outer cell that can be attributed to its originating colony and then the reasoning of 2 above can be repeated.

□

17. Communication

17.1. **Retrieval rules.** Here, the rules for information retrieval from neighbor colonies will be given. Neighbors of a cell are of the form $\vartheta_j(x)$ for $j \in Nb\_ind$ where

$$Nb\_ind = \{-1.5, -1, 0, 1, 1.5\}$$

was defined in Subsection 8.2. Now these same notions of neighbor are applied to big cells and the distance unit is QB rather than B. We define $Mail\_ind$ and $\overline{k}$ as in 7.4.2:

$$Mail\_ind = \{-1.1, -0.1, 0.1, 1.1\},$$
$$\overline{k} = \text{sign}(k)\lfloor |k| \rfloor.$$

First some definitions will be given, for a cell x and its neighbor y in direction j which is equal to $\vartheta_{1.5j}(x)$ if the latter is nonvacant and $\vartheta_j(x)$ otherwise. For a cell x, let

$$x^* = x - Addr(x)B \tag{17.1}$$

be the base of the *originating colony* of x and

$$x^{*0} = x - (Addr(x) \bmod Q)B$$

be the base of the *colony* of x. Thus, we have $x^* = x^{*0}$ iff $Addr(x) \in [0, Q-1]$. The relation of cells and their originating colonies will define the predicate

$$Edge_j(x) = \begin{cases} 0 & \text{if } x^* = y^*, \\ 1 & \text{if } x \text{ and } y \text{ are members of two adjacent colonies,} \\ 1.5 & \text{if } 1 < (x^* - y^*)/(jQB) < 2, \\ \infty & \text{otherwise.} \end{cases}$$

The field $Mail_k$ has the same subfields as in Subsection 7.4. The subfield $Mail_k.Status$ can now also have the value

$$Fickle$$

standing for "transitional". When $Mail_k.Status = Fickle$ then the other subfields of $Mail_k$ will play no role therefore we will, informally, also write $Mail_k = Fickle$. The mail field $Mail_k$ of a cell will cooperate, in direction j, with mail field

$$peer(k, j) = k - j\lfloor Edge_j \rfloor$$

of a neighbor if the latter index is in $Mail\_ind$. To control the track $Mail_k$, there is a one-bit track

$$Control_k \in \{post, get\}$$

which moves in the direction $-\text{sign}(k)$ (opposite to the direction of mail movement which it controls). For $|k| = 0.1$, the values of $Control_k$ will simply depend on age: in certain stages of the program the mail must be posted, in others it must be passed. For $|k| = 1.1$, the value of $Control_k$ will travel in the direction $-\text{sign}(k)$. For robustness, a change in the value of $Control_k$ will propagate into the inside of the next colony only when it is not to be followed by a change back too soon. By definition, for $j \in \{-1, 1\}$, either $\vartheta_j(\mathbf{x})$ or $\vartheta_{1.5j}(\mathbf{x})$ is vacant. For every field F we denote by

$$F^{(j)} \tag{17.2}$$

the field F of the nonvacant cell $\vartheta_{nj}(\mathbf{x})$. Here is the formal propagation of control:

```
rule Propag_control {
   pfor j in {−1, 1} do {
      cond {
         ?  Edge_j > 0 or
(1)            (if ϑ_{2j}(x) is in the colony of x then Control^{2j}_{1.1j} = Control^{j}_{1.1j})
         !  Control_{1.1j} := Control^{(j)}_{peer(1.1j,j)}
         }}}
```

Condition (1) will make sure that a damage rectangle creates at most 1 wrong value of the control and that this wrong value practically does not propagate.

Remark 17.1. In the present discussion, we assume that $|Mail_k.Info| = |Info|$, which is not really true for a separable transition rule. However, as mentioned in Remark 7.9, the only modification for the case of a narrow mail track is that the *Info* track must be split up into narrower tracks (packets) and each narrower track must be mailed in a separate copy operation. ♢

The rule below shows when and how mail will be posted:

```
rule Post_mail {
   for k ∈ {−1.1, 1.1} do {
      cond {
         ?  Control_k = post and Kind ∈ {Member, Growth}
         !  cond {
            ?  Age ∉ [Q, (grow_start)]
            !  Mail_k := Fickle
            ?!  {
               Mail_k.Info := Info
               Mail_k.Addr := Addr
               }}}}}
```

Even if the posting of mail is requested, the posted mail will have value *Fickle* if the age of the sending cell is close to a work period transition of its colony (this will be the case with all growth cells).

Let us define, for $j = -\operatorname{sign}(k)$ and each subfield F of $Mail_k$,

$$Mail\_to\_receive_k.F = \begin{cases} Undef & \text{if } peer(k,j) \text{ is undefined,} \\ j \cdot Edge_j & \text{if } F = Fromnb,\ |k| = 0.1,\ 1 \le Edge_j < \infty, \\ Mail^{(j)}_{peer(k,j)}.F & \text{otherwise.} \end{cases}$$

The "hand-shaking" condition

$$Mail\_used(k)$$

says that the cell is free to rewrite $Mail_k$ since the value was already used by the neighbor in direction $j = \operatorname{sign}(k)$. It is defined to be true if either $peer(k,j)$ or $peer(k,-j)$ is undefined, or the following holds for all F:

$$Mail^{(j)}_{peer(k,j)}.F = \begin{cases} -j \cdot Edge_j & \text{if } F = Fromnb,\ k = 1.1j,\ 1 \le Edge_j < \infty, \\ Mail_k.F & \text{otherwise.} \end{cases}$$

Information on the $Mail_k$ tracks moves under the effect of the following rule, except when it is overridden by *Post_mail*.

```
rule Move_mail {
   pfor k in Mail_ind do {
```

? $Control_k = get$ `and` $Mail\_used(k)$
! $Mail_k := Mail\_to\_receive_k$
}}

Arriving mail from neighbor m will be stored in location $\_Retrieved_m$ on track $Port_m$. Let us call by $\_Retrieved_m.Status$ the status track of the location $\_Retrieved_m$. Locations $\_Retrieved_m$ will be checked as to whether the information retrieved in them is transitional (a similar situation arises in the proof of Theorem 8.10 (Reach Extension)). The program

$$Check_0(check\_retrieve, Ck\_res, loc_1, loc_2, \dots)$$

checks whether for each of the locations that are its arguments, there is a $c \in \{Normal, Undef\}$ such that with the possible exception of 3 cells, all symbols are equal to c. The exceptions allow for one damage rectangle in each interval of size QB along the way from which information is retrieved. The checking rule $Check_1()$ was defined in 16.2, and the program *recheck_1* was defined at (16.4).

`subrule` *Age_check* {
$Check_1(check\_retrieve, Ck\_res, (\_Retrieved_m.Status : m \in Nb\_ind))$;
$Check_1(recheck\_1, Ck\_res, Ck\_res, 2)$;
}

Lemma 17.2 (Age Check)**.**

(a) *Assume that at the beginning of an application of Age_check, for all m, with the possible exception of 4 symbols, either all elements of the string in the location $\_Retrieve_m.Status$ are Normal or all are Undef. Assume moreover that either the check is damage-free or we can write 3 in place of 4 above. Then at the end, with the possible exception of 2 adjacent cells, all cells of the colony have $Ck\_res = 1$.*

(b) *Assume that at the beginning of Age_check, for some m, it is not true that with the possible exception of 4 symbols, either all elements of the string in the location $\_Retrieve_m.Status$ are Normal or all are Undef. Then at the end, with the possible exception of 2 adjacent cells, all cells of the colony have $Ck\_res = 0$.*

Proof. The proof is similar to the proof of Lemma 16.1 (Refresh). □

The subrule *Retrieve* is essentially a repetition of another subrule.

`subrule` *Retrieve* {
$Ck\_res := 0$
`repeat 5 times` {
Retr_cycle;
idle for $\lambda(End - (\text{grow\_start}) + Q)$ steps
}}

The long idle time between the application allows for getting the neighbor colonies out of a possibly transitional situation. The rule *Retr_cycle* will send the commands *post* and *get* on its control lines. Command *post* is repeated just enough times to get it across. Command *get* is repeated enough times for the mail track to transfer the information from any neighbor. During this time, the rule will snatch information from each track $Mail_k$ to the appropriate track $Port_m$. Then, the information will be checked for safety using *Age_check*. If it is safe then the retrieval will not be repeated: the rule still goes through the same number of steps but without resetting the receiving area and without any snatching.

`rule` *Retr_cycle* {
`cond` {
? $Ck\_res = 0$

```
          !  for m ∈ Nb_ind ∖ {0} do {
             Write(Vacant_str, _Retrieved_m);   }};
      repeat 3Qλ times {
(1)      for j = −0.1, 0.1 do {
             Control_j := post   }};
(2)   repeat 2Qλ times {
         for j = −0.1, 0.1 do {
             Control_j := get   }}
(3)   ‖ cond {
         ?  Ck_res = 0
         !  for j = −0.1, 0.1 do {
            cond {
               ?  Addr = Mail_j.Fromaddr
               !  Port_{Mail_j.Fromnb} := Mail_j   }}};
      Age_check
      }
```

The total number of steps of *Retrieve* can be estimated as

$$c_1 Q\lambda \|\mathbb{S}_k\|/w_k$$

for an appropriate constant c_1.

17.2. **Applying the computation rules.** For proving part (e) of Condition 8.8 (Computation Property) for big cells, assume that for some rational a, we have

$$\mathit{Damage}^* \cap [x - 3QB, x + 4QB-] \times [a - T_\bullet{}^*/2+, a + 2T^{\bullet *}] = \emptyset. \tag{17.3}$$

By Lemma 16.6, if there is no switching time during $[a+, a + 2T^{\bullet *}]$ then $\eta^*(x, a+) = \mathit{Vac}$.

Lemma 17.3 (Retrieval)**.** *Assume that σ_0 or σ_1, σ_2 is defined for η^*. Then we have case* (e) *or* (f) *or* (g) *of Condition 8.8 (Computation Property).*

Proof. As in the proof of Lemma 16.5 (Legality), consider a time v_2 such that $\eta^*(x, v_2) \neq \mathit{Vac}$. If σ_0 is defined then $v_2 = \sigma_0$, while if σ_2 is defined then it is some time very close from below to σ_2. Let us define v_1 as in that proof. As in part 2 of that proof, the Large Progress Lemma can extend the path repeatedly backward from v_1 until we either arrive at either an outer or germ cell or at a cell $c_0 = (x_0, v_0)$ in the same work period, with $\mathit{Age}(c_0) \leq 2Q$.

1. Assume that c_0 is a germ cell. Then we have case (g) of the Computation Property.
   Proof. Repeated application of the Large Progress Lemma gives a time interval long enough during which the colony and the adjacent colonies on both sides have been covered by germ cells. This implies that the state of the new big cell is latent and also that for the big cell x, we have $\eta^*(\vartheta_j(x, t, \eta^*), t) = \mathit{Vac}$ for some time t in $[a+, \sigma_0-]$ and for $j \neq 0$.
2. Now assume that c_0 is a member cell.
   From time v_0 on, we can follow the development of colony $\mathcal{C}$. The computation process can be treated similarly to the proof in the Legality Lemma; the additional problem is the retrieval of the needed information from the neighbor colonies in such a way that all retrieved information can be attributed to a single time instant. (This issue was solved once already in the proof of Theorem 8.10 (Reach Extension)).

Event (17.3) implies that in the whole space-time area in which this communication takes place only the usual damage rectangles must be considered. Due to (13.8), the whole computation fits

into an interval of length $T_\bullet{}^*$ and therefore in each direction, at most 3 damage rectangles can affect the retrieval (considering communication with a non-adjacent neighbor colony).

The extension arms of the colony will be extended by the rule *Extend*, defined in Subsection 12.4. The success of carrying out the extension over possible latent or germ cells or an opposing extension arm in the way is guaranteed by Lemma 12.8 (Creation). If the extension cells of $\mathcal{C}'$ are growth cells then they are stronger but the age check will fail in this case, so we can assume that this is not the case. If there are extension arms from both colony $\mathcal{C}$ and a non-adjacent neighbor colony $\mathcal{C}'$ then the strength ordering gives preference to one side and therefore soon only one of the extension arms remains.

Part (1), of rule *Retr_cycle* sends the message *post* along the control line $Control_{0.1}$ to the cells of the nearest colony $\mathcal{C}'$ (or its extension) on the left communicating with $\mathcal{C}$. The existence and condition of $\mathcal{C}'$ will be discussed below. This message will be maintained by the healing rule as a locally maintained field. Its value arrives on control line $Control_{1.1}$ to $\mathcal{C}'$ (or its extension). There, its value is propagated (carefully) by *Propag_control*. Via the rule *Post_mail*, the signal causes the cells of $\mathcal{C}'$ to post the needed information.

In due time, part (2), of rule *Retr_cycle* sends the message *get* along the same control line to colonies $\mathcal{C}'$ and $\mathcal{C}$; this causes the track $Mail_{1.1}$ of $\mathcal{C} - QB$ to pass its information to the right, to track $Mail_{0.1}$ of $\mathcal{C}$, which passes it further to the right. Damage can occur during the passing phase, but after healing its effect is confined to information (not the control fields) in the cells (at most two) which it changed. During the same phase, due to the parallelly running part (3) of *Retrieve*, the $Port_m$ fields of each cell of $\mathcal{C}$ snatch the information addressed to them from the right-moving track $Mail_{0.1}$.

At the end of *Retr_cycle*, rule *Age_check* will be applied. By Lemma 17.2 (Age Check), at the end of the application of this rule, all but maybe 2 cells of the track *Check_res* contain the same value. We will say that the process *passed* or *failed the age check*. The rule *Retr_cycle* is iterated 5 times but it will not affect the $Port_m$ tracks after it passes the age check.

2.1. The age check can fail at most 4 times.

Proof. For every neighbor colony from which mail is to be retrieved, at most two iterations of *Retr_cycle* can fail the age check. Normally, there can only be one such failure, when the colony was found in a transition age: this gives at most 1 failure on the left and 1 failure on the right. Due to the idling between iterations of *Retr_cycle*, the next iteration will find the colony past the transition age.

It can also happen that the neighbor colony was found in a transition age just before vanishing and next time it will be found in a transition age when it is just being recreated by a farther colony. However, it is easy to see that this can happen only for one neighbor colony on the left and one on the right, bringing the total number of failures to at most 4.

Take an iteration of *Retr_cycle* that passes the age check. Let t_0 be the earliest time after the beginning of this iteration when the whole area $D = [x - 2QB, x + 3QB]$ is free of the healable wake of any damage rectangle, and let t_1 be the end of this iteration. Let us call any colony $\mathcal{C}'$ with base x' in $[x - 2Q+, x - Q]$ a *potential left neighbor colony*.

2.2. Suppose that there is a potential left neighbor colony $\mathcal{C}'$ that stays covered during $[t_0, t_1]$ (a temporary hole due to the damage rectangle and healing within $4\tau_1$ time units does not count).

It is easy to see by Lemma 16.7 (Present Attribution) that all cells between $\mathcal{C}'$ and $\mathcal{C}$ (if any) not in the wake of a damage rectangle belong to extensions of either $\mathcal{C}'$ or $\mathcal{C}$.

The retrieval from $\mathcal{C}'$ proceeds according to the program. The mail will be passed from $\mathcal{C}'$ to $\mathcal{C}$ through the remaining extension arm (if any). The age check guarantees that all the retrieved information can be attributed to a single instant of time before t_1, since it will not change during the at most $3Q\tau_1$ time units needed for its reading (we count τ_1 for each mail-passing step, and add $Q\tau_1$ more time units for the possible push-back of extension arm by an opposite one).

According to Lemma 16.7 (Present Attribution), before retrieval, each of the strings to be retrieved represents some string within 2 deviations. During retrieval, damage can change at most 4 cells, increasing the number of deviations to at most to 6. Since we use a 6-error-correcting code the computation proceeds from here as expected.

2.3. Suppose now that there is no potential left neighbor colony that stays covered during $[t_0, t_1]$.

There are the following possibilities, according to the Present Attribution Lemma 16.7:

(1) No cells occur in $[x - QB, x-]$ during $[t_0, t_1]$;

(2) Growth to the right from some colony $\mathcal{C}' - QB$ where $\mathcal{C}'$ is a potential left neighbor colony;

(3) A potential left neighbor colony $\mathcal{C}'$ is beginning to disappear as in part (4) of the Lemma 16.7 (Present Attribution). Then within $2Q\tau_1$ time units hereafter all cells of $\mathcal{C}'$ disappear.

Let us look at the $Port_i$ track before the age check, for $i \in \{-1, -1.5\}$. If for some cell x on it $Port_i(x)$ is not the direct result of damage and $Port_i.Status(x) \neq Undef$ then this value was obtained from a potential neighbor colony $\mathcal{C}'$ by mail during the last retrieval cycle. In all the cases considered, this value is then *Fickle*. Since the age check passes, Lemma 17.2 (Age Check) implies that, with the possible exception of 2 intervals of 2 cells, the cells of $\mathcal{C}$ have $Port_i.Status = Undef$. Thus at the end of *Retrieve*, the string on the $Port_i$ track represents a vacant value within 4 errors.

2.3.1. There is a time interval of length at least $3Q\tau_1$ before t_1 in which all big cells corresponding to potential left neighbor colonies $\mathcal{C}'$ are vacant and in which therefore the information collected by *Retrieve* can be attributed to any instant.

Proof. In case (1), the big cell corresponding to colony $\mathcal{C}'$ is indeed empty all the time. In case (2), the growth process from colony $\mathcal{C}' - QB$ could not have started long before t_0 since so few cells reported unsafe age: therefore the duration of growth provides the necessary long interval. In case (3), cells of $\mathcal{C}'$ start vanishing within Q steps after t_0. Otherwise namely the $Port_i$ track would snatch values with $Status = Fickle$ and then would not pass the age check. Once the vanishing of colony $\mathcal{C}'$ started, it will be over within $Q\tau_1$ time units. Repopulating it by growth will take sufficiently long that it does not create a new cell during this retrieval cycle.

3. Assume that c_0 is an outer cell. Then we have case (f) of the Computation Property.

Proof. The fact that the new big cell is latent can be similarly concluded. The Attribution Lemma allows us to attribute the outer cell in question to a neighbor colony, say with base cell $x - QB$ on the left, from which it has grown. This implies $Growing_1 = 1$ for this colony, which, as seen above, implies $Creating_1 = 1$ in the big cell $x - QB$.

Now the reasoning of 2 above can be applied to this big cell, seeing that it computes its next state according to the transition rule from information retrieved from its neighbors. In particular, it applies the rule *Arbitrate* and therefore can only have $Creating_1 = 1$ if the big cell x is vacant when observed.

Therefore, on the other hand, the creation of the latent big cell x can be attributed to the rule *Create*, hence the transition rule was also satisfied in big cell x.

□

The following lemma covers the cases left after the above lemma.

Lemma 17.4 (Growth)**.** *Assume that we have* (17.3) *and* $\eta^*(x,t) = Vac$ *for* $t \in [a+, a+2T^{\bullet *}]$. *Then part* (h) *of Condition 8.8 (Computation Property) applies to* η^*.

Proof. Let $\mathcal{C}$ be the colony whose base is x. If there is any (y,t) with $0 < |y-x| < QB$, $t \in [a+, a+2T^{\bullet *}]$ and $\eta^*(y,t) \neq Vac$ then we are done; assume therefore that there is no such y. If there is any t in $[a+, a+2T^{\bullet *}]$ when both $\eta^*(x-QB,t)$ and $\eta^*(x+QB,t)$ are vacant then at this time there is no potential creator, and we are done. The case remains when for all t in $[a+, a+2T^{\bullet *}]$ there is a $j \in \{-1,1\}$, with $\eta^*(x+jQB,t).Creating_{-j} = 1$ and there is no big cell y during this time whose body intersects the potential body of the big cell x. We will show that this case does not occur. Without loss of generality, suppose

$$\eta^*(x-QB, a+2T^{\bullet *} - {T_\bullet}^*/2).Creating_1 = 1.$$

1. Suppose that there is no t in $[a+, a+2T^{\bullet *} - {T_\bullet}^*/2]$ with $\eta^*(x+QB,t).Creating_{-1} = 1$. Then for all t in this interval, we have $\eta^*(x-QB,t).Creating_1 = 1$. Tracing backward and forward the evolution of the colony of the big cell $x-QB$ (and applying Lemmas 16.5 (Legality) and 17.3 (Retrieval), we will find a whole work period $[u_1,u_2]$ of colony $\mathcal{C}-QB$ in $[a-{T_\bullet}^*/2, a+2T^{\bullet *} - {T_\bullet}^*/2]$. At the end of this work period, growth to the right takes place.

   If the growth succeeds this would contradict our assumption: suppose that does not. This can only happen, according to Lemma 12.8 (Creation), if some non-germ cell z is in the way. Lemma 16.7 (Present Attribution), shows that z is a left extension cell of a live big cell y in $[x+QB+, x+2QB-]$, and is therefore not stronger than the right growth it is keeping up.

   The rule *Arbitrate* kills z since it prefers right growth to the left one, and therefore z does not really prevent the right growth from succeeding. Since this preference is arbitrary assume that the rule *Arbitrate* actually prefers growth in the left direction and that z is a left growth cell. We can trace backward and forward the evolution of the colony of y to see that it carries its growth to conclusion, resulting in a new big cell $y-QB$ before time $a+2T^{\bullet *}$, whose body intersects the body of x. This has also been excluded.
2. Suppose that there is also a t in $[a+, a+2T^{\bullet *} - {T_\bullet}^*/2]$ with $\eta^*(x+QB,t).Creating_{-1} = 1$. Tracing backward and forward the evolution of the colony of the big cells $x-QB$ and $x+QB$ (and applying the Legality and Retrieval lemmas) it is easy to see that they are non-vacant for all t in $[a+, a+2T^{\bullet *}]$ (since according to Condition 12.6, $Creating_j = 1$ implies $Dying = 0$).

   It is easy to see that there is a $j \in \{-1,1\}$ and $[t_1,t_2] \subset [a-{T_\bullet}^*/2, a+2T^{\bullet *}]$ such that $[t_1,t_2]$ is a dwell period of big cell $x+jQB$ with $Creating_{-j} = 1$. Assume first that $j=-1$ can be chosen.

   In the colony $\mathcal{C}$ of x between the colonies $\mathcal{C}-QB$ and $\mathcal{C}+QB$, due to Lemma 16.7 (Present Attribution), all non-germ cells belong to the extension of one of these two colonies. The growth rule of $x-QB$ will try to create a latent big cell in $\mathcal{C}$. If the growth rule of $\mathcal{C}+QB$ is not active

then there are no obstacles to this creation and it succeeds, contrary to the original assumption: so we are done. Moreover, it succeeds even if the growth rule of $\mathcal{C}+QB$ is active since the strength relations favor growth from the left.

Assume now that only $j = 1$ can be chosen. The reasoning is similar to the above except when the growth rule of $\mathcal{C}-QB$ is active at the same time when $\mathcal{C}+QB$ is trying to grow left. This can only happen when the dwell period $[t_1, t_2]$ of big cell $x+QB$ overlaps with a dwell period $[u_1, u_2]$ of big cell $x-QB$ with $\mathit{Creating}_1 = 1$. By our assumption, $[u_1, u_2] \not\subset [a - T_\bullet{}^*/2, a + 2T^{\bullet *}]$.

In the next reasoning, we will use the fact that growth is confined to a time interval of length at most $T_\bullet{}^*/2$ at the end of any colony work period.

Assume $t_2 > a + 2T^{\bullet *} - T_\bullet{}^*/2$. Then we must have $u_2 > a + 2T^{\bullet *}$. Hence the previous dwell period $[u_0, u_1]$ of $x-QB$ is in $[a+, a + 2T^{\bullet *}]$ and must have $\mathit{Creating}_1 = 0$. Therefore $\mathit{Creating}_{-1} = 1$ in the previous dwell period $[t_0, t_1]$ of $x+QB$. Due to the confinement of growth to the end of the work period, the growth to the left from $\mathcal{C}+QB$ at the end of the period $[t_0, t_1]$ will be undisturbed by growth to the right from $\mathcal{C}-QB$ and the creation succeeds near time t_1, contrary to the original assumption.

Assume now $t_2 \le a + 2T^{\bullet *} - T_\bullet{}^*/2$. Then due to the confinement of growth just mentioned, we have $u_2 \le a + 2T^{\bullet *}$. Hence, $u_1 < a - T_\bullet{}^*/2$, and the next dwell period of $x-QB$, contained entirely in $[a+, a + 2T^{\bullet *}]$, must have $\mathit{Creating}_1 = 0$. Then the next dwell period of $x+QB$ is still entirely in $[a+, a + 2T^{\bullet *}]$ and has $\mathit{Creating}_{-1} = 1$, and the creation to the left succeeds.

□

17.3. **The error parameters.** Lemma 9.4 (Amplifier) says that given a (broadcast) amplifier frame *Frame* with large enough R_0, we can complete it to a uniform (broadcast) amplifier. We leave out the case of broadcast amplifiers: the proof comes with some simplification from the proof for general amplifiers.

The main ingredient of our amplifier is the sequence of transition functions $\mathit{Sim\_trans}_k$ and the sequence of codes φ_{k*}, φ_{k**}, φ^*_k, which was defined in the course of the proof. It remains to verify the amplifier properties listed after (9.21). Let us recall them here.

(a) (Rider^k) with guard fields (Guard^k) is a guarded shared field for (φ_{k*}), (φ_{k**}), as defined in Subsection 4.2;
(b) the damage map of the simulation Φ_k is defined as in Subsection 8.1;
(c) Tr_k is combined from $\mathit{Rd\_trans}_k$ and $\mathit{Sim\_trans}_k$, with fields Rider^k, Guard^k;
(d) $M_k = \mathit{Rob}(Tr_k, B_k, T_{\bullet k}, T^\bullet{}_k, \varepsilon_k, \varepsilon'_k)$;
(e) Φ_k has ε''_k-error-correction for φ_{k**};

The first three properties follow immediately from the definition of Tr_k and the code, given in the preceding sections. It has also been shown, by induction, that M_{k+1} is a robust medium simulated by M_k via the code, with ε_k as the error bound and $T_{\bullet k}$, $T^\bullet{}_k$ as the work period bounds. It needs to be shown yet that ε'_k indeed serves as the bound in the definition of $M_k = \mathit{Rob}(\cdots)$, and that the simulation has ε''_k-error-correction.

ε'_{k+1} must bound the difference from 0.5 of the probability of the new coin-toss of the simulated computation, and the simulation in the work period must be shown to have ε''_k-error-correction. We must show that in a big cell transition, the probability that the Rand^* field is 1 is in $[0.5 - \varepsilon'_{k+1}, 0.5 + \varepsilon'_{k+1}]$. (We also have to show that the bounds on the probabilities are of the form of sums and products as required in the definition of canonical simulation, but this is automatic.)

In case there is no damage rectangle during the whole work period, the field *Rand** was computed with the help of the rule *Randomize*. This rule took the value X found in field *Rand* of the base cell of the colony and copied it into the location holding *Rand**.

Remark 17.5. This is the point where we use that our simulation is a canonical simulation but not necessarily a deterministic one: the time at which the the base cell tosses the coin can be found using a stopping time within the colony work period. ♢

By the property of M_k, the probability that $X = 1/2$ is within ε'_k of 0.5. By its definition, ε''_k upper-bounds the probability that any damage rectangle intersects the colony work period. Therefore the probability that *Rand*$^* \neq X$ can be bounded by ε_k. Hence, the probability that the *Rand** field is 1 is in

$$[0.5 - \varepsilon'_k - \varepsilon''_k, 0.5 + \varepsilon'_k + \varepsilon''_k] = [0.5 - \varepsilon'_{k+1}, 0.5 + \varepsilon'_{k+1}].$$

As just noted, with probability ε''_{k+1}, no damage rectangle occurs during a colony work period. Under such condition, the rule *Refresh* works without a hitch and each cell contains the information encoded by the code φ_{**} from the state of the big cell. This shows that our simulation has the ε''_k-error-correction property.

18. Germs

18.1. **Control.** Recall the definition of $\Gamma(I, d)$ in (10.2).

Lemma 18.1. *Let x_0 be a site and $t_1 < t_2$ times. Assume that (x_0, t_1) is controlled and $\Gamma(x_0, 3B) \times [t_1 - 8T^\bullet, t_2]$ is damage-free. Then (x_0, t_2) is controlled.*

Proof. In the proof below, *controlling* always means controlling (x_0, t_1). Let $t_0 = t_1 - 6T^\bullet$. For t in $[t_0, t_2]$, let

$$A(t) = \Gamma(x_0, 3B) \times [t_0, t].$$

1. Let $t \in [t_1, t_2]$. Assume that x_0 is controlled for all $t' < t$. Then $[x_0 - 3B, x_0 + 3B-] \times \{t\}$ contains a cell.
   Proof. By our assumption, for all $t' < t$, each set $[x_0 - B, x_0 + B-] \times [t' - 6T^\bullet, t']$ contains a cell x_1. If it stays until time t then we are done. If it disappears the death must have been caused by *Arbitrate*, due to a cell y_1 with $|y_1 - x_1| < 2B$ about to create another cell whose body intersects with the body of x_1. Cell y_1 had $\mathit{Dying} = 0$ to be able to kill and therefore survives until time t.
2. Let $t \in [t_1, t_2]$. Assume that x_0 is controlled for all $t' < t$. Then $\Gamma(x_1, B) \times [t - 6T^\bullet, t]$ contains a cell.
   Proof. Bu assumption, for all $t' < t$, each set $\Gamma(x_0, B) \times [t' - 6T^\bullet, t']$ contains a cell x_1. Without loss of generality, suppose $x_1 \le x_0$. If this is also true for $t' = t$ then we are done. Suppose it does not: then one such cell x_1 disappears at time $t - 6T^\bullet$. This must have been caused by the rule *Arbitrate*, due to some cell y_1 about to create an adjacent neighbor whose body intersects with the body of x_1. Without loss of generality, assume $y_1 \in [x_1 - 2B+, x_1 - B-]$. According to the *Arbitrate*, cell y_1 must have had $\mathit{Dying} = 0$, $\mathit{Creating}_1 = 1$ to be able to erase and therefore survives until time t. Since according to Condition 12.6, only the rule *Arbitrate* changes $\mathit{Creating}_1$, and only when a right neighbor has appeared, cell y_1 keeps trying to create a right neighbor.

   2.1. Suppose that y_1 succeeds in creating $y_1 + B$ before time $t_0 + 2T^\bullet$.

   If $y_1 + B \ge x_0 - B$ then it will control; otherwise, $y_1 + B$ will try to create $y_1 + 2B$. If it succeeds in $2T^\bullet$ time units then the created cell will control; if it does not then then according to the Computation Property, another cell in $[y_1 + 2B+, y_1 + 3B-]$ interferes and it will control.

   2.2. Suppose that the creation does not succeed within $2T^\bullet$ time units.

   Then a cell $x_2 \in [y_1 + B+, y_1 + 2B-]$ interferes. If $x_2 \ge x_0 - B$ then x_2 will control, suppose therefore that $x_2 < x_0 - B$. We have $|x_1 - x_2| < B$ since $x_1, x_2 \in [y_1 + B, y_1 + 2B]$. Therefore x_2 must have arisen after x_1 disappeared. Due to part 1 above, x_2 could not have arisen by spontaneous birth, hence it must have been created by a right neighbor. If x_2 was created after t_0 then its creating right neighbor is alive after t_0 and it will control. Suppose therefore that x_2 was created at time t_0 and its right neighbor disappeared by time t_0.

   Suppose that x_2 stays for at least $2T^\bullet$ time units; then it tries to create $x_2 + B$. If it succeeds then $x_2 + B$ will control; if it does not then the cell that interferes in the creation will control.

   Suppose that x_2 disappears before $t_0 + 2T^\bullet$. Suppose that y_1 now succeeds in creating $y_1 + B$ before time $t_0 + 4T^\bullet$. If $y_1 + B \ge x_0 - B$ then it will control; otherwise, it will try to create $y_1 + 2B$. If it succeeds in $2T^\bullet$ time units then the created cell will control. If it does not then the cell that interferes will control.

Suppose that y_1 still does not succeed in its creation. Then another cell x_3 appears before $t_0 + 4T^\bullet$ that prevents this. This x_3 must have been created by a right neighbor that will control.

□

Lemma 18.2. *The media of our amplifier have the lasting control property.*

Proof. Assume that for some sites x_0 and times $t_1 < t_2$, with

$$I = \Gamma(x_0, ((t_2 - t_1)T^\bullet/T_\bullet + 4)B),$$

$I \times \{t_1\}$ is blue and $I \times [t_1 - 2T^\bullet, t_2]$ is damage-free. We have to prove that (x_0, t_2) is blue.

Lemma 18.1 implies that this point is controlled; what remains to show is that the controlling cell in $\Gamma(x_0, B) \times [t_2 - 6T^\bullet, t_2]$ is blue. Due to part 1 above, in the area considered, there is always some cell within distance $3B$ left or right of any site, and this prevents any germ cell from being born. If therefore the controlling cell would be non-blue then one could construct from it a path of non-blue cells backwards. This path cannot reach outside I during $[t_1, t_2]$ since it needs at least $T_\bullet$ time units to move one cell width away from x_0. □

Let us introduce an undirected graph $G(\eta)$ on space-time points of a space-time configuration η by defining below the following kinds of edge.

(1) points (x, t) and (x, t') are connected for $t < t'$ if a cell of η is present in x for all of $[t, t']$;
(2) suppose that $|x - y| < 2B$ and there is a cell at time t in both x and y, and moreover there is a cell in y during the a whole dwell period of x containing t. Then (x, t) and (y, t) are connected.
(3) if a cell has just been created in (x, t) by an adjacent cell y whose last dwell period before t begins in t' then (x, t) and (y, t') are connected;

These are all the edges of graph G.

Lemma 18.3. *Let* $t_0 = t_1 - 8T^\bullet$,

$$I = \Gamma(x_0, 9B).$$

Assume that $I \times \{t_0\}$ *is controlled and the area* $I \times [t_0 - 8T^\bullet, t_1]$ *is damage-free. Then there are sites* $z_1 \le x_0 \le z_2$ *and a path in the graph* G *defined on* $I \times [t_0, t_1]$*, connecting* (z_1, t_1) *with* (z_2, t_1).

Proof. Since (x_0, t_0) is controlled there is a cell (x_1, u_1) in $\Gamma(x_0, B) \times [t_0 - 6T^\bullet, t_0]$. Without loss of generality, assume that $x_1 \ge x_0$.

1. Suppose that x_1 stays a cell until time t_1.

Since $(x_1 - B, t_0)$ is controlled there is a cell (x_2, u_2) in $\Gamma(x_1 - B, B) \times [t_0 - 6T^\bullet, t_0]$. If the cell in x_2 exists until time t_1 then necessarily $x_2 < x_0$, and it is easy to see that there is a time v_1 such that the points (x_1, t_1), (x_1, v_1), (x_2, v_1), (x_2, t_1) form a path in G.

1.1. Suppose that the cell in x_2 lasts until $t_0 + 2T^\bullet$.

Then it coexists with x_1 at time t_0, therefore $x_2 \in [x_1 - 2B+, x_1 - B]$. Since $(x_2 - B, t_0)$ is controlled there is a cell (x_3, u_3) in $\Gamma(x_2 - B, B) \times [t_0 - 6T^\bullet, t_0]$. If the cell in x_3 exists until time t_1 then $x_3 \le x_2 - B$, and it is easy to see that there are times v_i such that the points (x_1, t_1), (x_1, v_1), (x_1, v_2), (x_2, v_2) (x_2, v_3), (x_3, v_3), (x_3, t_1) form a path in G.

If the cell in x_3 disappears before time t_1 then this has been caused by *Arbitrate*, via a cell y_1 about to create another cell intersecting with the body of x_3. As seen in similar discussions above, cell y_1 exists during the whole of $[t_0 - 8T^\bullet, t_1]$. Therefore it is not equal to x_2 which is assumed to disappear. But its body does not intersect the body of of either x_2 or x_3 hence $y_1 \in [x_3 - 2B+, x_3 - B-]$.

If cell x_3 lasts until $t_0 + 2T^\bullet$ then it is easy to see that there are times v_i such that the points (x_1, t_1), (x_1, v_1), (x_1, v_2), (x_2, v_2) (x_2, v_3), (x_3, v_3), (x_3, v_4) (y_1, v_4), (y_1, t_1) form a path in G.

Suppose that x_3 disappears before $t_0+2T^\bullet$. Now y_1 tries to create y_1+B which (if this succeeds) tries to create $y_1 + 2B$, etc. until reaching x_1. If we reach x_1 this defines a path again between (x_1, t_1) and (y_1, t_1).

Suppose that one of these creations, e.g. the creation of $y_1 + B$ does not succeed. Then a cell exists in

$$[y_1 + B+, y_1 + 2B-] \times [t_0 + 2T^\bullet, t_0 + 4T^\bullet].$$

The space is not sufficient for this cell to be born spontaneously, therefore it can be traced back via creations to one of x_1, x_2, x_3 or to some cell existing during $[t_0, t_0 + 2T^\bullet]$ between y_1 and x_2. This way again a path can be defined.

1.2. Suppose that the cell in x_2 disappears before $t_0 + 2T^\bullet$.

This has been caused by *Arbitrate*, via a cell y_1 about to create another cell intersecting with the body of x_2; as above, cell y_1 exists during the whole of $[t_0 - 8T^\bullet, t_1]$. Then either $y_1 = x_1$ or $y_1 \in [x_2 - 2B+, x_2 - B-]$. In the second case, there are points v_i such that (x_1, t_1), (x_1, v_1), (x_1, v_2), (x_2, v_2) (x_2, v_3), (y_1, v_3), $y_1, t_1)$ form a path in G. Suppose therefore $y_1 = x_1$.

After x_2 dies, either y_1 succeeds creating $y_1 - B$ before time $t_0 + 4T^\bullet$ or some cell (x'_2, u'_2) interferes with this. In the first case, let us call $x'_2 = y_1 - B$. In both cases, we can continue as in part 1.1 above with x'_2 in place of x_2 and $t_0 + 5T^\bullet$ in place of $t_0 + 2T^\bullet$.

2. Suppose that the cell in x_1 disappears before time t_1.
This has been caused by *Arbitrate*, via a cell y_1 about to create another cell intersecting with the body of x_1; again, y_1 exists through all the time considered here. If $y_1 > x_1$ then we can continue as in part 1 above with y_1 in the role of x_1 above and x_1 in the role of x_2 above. If $y_1 < x_1$ then we can do the same, after a left-right reflection.

□

Lemma 18.4. *The simulations of our amplifier have the control delegation property.*

Proof. Let $M = M_k$, $\Phi = \Phi_k$ for some k, for our amplifier, and let η be a trajectory of M. Let $T^\bullet = T^\bullet{}_k$, $T^{\bullet *} = T^\bullet{}_{k+1}$, etc. Let

$$I = \Gamma(x_0, (8T^{\bullet *}/T_\bullet{}^* + 4)QB).$$

Assume that $I \times \{t_1 - 8T^\bullet{}_2\}$ is blue in $\eta^* = \Phi^*(\eta)$ and $I \times [t_1 - 16T^\bullet{}_2, t_1]$ is damage-free in η. We must prove that (x_0, t_1) is blue in η.

As mentioned at the beginning of Section 11, the simulation φ^*_k consists of two parts: the simulation of M_{k+1} by a medium M'_k of reach 2, and the simulation of M'_k by M_k as shown in Theorem 8.10. Control delegation from M'_k to M_k is trivial, since the cells of the two media are essentially the same; hence, we need to consider only control delegation from M_{k+1} to M'_k.

Lemma 18.3 implies that there are sites $z_1 \le x_0 \le z_2$ and a path in the graph $G(\eta^*)$ defined on $I \times [t_0, t_1]$, connecting (z_1, t_1) with (z_2, t_1). This path consists of blue big cells: indeed, since the area is controlled in η^* birth is excluded, therefore a non-blue big cell would have to trace back its origin to outside I, but the complement of I is too far to do this in the available time. The path gives rise in a natural way to a sequence $(c_1, \dots, c_n)$ of small cells with $c_1 = (z_1, t_1)$, $c_n = (z_2, t_1)$, and for each i, either c_i and c_{i+1} are in two consecutive dwell periods of the same cell, or have the form $(x, t), (y, t)$ with $|x - y| < 2B$. All these cells are member or extension cells of the colonies of

the big cells of the path. Since *Color* is a broadcast field and $Guard = -1$ in all our cells, these small cells are all blue. Let us form a polygon P_1 connecting these points with straight lines.

Let A be the set of elements of $I \times [t_1 - 16T^\bullet{}_2, t_1]$ reachable from (x_0, t_1) along any polygon at all without crossing P_1. It is easy to see that A is controlled; let us show that it is also blue. Indeed, according to Lemma 18.1, not birth can take place in a controlled area, therefore from any cell in A a path of ancestors passes to the outside of A. When it crosses P_1 one of its cells must be equal to a blue cell and therefore the whole path must be blue. □

18.2. **The program of a germ.** We now add some rules to the program of our amplifier to make it self-organizing. A germ starts out as a single cell with $Age = 0$ and $Addr = -Q$ and tries to grow into an area covering 3 colonies by growing right, but can be held up by some other cells. The cell with address $-Q$ will be called the *leader* of the germ (calling it the "base" would be confusing). Germ cells are weaker than non-germ cells, hence a growing germ cannot intrude into an extended colony and a growing colony can destroy germ cells in its way. In what follows, the germ growth rule is based on *work periods* just as colonies. Not all germ work periods are alike: each work period is associated with a *level*. Let

$$l = \log(3Q) \tag{18.1}$$

(not necessarily an integer). The level of the germ is stored in a field

$$Level \in [0, l-] \cup \infty$$

of each of its cells. Cells stronger than germ cells have level ∞. Latent cells are level 0 germ cells; thus, even latent cells have color. *Age* will be reset to 0 at the beginning of each work period, and the work period of level s lasts until $Age = U(s)$ where

$$U(s) = 4p_1\lambda 2^s.$$

We say that a work period has level s, if the leader cell has that level. Typically, in this case the germ has already been extended to a size at least $2^s B$. Therefore if a right edge cell has $Addr + Q < 2^{Level} - 1$ then it will be called *exposed*. This extends the definition of exposed edges given in Subsection 12.1 without affecting any previous reasoning using this notion. There is also a field

$$Active \in \{0, 1\}.$$

Cells with $Active = 1$ are called *active*, others *passive*. Here are the rules using these properties:

1. If a germ cell sees a (not necessarily adjacent) germ cell on the left with the same color but higher level then it becomes vacant.
2. When a passive germ cell of level s sees an active germ cell on the left, of the same color and level, with $Age \le U(s) - 2p_1 2^s$, then it turns vacant.

The work period is divided into two parts: *growth and computation*. In growth, the right end attempts to grow to size 2^{s+1}. The computation part is omitted if the right end has address $2Q - 1$ (the final goal). This part, which takes the 2^{s+2} last steps of the work period, checks whether the germ has size $2^{s+1}B$, by propagating left the information about the address of the right edge in a field *New_level*. If *New_level* > *Level* then in the next work period, each cell sets *Level* := *New_level*. Otherwise, the leader cell chooses a random number in $\{0, 1\}$ and broadcasts it right on the track *Active*. This ends the computation.

Remark 18.5. Probably a much simpler rule (e.g. a random decision each time there is a conflict) would do but our rule makes the proof very simple. On the other hand, the estimates of the proof would be much better with a somewhat more complicated rule that allows for some error-correction in the germ cells, e.g. erasing a small germ surrounded by cells of different color. ♢

18.3. **Proof of self-organization.**

Lemma 18.6 (Germ Attribution)**.** *Let $c_1 = (x_1, t_1)$ be a blue germ cell, of level s. Assume that the area*

$$[x_0 - 3 \cdot 2^s B, x_0 + 2^s B] \times [t_1 - 2 \cdot 2^s \tau_1, t_1 + 2U(s)\lambda]$$

is free of damage and of nonblue cells. Then either c_1 is part of a domain with no exposed edge or there is a blue cell of higher level in the above area.

Proof. Let us construct a path P_1 backward from c_1 in such a way that we keep the path on the left edge of the germ: thus, whenever possible we move left on a horizontal link, otherwise we move back in time on a vertical link. At time

$$t_0 = t_1 - 2^s \tau_1,$$

the path might stop at a cell c_0 one level lower than c_1.

1. There is a time t_3 in $[t_0, t_1]$ when $P_1(t_3)$ is a leader and the germ has size $\geq 2^s$.
   Proof. Suppose this is not so. If the right end is exposed at time t_0 then it remains so and the germ decays away in the given time. If the left end is exposed during all of $[t_0, t_0 + (\text{split\_time})]$ then the left end decays, becomes unhealable, and the germ will decay. Suppose therefore that both ends are protected by some time $t_3' < t_0 + (\text{split\_time})$. If the right end has level s at this time, then it has the desired size, too.

   Suppose the germ at the time t_3 has size $< 2^s B$ and the right end has a lower level allowing this size. If the left end is younger than the computation start then the computation would never allow it to switch to a higher level with this small size, so the size $2^s B$ must be reached at some time t_3. If it is older than the computation start then (unless an end becomes exposed and the germs dies on that account) the age advances quickly to the new work period, and the whole germ obtains the new, higher level. If the germ is still small its right becomes exposed and then the whole germ will be wiped out.

Let us follow the development of the germ from t_3 forward. If the left end survives then the germ survives till time t_1 and we are done. If the left end will be destroyed by a cell c_2 with higher level then we are done.

The last possibility is that the left end will be destroyed by an active cell c_4 on the same level. Let us construct a path P_2 backward from c_4 in a way similar to P_1. By the reasoning of part 1 above we arrive at a time t_5 at which $P_2(t_5)$ is a leader and the germ has size $\geq 2^s$.

Following the development of this (active) germ forward, its left end either survives or will be killed by a cell c_2 of level $> s$ (in which case we are done, having found c_2). If the left end survives then the germ of c_4 kills the germ of c_1, growing over at least 2^s killed cells. (It is easy to see that new germ cells arising in place of the killed ones do not have sufficient time to reach level s and are therefore no obstacle to this growth.) This creates a germ of size $\geq 2^{s+1}$ whose level will increase after its next computation notices this. All this happens within at most 2 work periods of the germ of c_4. □

Lemma 18.7 (Level Increase)**.** *For a level $s \geq 0$, integers $n > 1$, $r \geq 3 \cdot 2^s B$, some site x_0, some rational time t_0, with $c_0 = (x_0, t_0)$, let*

$$\begin{aligned} d_1 = d_1(s) &= 3U(s)p_1, \\ L_0 &= \Gamma(x_0, 3 \cdot 2^s \lambda B), \\ I_0 &= \Gamma(L_0, 3QB), \\ G_0 &= [t_0 - 2 \cdot 2^s \tau_1, t_0], \\ G_1 &= [t_0 - 3T^\bullet, t_0], \\ H_1 &= [t_0, t_0 + nd_1 T^\bullet]. \end{aligned}$$

Assume

$$nd_1 T^\bullet < T_\bullet{}^*. \tag{18.2}$$

Let $\mathcal{G}_0$ be the event that $I_0 \times (G_0 \cup H_1)$ is damage-free, $I_0 \times G_1$ is blue, and c_0 has level s. Let $\mathcal{H}_1$ be the event that $L_0 \times (G_1 \cup H_1)$, is blue and contains a cell of level $s+1$. Then there is a constant $\kappa_1 > 0$ such that the probability of $\neg\mathcal{H}_1 \cap \mathcal{G}_0$ is at most $e^{-\kappa_1 n}$. Moreover, this estimate is also understood in the sense of an injective canonical simulation.

Proof.

1. All cells in $L_0 \times H_1$ are blue.

   Proof. Non-blue germ cells cannot arise in a blue area. If a non-blue non-germ cell would occur it could be attributed to a full non-blue colony. It takes two full consecutive colony-occupations to reach into L_0 from outside I_0, and this takes at least $T_\bullet{}^*$ time units which is, by the assumption (18.2), more than our total time.

Lemma 18.6 shows that, in the absence of damage, either the conclusion of the present lemma holds or c_0 is part of a germ of size at least 2^s, without exposed edges. Let $c_1 = (x_1, t_0)$ be the left endcell of this germ.

2. The following holds during

$$J_1 = [t_0, t_0 + d_1 T^\bullet] :$$

   either a cell with level $> s$ occurs in $\Gamma(x_1, 2^{s+1}B)$ or a leader cell $c_2 = (x_2, t_2)$ with level s occurs in $[x_1 + 2^s B, x_1 + 2^{s+1}B-]$. In the latter case, at time t_2, all cells between x_1 and x_2 belong to the germ of c_1, with no place left for cells between them.

   Proof. If c_1 is killed during $[t_0, t_0 + 2U(s)\tau_1]$ from the left by a higher-level cell then we are done. Suppose that it will be killed by an active cell c_2 of the same level. Then Lemma 18.6 (Germ Attribution) implies that c_2 is contained in a germ without exposed edges, hence of size $\geq 2^s$. This germ conquers the area of the germ of c_1 within the next $2U(s)\tau_1$ time units, and creates a cell of level $s+1$, which is one of the desired outcomes.

   Suppose that this does not happen. If the growth of the germ of c_1 will not be held up then it will create a cell of level $s+1$ within $2U(s)\tau_1$ time units. It can be held up only by a cell of higher or equal level. A cell of higher level would give what is required: suppose that this does not happen. If it is held up by a cell of the same level, then unless this cell is a leader it decays within time τ_1. Therefore the growth succeeds unless a leader cell c_2 of the same level is in the way and all cells between x_0 and x_1 belong to the germ of c_1, with no place left for cells between them.

3. Let $t_3 = t_2 + d_1 T^\bullet$, $J_2 = [t_2, t_3]$. Let $\mathcal{J}_1(t_2)$ be the event that $Level(x_1, t_2) = s$, there is a leader cell $c_2 = (x_2, t_2)$ of level s in $[x_1 + 2^s B, x_1 + 2^{s+1}B-]$, and at time t_2 all cells between x_1 and

x_2 belong to the germ of c_1, with no place left for another cell. The probability that $\mathcal{G}_0 \cap \mathcal{J}_1(t_2)$ holds and no cell of level $> s$ occurs during J_2 in $\Gamma(x_1, 2^{s+1}B)$ is at most

$$1 - (0.5 - \varepsilon')^3.$$

Proof. If x_1 will be killed from the left during J_2 then we will be done by the same reasoning as in part 2 above. Suppose that this does not happen.

Suppose that that x_2 is the later one among x_1 and x_2 to make a work period transition after t_2, at time v_2. Let v_3 be its next work period transition. Let u_2 be the last work period transition time of x_1 before v_2 and u_3, u_4 the next work period transitions of x_1. Let $\mathcal{J}_2(t_2)$ be the event that the coin tosses at time u_2, u_3 make x_1 active and the coin toss at time v_2 makes x_2 passive.

Suppose now that x_1 is the later one to make its first transition after t_2, at time u_3. Let u_4 be its next work period transition. Let v_3 be the last work period transition of x_2 before u_3 and v_4 its next work period transition. Let $\mathcal{J}_2(t_2)$ be the event that the coin tosses at time v_3, v_4 make x_2 passive and the coin toss at time u_3 makes x_1 active.

The randomization part of the Computation Property implies that, in the absence of damage, the probability of $\mathcal{J}_1(t_2) \cap \neg\mathcal{J}_2(t_2)$ in both cases is at most $1 - (0.5 - \varepsilon')^3$. Let us show that in both cases if $\mathcal{J}_1(t_2) \cap \mathcal{J}_2(t_2)$ holds then $\mathit{Level}(x_1, u_4) = s + 1$. i.e. the germ of x_1 will overrun the germ of x_2 while it is passive.

In the first case, suppose that the work period $[v_2, v_2]$ of x_2 covers the work period $[u_3, u_4]$ of x_1. Then x_1 certainly has sufficient time to overrun. If $[v_2, v_2]$ does not covers $[u_3, u_4]$ then it is divided between $[u_2, u_3]$ and $[u_3, u_4]$, therefore x_1 will have sufficient time in one of these work periods for the overrun. The reasoning is analogous in the other case.

Repeated application of the above reasoning gives the probability upper bound $(1 - (0.5 - \varepsilon')^3)^{n-1}$. Indeed, we reason about n disjoint windows (separated by the times $t_2, t_3, t_4, \dots$ where $t_{i+1} = t_i + d_1 T^\bullet$). If $\mathcal{J}_2(t_2)$ fails then $\mathcal{J}_1(t_3)$ still holds in the window between t_3 and t_4, so the reasoning is applicable to this new window, etc. The probability bounds will multiply due to the disjointness of the windows. □

Recall the definition of l in (18.1).

Lemma 18.8 (Birth)**.** *For integer $n > 1$, and $c_0 = (x_0, t_0)$, let*

$$\begin{aligned} d_2 &= 120 p_1^2 \lambda Q, \\ I_0 &= \Gamma(x_0, 33QB), \\ L_l &= \Gamma(x_0, 30QB), \\ G_0 &= [t_0 - Q\tau_1, t_0], \\ G_1 &= [t_0 - 3T^\bullet, t_0], \\ H_l &= [t_0, t_0 + nd_2 T^\bullet]. \end{aligned}$$

Let $\mathcal{G}_0$ be the event that $I_0 \times (G_0 \cup H_l)$ is damage-free, and $I_0 \times G_1$ is blue. Let $\mathcal{H}_l$ be the event that $L_l \times (G_0 \cup H_1)$ contains a big blue cell.

Assume

$$nd_2 T^\bullet < T_\bullet{}^*. \tag{18.3}$$

There is a constant $\kappa_1 > 0$ such that the probability of $\neg\mathcal{H}_l \cap \mathcal{G}_0$ is at most $le^{-\kappa_1 n}$. Moreover, this bound is also understood in the sense of an injective canonical simulation.

Proof. Repeated application of Lemma 18.7 (Level Increase). Notice that

$$\sum_{0\le s<l} d_1(s) = 3p_1 \sum_{0\le s<l} U(s) = 12p_1^2\lambda \sum_{0\le s<l} 2^s \le 24p_1^2\lambda 2^l < 120p_1^2\lambda Q = d_2.$$

For level $s \ge 0$, let

$$L_s = \Gamma(x_0, 6\cdot 2^s B),$$
$$t_s = t_0 + 24np_1^2\lambda T^\bullet 2^s.$$

Then $t_s = t_{s-1} + nd_1(s)T^\bullet$ for $s > 0$. Let $\mathcal{H}_s$ be the event that a cell of level s appears in L_s before time t_s. The blueness of I_0 implies $\mathcal{H}_0$. Repeated application of Lemma 18.7 (Level Increase) shows that the probability of $\neg\mathcal{H}_s \cap \mathcal{H}_i$ for $i < s$ is at most $e^{-\kappa_1 n}$ for an appropriate constant κ_1. From here, we obtain the required upper bound on the probability of non-birth. □

Proof of Lemma 10.2. Let η be a trajectory of a medium M_k of our amplifier and $\eta^* = \Phi_k^*(\eta)$. We will use the notation $Q = Q_k$, etc. Let us define some quantities with the absolute constants C_0, C_1, C_2, κ_1 that will be defined later.

$$D_1 = C_1 B,$$
$$D_2 = C_1 QB,$$
$$O_1 = C_2 T^{\bullet *},$$
$$q_1 = (24C_1Q + 3C_2U + 1)\varepsilon,$$
$$q_2 = 8(C_1Q)^2\sigma^2,$$
$$q_3 = e^{-\kappa_1 U/Q} C_1/33.$$
$$\sigma' = q_1 + q_2 + q_3.$$

Our goal is to show that for every time v_0, if η is (D_1, σ)-blue at time v_0, then η^* is (D_2, σ')-blue at time $v_0 + O_1$. Then we will be done since (10.3) implies $\sigma' \le \sigma_{k+1}$.

As mentioned at the beginning of Section 11, the simulation φ_k^* consists of two parts: the simulation of M_{k+1} by a medium M_k' of reach 2, and the simulation of M_k' by M_k as shown in Theorem 8.10.

1. We will concentrate on the self-organization from M_k' to M_{k+1}.
   The self-organization from M_k to M_k' is similar but much simpler, so we omit it (it refers to the proof of Theorem 8.10).

We will use the inequalities

$$T^{\bullet *}/T_\bullet{}^* \le 3, \tag{18.4}$$
$$T^{\bullet *}/T_\bullet < 2U, \tag{18.5}$$

where (18.4) is the same as (9.7) and (18.5) follows from (9.5). Let v_0 be a time, and let J_2 be a space interval of length D_2. Let

$$d_2 = 120p_1^2\lambda Q,$$
$$J_0 = \Gamma(J_2, D_2),$$
$$J_0' = \Gamma(J_2, D_2 - B),$$
$$G_0 = [v_0 - Q\tau_1, v_0],$$
$$G_1 = [v_0 - 5T^\bullet, v_0],$$
$$H_0 = [v_0, v_0 + O_1],$$
$$H_1 = [v_0, v_0 + T_\bullet{}^*].$$

Assume that η is (D_1, σ)-blue at time v_0. Let $\mathcal{E}_1$ be the event that there is no damage in $J_0' \times (G_0 \cup H_0)$. The Restoration Property gives the upper bound

$$8\frac{3D_2}{B}\frac{O_1 + Q\tau_1}{T_\bullet}\varepsilon \le (24C_1Q + 3C_2U + 1)\varepsilon \tag{18.6}$$

on the probability of $\neg\mathcal{E}_1$, where we used (18.4),(18.5).
Let $\mathcal{E}_2$ be the event that there is a subinterval J_1 of J_0 of size $2.5D_1$ such that $(J_0 \setminus J_1) \times G_1$ is blue.
2. The probability of $\mathcal{E}_1 \cap \neg\mathcal{E}_2$ is upper-bounded by

$$(D_2/B)^2\sigma_1^2 \le 8(C_1Q)^2\sigma_1^2. \tag{18.7}$$

Proof. The proof is a little similar to the proof of Lemma 8.4, only simpler. If

$$(iD_1/2 + [0, D_1-]) \cap J_0 \times G_1$$

is blue for all $i \in \mathbb{Z}$ then clearly $J_0 \times G_1$ is blue. For some i, j with $|i - j| \ge 4$, let $\mathcal{F}_{i,j}$ be the event that this set is not blue, for either i or j. The blueness of η at time v_0 implies the upper bound σ^2 on the probability of $\mathcal{F}_{i,j}$. The probability that there is an i, j such that $\mathcal{F}_{i,j}$ holds is at most $8(D_2/B)^2\sigma^2$. If there is no such pair i, j then it is easy to see that $\mathcal{E}_2$ holds.
Under condition of $\mathcal{E}_1 \cap \mathcal{E}_2$, let J_1 be the subinterval introduced in the definition of $\mathcal{E}_2$. Let

$$I_{0,i} = \Gamma(66QBi, 33QB).$$

Let $\mathcal{E}_3$ be the event that for all i such that $I_{0,i} \subset J_0' \setminus J_1$ the set $I_{0,i} \times (G_0 \cup H_1)$ contains a big blue cell. With $n < {T_\bullet}^*/(d_2T^\bullet)$, Lemma 18.8 (Birth) gives the upper bound

$$\log(5Q)e^{-\kappa_2 n}\frac{D_2}{33QB},$$

for an appropriate constant κ_2, on the probability of $\mathcal{E}_1 \cap \mathcal{E}_2 \cap \neg\mathcal{E}_3$. Using (18.4), (18.5) and (9.16), we can compute a constant κ_1 for upperbounding the above by

$$e^{-\kappa_1 U/Q}C_1/33. \tag{18.8}$$

We will show that under condition $\mathcal{E}_1 \cap \mathcal{E}_2 \cap \mathcal{E}_3$, the trajectory η^* is blue over

$$A_0 = J_2 \times [v_0 + O_1 - 5{T_\bullet}^*, v_0 + O_1].$$

3. Assume that J_1 is disjoint from $\Gamma(J_2, D_2/2)$.
3.1. $\Gamma(J_2, 2QB) \times H_0$ contains no non-blue cell.

Proof. Assume that, on the contrary, it contains such a cell. Let us build a path of ancestors from this cell. This path cannot end in a birth inside $\Gamma(J_2, D_2/2 - 2B)$ since Lemma 18.1 (Lasting Control) implies that no birth takes place there. Let us show that the path will not leave this set either. We will show that it does not leave on the right. For this, whenever we have a choice between father and mother, we build the path to the left. This way, the path only moves to the right when the cell is obtained from its mother by growth or end-healing. Once the backward path moved into the originating colony of a growth it will stay there at least until the beginning of the work period before it can move right again. It needs therefore at least $(i-1){T_\bullet}^*$ time units to move over i colonies, and its total displacement will be at most $QB(1+O_1/{T_\bullet}^*)$. For this to be smaller than the distance of the complement of $\Gamma(J_2, D_2/2-2B)$ from $\Gamma(J_2, 2QB)$, we need

$$QB(1 + O_1/{T_\bullet}^*) < D_2/2 - 2QB - 2B.$$

Rearranging and using $T^{\bullet *}/T_{\bullet}{}^{*} \leq 3$ from (9.7) show that this is satisfied if

$$2(4 + 3C_2) < C_1. \tag{18.9}$$

Therefore the path never leaves $\Gamma(J_2, D_2/2 - 2B)$: this is impossible since the path is not blue but the set it is in is blue at time v_0.

Let $x \in J_2$: we will show that x becomes controlled in η^* at some time before $v_0 + O_1 - 5T_{\bullet}{}^{*}$. Then Lemma 18.1 will imply that it stays controlled until $v_0 + O_1$ and part 3.1 above shows that it is actually blue.

Let i be such that $x \in I_{0,i}$. Event $\mathcal{E}_3$ implies that $I_{0,i}(G_0 \cup H_1)$ contains a big blue cell (y, t). This cell is within distance $66QB$ from x, say, to the left. As soon as it arises it begins to create a right neighbor, which also creates a right neighbor, etc., until the chain either reaches x or will be prevented by another blue cell z within distance QB on the right. In the latter case, we continue from z, and see that the chain controls x by time $t + 67T^{\bullet *}$: hence we are done if

$$C_2 \geq 73.$$

4. Assume that J_1 intersects $\Gamma(J_2, D_2/2)$.

4.1. There are no non-germ non-blue cells in

$$\Gamma(J_2, 2QB) \times [v_0 + Q\tau_1, v_0 + O_1].$$

Proof. The size of J_1 is $2.5D_1 < QB$ according to our assumption about Q. Therefore if it has any non-blue non-germ cells at time v_0 then these cells are in domains with an exposed edge and will decay within $Q\tau_1$ time units. It can be seen just as in part 3.1 above that non-blue non-germ edges don't have time to grow in from outside $\Gamma(J_2, D_2)$.

Non-blue germ cells in J_1 may also begin to grow to the right. But within 5 cell widths, they meet an existing blue cell x, as seen in part 1 of the proof of Lemma 18.1 (Lasting Control). x will not be erased by a non-blue germ cell on the left. It will not be erased by a germ cell on the right either since germ cells erase other germ cells only in the right direction. It might be erased by a non-germ cell y on the right, that is about to create a left adjacent neighbor to itself. Cell y (or, even the neighbor $y - B$ it creates) will then be the next obstacle and it can only be eliminated in a similar way. The time between these successive eliminations (except possibly between the first two) is at least $T_{\bullet}{}^{*}$ since for the next occurrence of such an event, we must wait for the colony in question to die and for a new colony to overtake and start growing a left extension. Therefore the total number of cells that can be added to the non-blue germ in this exotic way is at most $O_1/T_{\bullet}{}^{*} + 2 < 3C_2 + 2$ (using $T^{\bullet *}T_{\bullet}{}^{*} < 3$ again), which still leaves it a germ if

$$C_0 > 3C_2 + 2. \tag{18.10}$$

Let $x \in J_2$: we will show as in 4 above that x becomes controlled in η^* at some time before $v_0 + O_1 - 5T_{\bullet}{}^{*}$.

Let $I_{0,i}$ be closest possible to x while disjoint from that J_1. Event $\mathcal{E}_3$ implies that $I_{0,i}(G_0 \cup H_1)$ contains a big blue cell (y, t). This cell is within distance

$$(1.25) \cdot D_1 + 2 \cdot 66QB = 1.25C_1B + 132QB$$

from x, say, to the left. As soon as it arises it begins to create a right neighbor, which also creates a right neighbor, etc., until the chain either reaches x or will be prevented by another blue cell z

within distance QB on the right. In the latter case, we continue from z, and see that the chain controls x by time $t + 133T^{\bullet *}$: hence we are done if

$$C_2 \geq 139.$$

We have shown that the probability that η^* is not blue over A_0 is at most σ'. We bounded it, referring only to events within a window with space projection J_0. It follows that for a group of m disjoint space translations of this window, the probability bounds that η^* is not blue over any of the corresponding translations of A_0, is bounded by σ'^m. □

19. Some applications and open problems

19.1. **Non-periodic Gibbs states.** Consider spin systems in the usual sense (generalizations of the Ising model). All 2-dimensional spin systems hitherto known were known to have only a finite number of extremal Gibbs states (see e.g. [1]): thus, theoretically, the amount of storable information on an $n \times n$ square lattice did not grow with the size of the lattice. In 3 dimensions this is not true anymore, since we can stack independent 2-dimensional planes: thus, in a cube C_n of size n, we can store n bits of information. More precisely, the information content of C_n can be measured by the dimension of the set of vectors

$$(\mu\{\, \sigma(x) = 1 \,\} : x \in C_n)$$

where μ runs through the set of Gibbs states. This dimension can be at most $O(n^{d-1})$ in a d-dimensional lattice, since the Gibbs state on a cube is determined by the distribution on its boundary. The stacking construction shows that storing $\Omega(n^{d-2})$ bits of information is easy. We can show that $\Omega(n^{d-1})$ is achievable: in particular, it is possible to store an infinite sequence in a 2-dimensional spin system in such a way that n bits of it are recoverable from any n sites with different x coordinates.

For this, we apply a transformation from [15] (see also [3]): a probabilistic cellular automaton M in d dimensions gives rise to an equilibrium system M' in $(d+1)$ dimensions. Essentially, the logarithms of the local transition probabilities define the function J and space-time configurations of M become the space-configurations of M'. Non-ergodicity of M corresponds to phase transition in M'. In the cellular automaton of Theorem 5.6, each infinite sequence ϱ gives rise to a space-time configuration storing the bits of ϱ in consecutive cells. Now, each of these space-time configurations gives rise to a separate Gibbs state belonging to one and the same potential.

Let us note that though the Gibbs system is defined in terms of an energy function $H(\sigma)$, it is not helpful to represent this energy function in terms of a temperature T as $H(\sigma)/T$. The reason is that the individual terms of $H(\sigma)$ do not depend linearly on the error probability ε (or any function of it). In fact, we believe it can be proved that if an artificial T is introduced (with $T = 1$ for a certain sufficiently small value of ε) then a slight decrease of T destroys the phase transition.

19.2. **Some open problems.**

19.2.1. *Turing machines.* It is an interesting question (asked by Manuel Blum) whether a reliable Turing machine can be built if the tape is left undisturbed, only the internal state is subject to faults. A construction similar to reliable cellular automaton seems to be possible. It is not known whether there is a simpler construction or, whether there is a way to derive such a machine from reliable cellular automata.

19.2.2. *Relaxation time as a function of space size.* Consider now Toom's medium as a typical example of a medium non-ergodic for $m = \infty$. It follows from Toom's proof that, for small enough fault probability ε, we have $\lim_{m\to\infty} r_m(0, 1/3) = \infty$, i.e. the increase of space increases the relaxation time (the length of time for which the rule keeps information) unboundedly. The speed of this increase is interesting since it shows the durability of information as a function of the size of the cellular automaton in which it is stored. Toom's original proof gives only $r_m(0, 1/3) > cm$ for some constant c. The proof in [5], improving on [14], gives $r_m(0, 1/3) > e^{cm}$ for some constant c, and this is essentially the meaning of saying that Toom's rule helps remember a bit of information for exponential time.

So far, the relaxation times of all known nontrivial non-ergodic media (besides Toom's, the ones in [10] and [11]) depend exponentially on the size of the space. It is an interesting question whether this is necessary. In a later work, we hope to show that this is not the case and that there are non-ergodic media such that $r_m(n, 1/3) < m^c$ holds for all n, for some constant c. The main idea is

that since the medium will be able to perform an arbitrary reliable computation this computation may involve recognizing the finiteness of the space rather early (in time m^c) and then erasing all information.

19.2.3. *Relaxation time as a function of observed area.* Lemma 5.5 seems to suggest that the issue of information loss in a cellular automaton is solved by the question of mixing for $m = \infty$. This is not so, however: as noted together with Larry Gray, Leonid Levin and Kati Marton, we must also take the dependence of $r_m(n, \delta)$ on n into account. As time increases we may be willing to use more and more cells to retrieve the original information. Even if $r_\infty(n, \delta) < \infty$ for each m, we may be satisfied with the information-keeping capability of the medium if, say, $r_m(n, 1.9) > e^{cn}$ for some constant c.

In some mixing systems, the dependence of $r_m(n, \delta)$ on n is known.

Example 19.1 (The contact process.). The contact process is a one-dimensional *CCA* as defined in Subsection 2.4. Let us note that this this process is not noisy: not all local transition rates are positive. The process has states 0,1. In trajectory $\eta(x, t)$, state $\eta(x, t) = 1$ turns into 0 with rate 1. State $\eta(x, t) = 0$ turns into 1 with rate $\lambda(\eta(x - 1, t) + \eta(x + 1, t))$. It is known that this process has a critical rate $\lambda_c \in [0+, \infty-]$ with the following properties.

If $\lambda \le \lambda_c$ then the process is mixing with the invariant measure concentrated on the configuration ξ with $\xi(x) = 0$ for all x. If $\lambda > \lambda_c$ then the process is non-ergodic.

If $\lambda < \lambda_c$ then it is known (see Theorem 3.4 in Chapter VI of [20]) that the order of magnitude of $r_\infty(n, \delta)$ is $\log n$.

If $\lambda = \lambda_c$ then the convergence is much slower, with an order of magnitude that is a power of n (see Theorem 3.10 in Chapter VI of [20] and [6]). ♢

We believe it possible to construct a medium that is mixing for $m = \infty$ but loses information arbitrary slowly: for any computable function $f(n)$, and a constant c there is a medium with

$$r_m(n, 1.9) > f(n) \wedge e^{cm}$$

for finite or infinite m. Notice that this includes functions $f(n)$ like $e^{e^{e^n}}$. Such a result could be viewed as an argument against the relevance of the non-ergodicity of the infinite medium for practical information conservation in a finite medium. The construction could be based on the ability to perform arbitrary computation reliably and therefore also to destroy locally identifiable information arbitrarily slowly.

References

1. Michael Aizenman, *Translation invariance and instability of phase coexistence in the two-dimensional Ising system*, Comm. Math. Phys **73** (1980), no. 1, 83–94.
2. Charles H. Bennett, G. Grinstein, Yu He, C. Jayaprakash, and David Mukamel, *Stability of temporally periodic states of classical many-body systems*, Physical Review A **41** (1990), 1932–1935.
3. Charles H. Bennett and Geoffrey Grinstein, *Role of irreversibility in stabilizing complex and nonergodic behavior in locally interacting discrete systems*, Physical Review Letters **55** (1985), 657–660.
4. E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning ways for your mathematical plays*, Academic Press, New York, 1982.
5. Piotr Berman and Janos Simon, *Investigations of fault-tolerant networks of computers*, Proc. of the 20-th Annual ACM Symp. on the Theory of Computing, 1988, pp. 66–77.
6. C. Bezuidenhout and G. Grimmett, *The critical contact process dies out*, Annals of Probability **18** (1990), 1462–1482.
7. R. E. Blahut, *Theory and practice of error-control codes*, Addison-Wesley, Reading, MA, 1983.
8. Paula Gonzaga de Sá and Christian Maes, *The Gács-Kurdyumov-Levin Automaton revisited*, Journal of Statistical Physics **67** (1992), no. 3/4, 607–622.
9. R. L. Dobrushin and S. I. Ortyukov, *Upper bound on the redundancy of self-correcting arrangements of unreliable elements*, Problems of Information Transmission **13** (1977), no. 3, 201–208.
10. Peter Gács, *Reliable computation with cellular automata*, Journal of Computer System Science **32** (1986), no. 1, 15–78.
11. ———, *Self-correcting two-dimensional arrays*, Randomness in Computation (Greenwich, Conn.) (Silvio Micali, ed.), Advances in Computing Research (a scientific annual), vol. 5, JAI Press, Greenwich, Conn., 1989, pp. 223–326.
12. ———, *Deterministic parallel computations whose history is independent of the order of updating*, Tech. report, Boston University, Department of Computer Science, Boston, MA 02215, 1995.
13. Peter Gács, Georgii L. Kurdyumov, and Leonid A. Levin, *One-dimensional homogenuous media dissolving finite islands*, Problems of Inf. Transm. **14** (1978), no. 3, 92–96.
14. Peter Gács and John Reif, *A simple three-dimensional real-time reliable cellular array*, Journal of Computer and System Sciences **36** (1988), no. 2, 125–147.
15. Sheldon Goldstein, Roelof Kuik, Joel L. Lebowitz, and Christian Maes, *From PCA's to equilibrium systems and back*, Commun. Math. Phys. **125** (1989), 71–79.
16. Lawrence F. Gray, *The positive rates problem for attractive nearest neighbor spin systems on* $\mathbf{Z}$, Z. Wahrscheinlichkeitstheorie verw. Gebiete **61** (1982), 389–404.
17. ———, *The behavior of processes with statistical mechanical properties*, Percolation Theory and Ergodic Theory of Infinite Particle Systems, Springer-Verlag, 1987, pp. 131–167.
18. Gene Itkis and Leonid Levin, *Fast and lean self-stabilizing asynchronous protocols*, Proc. of the IEEE Symp. on Foundations of Computer Science, 1994, pp. 226–239.
19. G. L. Kurdyumov, *An example of a nonergodic homogenous one-dimensional random medium with positive transition probabilities*, Soviet Mathematical Doklady **19** (1978), no. 1, 211–214.
20. Thomas M. Liggett, *Interacting particle systems*, Grundlehren der mathematischen Wissenschaften, vol. 276, Springer Verlag, New York, 1985.
21. Jacques Neveu, *Bases mathematiques du calcul des probabilités*, Masson et Cie, Paris, 1964.
22. Kihong Park, *Ergodicity and mixing rate of one-dimensional cellular automata*, Ph.D. thesis, Boston University, Boston, MA 02215, 1996.
23. Nicholas Pippenger, *On networks of noisy gates*, Proc. of the 26-th IEEE FOCS Symposium, 1985, pp. 30–38.
24. Charles Radin, *Global order from local sources*, Bull. Amer. Math. Soc. **25** (1991), 335–364.
25. Daniel A. Spielman, *Highly fault-tolerant parallel computation*, Proc. of the 37th IEEE FOCS Symposium, 1996, pp. 154–163.
26. Tommaso Toffoli and Norman Margolus, *Cellular automata machines*, (Cambridge, MA.), MIT Press, Cambridge, MA., 1987.
27. Andrei L. Toom, *Stable and attractive trajectories in multicomponent systems*, Multicomponent Systems (New York) (R. L. Dobrushin, ed.), Advances in Probability, vol. 6, Dekker, New York, 1980, Translation from Russian, pp. 549–575.
28. B. S. Tsirel'son, *Reliable information storage in a system of locally interacting unreliable elements*, Interacting Markov Processes in Biology (Pushchino) (V. I. Kryukov R. L. Dobrushin and A. L. Toom, eds.), Scientific Centre of Biological Research, Pushchino, 1977, In Russian. Translation by Springer., pp. 24–38.

29. John von Neumann, *Probabilistic logics and the synthesis of reliable organisms from unreliable components*, Automata Studies (Princeton, NJ.) (C. Shannon and McCarthy, eds.), Princeton University Press, Princeton, NJ., 1956.
30. Weiguo Wang, *An asynchronous two-dimensional self-correcting cellular automaton*, Ph.D. thesis, Boston University, Boston, MA 02215, 1990, Short version: *Proc. 32nd IEEE Symposium on the Foundations of Computer Science*, 1991.

Index

Computer Science Department, Boston University
E-mail address: gacs@bu.edu