Reliable Cellular Automata with Self-Organization

Peter Gács *
Computer Science Department
Boston University
gacs@bu.edu

January 24, 2024 (last printing)

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.

This is a corrected and strengthened version of the journal article [17].

Contents

1   Introduction

A *cellular automaton* is a homogenous array of identical, locally communicating finite-state automata. Traditionally, the model is called *interacting particle system* when time is continuous. By choosing the transition function appropriately, a cellular automaton can perform an arbitrary computation. Indeed, for any one-tape Turing machine one can construct a one-dimensional cellular automaton simulating it step-for-step.

Fault-tolerant information storage and computation 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 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 if the computers are sufficiently large, 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"[1] techniques are needed in conjunction with traditional error-correction.

1.1 Historical remarks

The problem of reliable computation with unreliable components was addressed in [37] in the context of Boolean circuits. Von Neumann's solution, as well as its improved versions in [12] and [31], rely on high connectivity and non-uniform constructs. The best currently known result of this type is in [33] 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, for example, two distinguished initial configurations:

$$\xi_0, \xi_1$$

in which all cells have state 0 and 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 *ergodic* if it eventually forgets all information about its initial configuration. Finite noisy cellular automata are always ergodic. 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 ergodic 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-ergodic.

The difficulty in constructing non-ergodic 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

---

[1]In distributed computing, "self-stabilization" refers to techniques of restoring some structure from an arbitrarily corrupted state; however, no new faults are assumed to occur during restoration.

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 an ergodic transition function, even in two dimensions, if the "failure" probabilities are not symmetric with respect to the interchange of 0's and 1's; it has not been proved to be non-ergodic even in the symmetric case. Perturbations of the one-dimensional majority voting function were actually shown to be ergodic in [21] and [22].

Noisy cellular automata remembering a bit forever (in the sense defined above) in dimensions 2 and higher were constructed in [35]. The paper [19] recognizes that adding one more dimension, Toom's idea can be used not just for remembering a bit but for simulating reliably an arbitrary computation. It designs a simple three-dimensional fault-tolerant cellular automaton that simulates arbitrary one-dimensional arrays. The theorem will be spelled out precisely in Section 6.2. Toom's original proof was simplified and adapted to strengthen these results in [5] (see also [16]).

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 [18] (see also [11]). It is now known (see [30]) that perturbation (at least, in a strongly biased way) makes this automaton ergodic.

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 [36] 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 [13] constructs a non-ergodic one-dimensional cellular automaton working in discrete time, using some ideas from the very informal paper [24] 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 [14] 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", that is 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.

1.3   New features

Asynchrony   In the three-dimensional fault-tolerant cellular automaton of [19], 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 in each moment, 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 [38]. Experiments combining this technique with the error-correction mechanism of [19] 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 [26].

Self-organization   Most hierarchical constructions, including our earlier ones, start from a complex, hierarchical initial configuration (in case of an infinite system, and infinite hierarchy). The present paper avoids this: its transi-

tion function increases the height of the hierarchy as the computation length requires it.

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. For example, 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 Freeze 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.

1.4   Overview of the paper

- Section 2 defines probabilistic cellular automata.

- Section 3 and spells out the main theorems for discrete time and infinite space.

- Section 4 introduces block codes and block simulations using colonies, and also generalized cellular automata (called *abstract media*), allowing more general kinds of simulation.

- Section 5 defines hierarchical codes and a hierarchy of simulations (called *amplifier*). 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 extends the main discrete-time theorems to the case of finite space.

- Section 7 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 8 shows the method we will use to simulate a discrete-time cellular automaton by a variable-period one.

- Section 9 develops some simple simulations, to be used either directly or as a paradigm. The example transition function defined here will correct any set of faults in which no two faults 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 called *robust media* for which nontrivial fault-tolerant simulations exist will be defined in Section 10. In these, cells are not necessarily adjacent to each other. The transition function can erase as well as create cells.

  The set of space-time points with "bad" values is called the "damage". The Restoration Property in Condition 10.4 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.

  As a history η of medium M_1 simulates a history η^* of a medium M_2, we define the damage of η^* in terms of that of η essentially as follows. Damage occurs 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 $\approx \varepsilon^2$.

- Section 11 defines the kind of amplifiers to be built. The *main lemma*, called the Amplifier Lemma, says that these amplifiers exist. The rest of the section applies the main lemma to the proof of one of the main theorems.

- Section 12 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 9. 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, that is predictability in the simulated configuration is restored.

- Section 13 defines the kind of local consistency needed for a colony to function.

- Section 14 gives the rules for killing, creation, growth, including the growth of germs which are precursors of colonies. It proves the basic lemmas about space-time paths connecting live cells. It also defines the healing rule; due to the need to restore some local clock values consistently with the neighbors, this rule is somewhat elaborate.

- Section 15 defines the decay rule and shows that a large gap will eat up a whole colony.

- Section 16 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 relies on Section 15.3 showing that if a gap will not be healed promptly then it grows.

- Section 17 proves the Healing Lemma, showing how the effect of a small amount of damage will be corrected.

- Section 18 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 19 introduces and uses the error-correcting computation rules.

- Section 20 proves the main properties of the reliable simulation defined in the preceding sections.

- Section 21 shows how the germ-growth rules lead to self-organization, and proves the main theorems that use self-organization.

- The concluding remarks in Section 22 hint at some applications and questions.

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.

2 Cellular automata

In the introductory sections, we confine ourselves to one-dimensional infinite cellular automata. Let us introduce some notation to be used throughout.

Notation 2.1 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 will use the notation

$$f(n) \overset{*}{<} g(n) \tag{2.1}$$

for what usually is written as $f(n) = O(g(n))$: that is for the fact that $f(n) \leq cg(n)$ for some constant c. We write $f(n) \overset{*}{=} g(n)$ if $f(n) \overset{*}{<} g(n)$ and $g(n) \leq f(n)$.

The standard mathematical notation for open intervals and for pairs is the same; we hope that the context will make the meaning always clear. We will use the same notation for intervals of integers as for those of real numbers: 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

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

and $\overline{A}$ for its closure: for example, $\overline{(a, b]} = [a, b]$. Given two space-time sets A, B, we denote

$$A + B = \{a + b : a \in A, b \in B\}. \tag{2.2}$$

Some lists of assertions are denoted by (a), (b), ... and some by (1), (2), ... The attempted convention is that for a list properties that all hold or are required (conjunction) 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 $\log_2$. For two strings u, v, we will denote by

$$u \sqcup v \tag{2.3}$$

their concatenation. ⌟

2.1 Deterministic cellular automata

Let us give here the most frequently used definition of cellular automata.
Later, we will use a certain generalization. First, the notions associated without considering time.

Definition 2.2 (Space and states) The set Λ 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

$$\boldsymbol{\pi}_{\mathrm{s}} E, \boldsymbol{\pi}_{\mathrm{t}} E \tag{2.4}$$

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

$$\xi(x)$$

for $x \in \Lambda$. Here, $\xi(x)$ is the state of site x. ⌟

Now, the notations needed for reasoning about the evolution of configurations. For uniform treatment, it is useful to view even discrete-time cellular
automata as working in continuous time, but of course they will change their
states only at certain predetermined instants.

Definition 2.3 (Space-time) The time of work of our cellular automata will
be the interval $[0, \infty)$. Our *space-time* is given by

$$\Lambda \times [0, \infty).$$

A *history* is a space-time function $\eta(x, t)$ which for each t defines a configuration. If in a history 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
histories 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

$$\eta(c, a-). \tag{2.5}$$

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

$$\eta(V)$$

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

Cellular automata describe a kind of dynamic for evolutions.

Definition 2.4 (Deterministic cellular automata) A *deterministic cellular automaton*

$$CA(Tr, \Lambda).$$

is determined by a transition function $Tr : \mathbb{S}^3 \to \mathbb{S}$ and the set Λ of sites. We will omit Λ from the notation when it is obvious from the context. A history 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 history η let us write

$$\overline{Tr}(\eta, x, t) = Tr(\eta(x-1, t-1), \eta(x, t-1), \eta(x+1, t-1)) \tag{2.6}$$

for the "intended" value of $\eta(x, t)$. ⌟

Given a configuration ξ over the space Λ 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 history 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.

Definition 2.5 (Capacity) We will 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 $Cap = \|\mathbb{S}\|$ 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.

Definition 2.6 (Fields) Nonempty 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. The field consisting of the bits $s(i), s(i+1), \ldots, s(j-1)$ of the state will be denoted

$$F[i{:}j]. \tag{2.7}$$

The array obtained from joining the same fields (say the Mail field) of all the different cells is called a *track*, (say the Mail track). (The terminology recalls tracks of a magnetic tape.) We will generally define fields that are either disjoint or contained in each other (but there may be some exceptions.) We will call the number of bits $|F|$ in a field its *width*, and and the proportion $|F|/Cap$ its *relative width*. The width and relative width of a *track* is defined to be the same as that of the corresponding field. ⌟

Example 2.7 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. 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 $\eta(x-1, t-1)$ of the left neighbor is the right-directed mail field $\eta(x,t).\mathsf{Mail}_{-1}$. This field, on the other hand, depends only on the right-directed mail field of the left neighbor and the workspace field $\eta(x, t-1).\mathsf{Work}$. 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. ⌐

The following concepts related to the transition function will play a role.

Definition 2.8 (Legality) Given a transition function Tr, we say that state s' is a *legal successor* of state s if there are states r, t with $s' = Tr(r, s, t)$. We define

$$legal(s, s') = 1$$

if this holds and 0 if it does not. ⌐

2.3   Probabilistic cellular automata

In probabilistic cellular automata, the transitions allow randomness.

Definition 2.9 A *random history* 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 history for every value of $\omega \in \Omega$. We will generally omit ω from the arguments of η. When we omit the mention of μ we will use P 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

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

⌐

Notation 2.10 We will denote the expected value of f with respect to μ by

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

where we will omit μ when it is clear from the context. ⌐

Definition 2.11 (Events) A function $f(\eta)$ with values 0, 1 (that is an indicator function) and measurable in the σ-algebra $\mathcal{A}$ 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 $s \in \mathbb{S}$, $(x, t_i) \in W$. ⌟

Sometimes, we may want to refer to events not necessarily expressible by the history η; still, we need a sense in which they become knowable over time. The notion of filtration captures this.

Definition 2.12 (Filtration) For all times t, we assume the existence of a σ-algebra

$$\mathcal{A}_t \supseteq \mathcal{A}(\Lambda \times [0, t]), \quad \mathcal{A}_{<t} = \bigcup_{u < t} \mathcal{A}_u.$$

The system $\{\mathcal{A}_t\}$ is required to be increasing:

$$t < u \Rightarrow \mathcal{A}_t \subseteq \mathcal{A}_u.$$

The function $t \mapsto \mathcal{A}_t$ is our *filtration*. ⌟

Example 2.13 The system $t \mapsto \mathcal{A}(\Lambda \times [0, t])$ is a filtration. ⌟

Transition matrices will play the role of transition functions.

Definition 2.14 (Probabilistic cellular automaton) A *transition matrix* $\mathbf{P}(s, (r_{-1}, r_0, r_1))$ is a function $\mathbf{P} : \mathbb{S}^4 \to [0, 1]$ with the property $\sum_s \mathbf{P}(s, \mathbf{r}) = 1$. For an arbitrary history η, and space-time point (x, t), denote

$$\overline{\mathbf{P}}(\eta, s, x, t) = \mathbf{P}(s, (\eta(x - 1, t), \eta(x, t), \eta(x + 1, t))). \qquad (2.8)$$

We will omit the parameter η when it is clear from the context.

A *probabilistic cellular automaton*

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

is characterized by saying which random histories are considered *trajectories*. Now a trajectory is not a single history (sample path) but a distribution over histories that satisfies the following condition, saying that the random history η is a trajectory if and only if the following holds. Let $s_1, \dots, s_n \in \mathbb{S}$.

$$\mathsf{P} \left(\bigcap_{i=1}^n \{\eta(x_i, t) = s_i\} \,\Big|\, \mathcal{A}_{t-1} \right) = \prod_{i=1}^n \overline{\mathbf{P}}(\eta, s_i, x_i, t - 1).$$

This expression uses the notion of conditional probability over sigma-algebras. For a more elementary statement of the same property, let $x_0, \dots, x_{n+1}$ be

given with $x_{i+1} = x_i + 1$. Let us fix an arbitrary history ζ and an arbitrary event $\mathcal{H} \ni \zeta$ in $\mathcal{A}_{t-1}$. Then we require

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

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 2.15 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 *finite Markov chain*. We get a finite Markov chain 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. A systematic theory of such systems and an overview of many results available in 1985 can be found in [26]. Here, we indicate an elementary construction similar to the one in [21].

Definition 2.16 (Transition rate matrix) Let us call a *transition rate matrix* a function $\mathbf{R} : \mathbb{S}^4 \to [0, \infty)$, written as

$$
\mathbf{R}(s, \mathbf{r}) \geq 0,
$$

with the normalization property $\mathbf{R}(r_0, (r_{-1}, r_0, r_1)) = 0$ for all $\mathbf{r}$. Its elements are called the *transition rates*. ⌟

We will obtain the continous-time process as the limit of certain discrete processes:

Definition 2.17 Consider a generalization

$$PCA(\mathbf{P}, B, \delta, \Lambda)$$

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 $0, \delta, 2\delta, 3\delta, \ldots$ for some small positive δ. ⌟

The body size parameter B will make more sense later, in the context of simulations. The parameter δ is of more interest now, as the limit $\delta \to 0$ will be considered. Fixing $\Lambda, \mathbf{P}$, let

$$M_\delta = PCA(\mathbf{P}, 1, \delta, \Lambda)$$

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. (The definition is sound when δ is small enough to make the last expression nonnegative.) It can be shown that with any fixed initial configuration $\eta(\cdot, 0)$, the distributions of trajectories η_δ of M_δ will converge to a certain random process η which is the continuous-time probabilistic cellular automaton with these rates. For the sense of convergence and other constructions of the same process, see [26], [21] and the works quoted there.

Definition 2.18 (Interacting particle system) The random process defined by the above procedure will be denoted by

$$CCA(\mathbf{R}, \Lambda),$$

and called the *(continuous-time) interacting particle system* with the given rate matrix. We call this system *noisy* if $\mathbf{R}(s, \mathbf{r}) > 0$ for all $s \neq r_0$. ⌟

It can be shown that the process defined this way is a Markov process, that is 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 more general definition allowing simultaneous change in a finite number of sites, see [26].)

2.5 Perturbation

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. Formally, we introduce perturbed versions of our cellular automata.

Discrete time The definition is somewhat more straightforward in the discrete-time case, since then only the transition function (or probability matrix) will be perturbed.

Definition 2.19 (Random perturbation of a deterministic system) We will say that a random history (μ, η) is a *trajectory* of the ε-*perturbation*

$$CA_\varepsilon(Tr, \Lambda)$$

of the transition function Tr if the following holds. For all $x_0, \ldots x_{n+1}, t$ with $x_{i+1} = x_i + 1$ for all $0 < i_1 < \cdots < i_k < n$,

$$\mu\Big(\bigcap_{j=1}^{k} \{\eta(x_{i_j}, t) \neq \overline{Tr}(\eta, x_{i_j}, t-1)\} \,\Big|\, \mathcal{A}_{t-1}\Big) \leq \varepsilon^k.$$

Here is the same property without using conditional probability over sigma-algebras: for events $\mathcal{H}$ in $\mathcal{A}_{<(t-1)}$ with $\mu(\mathcal{H}) > 0$,

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

⌟

Note that the model $CA_\varepsilon(Tr, \Lambda)$ defined this way is not a probabilistic cellular automaton, since even if the initial configuration is fixed there are many random processes that are accepted as its trajectories. Given any probabilistic cellular automaton $PCA(\mathbf{P}, \Lambda)$ such that $\mathbf{P}(s, \mathbf{r}, \Lambda) \geq 1 - \varepsilon$ whenever $s = Tr(\mathbf{r})$, the trajectories of this are accepted as trajectories of $CA_\varepsilon(Tr, \Lambda)$; however, these do not exhaust all 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.

This new freedom becomes important in our later construction.

Continuous time There are several choices of how to generalize perturbation to continous time.

Definition 2.20 (Random perturbation of a continous-time system) 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 the perturbed process is again a continuous-time interacting particle system, just with changed parameters. Other perturbations are imaginable, but not worth the trouble defining formally now.

3 Some results

We state some of the main results in this section; but a more detailed and formalized description of them will be given in Sections 6 and 7. For simplicity the theorems here are only for infinite space. A noisy cellular automaton on a finite space is ergodic, so eventually it forgets all about its initial state. However, as will be shown in the later sections, even in finite spaces the "relaxation time", that is the time before which information as well as computation results of the reliable cellular automaton can be trusted, grows (almost) exponentially as a function of the space size.

3.1 Information storage

Remembering a few bits The simplest task we may want to assign a cellular automaton is to store some information.

Definition 3.1 (Remembering some bits) Suppose that the bit string that is a local state has some field F (it can for example be the first two bits of the state). We will say that *Tr remembers* field F over an infinite set of sites Λ 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 all trajectories (μ, η) of the ε-perturbation $CA_\varepsilon(Tr, \Lambda)$ with $\eta(\cdot, 0) = \xi_s$, for all x, t we have

$$\mu\{\eta(x, t).F \neq s\} < f(t) + O(\varepsilon) \text{ where } \lim_{t \to \infty} f(t) = 0. \tag{3.1}$$

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

Let us call a configuration ξ *homogeneous* if there is a state $q \in \mathbb{S}$ such that $\xi(x) = q$ for all x. We say that *Tr* remembers F in a *self-organizing way* if the initial configurations ξ_s can be made homogeneous. ⌟

For some $c > 1$ let

$$h_0(N, c) = c^{(\log N)^{1/2}}. \tag{3.2}$$

Theorem 3.1 (Remembering a field in discrete time) *For any constant $c_1 > 1$ there is a one-dimensional probabilistic transition matrix that remembers a field, in a self-organizing way; the function $f(t)$ in (3.1) can be chosen as t^{-c_1}.*

If the space is finite with size N then in any cell, at time t, the probability of forgetting it at time t is bounded by $\varepsilon^{h_0(N,c_2)}$ for an appropriate constant c_2.

Thus for a finite space, the memory lasts, even if not for exponential time, but for a time exponential in $h_0(N, c_2)$.

Remark 3.2 *The error term in* (3.1) *improves with time, but why does it not decrease with ε? Technically, in our proofs it is the price of the need for symmetry-breaking random choices in self-organization, but it still seems improvable.* ⌟

In the proof of this theorem in [13] the initial configuration ξ_s has an infinite hierarchical structure, so the remembering was not self-organizing. In Theorem 3.1 this is not necessary anymore. We achieve the simplification of the result by a technique we call *self-organization*: the hierarchy will be built up by the probabilistic transition during computation. The following continuous-time version of the same result is also new:

Theorem 3.2 (Remembering a field in continuous time) *There is a one-dimensional transition-rate matrix that remembers a field in a self-organizing way.*

Remembering a long (possibly infinite) string In order to store information reliably in a computing device that deals with its bits individually, it is necessary to add redundancy; otherwise some bits can be lost in the very first step. So in order to store a string the initial configuration of the cellular automaton will contain it encoded by some error-correcting code (φ_*, φ^*) (the notation will be motivated in Section 4). A finite or infinite string ρ in some alphabet Σ will be encoded into a string $\varphi_*(\rho)$ in another alphabet $\mathbb{S}$ of symbols, cell states of some cellular automaton. These states have fields, and one of them can be called *Info*. The code we will use is such that if $\xi = \varphi_*(\rho)$ then for position n in ρ we have $\xi(x).\textit{Info} = \rho(x)$: the original word is in the code explicitly in the *Info* field, while the other fields serve for error checks and other structure. Though it will take some time to describe our codes (in the rest of the paper) as they have a hierarchical structure, in fact they are easy to compute: their computational complexity is low. Here is a theorem about remembering forever an infinite string:

Theorem 3.3 (Remembering a long string) *Given some alphabet Σ there is a one-dimensional deterministic cellular automaton $CA(Tr)$ with a field F, a code $\psi_* : \Sigma^{\mathbb{Z}_N} \to \mathbb{S}^{\mathbb{Z}_N}$ for finite or infinite N and a constant $c > 1$ such that for sufficiently small ε, if η is the trajectory of an ε-perturbation $CA_\varepsilon(Tr)$*

with $\eta(\cdot, 0) = \psi_*(\rho)$ then for all x, t we have

$$\mathsf{P}\{\eta(x, t).\mathsf{F} \neq \rho(x)\} = O(\varepsilon) + \varepsilon^{h_0(N,c)} t.$$

As we see this code turns our cellular automaton an information-transmission device from the present to the future, with *constant capacity*: storing one symbol per cell. (Some non-*Info* fields of the cells will be used for error-checks.)

3.2 Computation

In order to accommodate computations that last forever, and to formalize the idea of "more and more output", let us introduce a special alphabet:

Definition 3.3 (Standard alphabet) Let us call the set

$$\Sigma_0 = \{0, 1, \#, *\} \tag{3.3}$$

the *standard alphabet*. ⌐

Symbol $\#$ will be used to delimit binary strings, and $*$ will serve as a "don't-care" symbol. The following definition formalizes this idea.

Definition 3.4 (Specification relation between strings) 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 $r(i) \neq *$. For functions f, g, with values in Σ_0^* we will write $f \preceq g$ if for all s we have $f(s) \preceq g(s)$. ⌐

Thus, a don't-care symbol $r(i)$ imposes no restriction on s in the relation $r \preceq s$.

Definition 3.5 Let Tr be any deterministic transition function with states in some alphabet $\mathbb{A}$ 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

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

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

a) never changes *Input*;

b) has monotonic output;

c) does not change anything if the middle argument has all #'s in its input field, or if all three arguments have $*$ in all their fields.

We will call a cellular automaton with such a transition function a *standard computing medium*. Our construction uses for this a special field called

$$Color.$$

It can take a constant number of values; in one case, only $-1, 0, 1$. ⌟

The computation will occur in the segment of the infinite space marked with color 0; the space to the left and right of it will be marked with color -1 and 1 respectively. The reason for this will be explained in Section 12.

Definition 3.6 (Init) Let Σ_0 be the standard input/output alphabet, and $\mathbb{S}$ a set of states with $\mathbb{S}.Input = \Sigma_0$, having another field $Color \in \{-1, 0, 1\}$, and a distinguished "latent" state s_0. Let $\psi_* : \Sigma_0^* \to \mathbb{S}^*$ be an encoding. For any finite string $\rho \in \Sigma_0^n$, let $n' = |\psi_*(\rho)|$. For any finite or infinite $N > |\rho'|$, we construct the initial configuration

$$\xi' = Init_{\psi_*}(\rho) \in \mathbb{S}^N$$

as follows. Define ξ by setting $\xi(x) = \psi_*(\rho)(x)$ for each $x \in [0, n')$, and s_0 for all other x. In case $N = \infty$, obtain ξ' from ξ by setting

$$\xi'(x).Color = \begin{cases} 0 & \text{for } 0 \leq x < n, \\ -1 & \text{for } x < 0, \\ 1 & \text{for } x \geq n. \end{cases}$$

When the space is the finite one $\mathbb{Z}_N$ then, with $r = (N-n)/2$, set $\xi'(x).Color = -1$ for $-r \leq x < 0$, and 1 for $n \leq x < n + r$. ⌟

For some $c > 1$ let

$$h_1(t, c) = c^{(\log t)^{1/2} \log \log t}. \tag{3.4}$$

Theorem 3.4 (Reliable computation in dim 1) *Let Tr be a standard computing transition function over an alphabet Σ, and $\delta > 0$ a constant. There is*

- *a transition function Tr' with a state space $\mathbb{S}$ having fields Input, Output, Color, with $\mathbb{S}.Input = \mathbb{S}.Output = \Sigma_0$.*

- *a code $\psi_* : \Sigma_0^* \to \mathbb{S}^*$, with $|\psi_*(\rho)| \leq 2|\rho|$;*

- *a constant $c_1 > 1$*

such that the following holds for all $\rho \in \Sigma_0^*$. Let $\zeta(x,t)$ be any trajectory of Tr with $\zeta(x,0).\textbf{Input} = \rho(x)$ for $x \in [0,|\rho|)$ and filled with $*$'s otherwise. Let η be any trajectory of an ε-perturbation of Tr', with $\eta(\cdot,0) = Init_{\psi_*}(\rho)$. For all t, all $t' > t \cdot h_1(t,c_1)$, all $x \in \mathbb{Z}$ we have for all sufficiently small ε:

$$\mathsf{P}\{\zeta(x,t).\textbf{Output} \npreceq \eta(x,t').\textbf{Output}\} < \delta.$$

So the slowdown paid for reliability is somewhat worse than logarithmic: by a factor $h_1(t,c_1)$. We will see that the code ψ_* is simple, hiding no complex computation. The proof of this theorem relies on a move we can call *lifting*: though the input configuration is a hierarchically encoded one, but only to the level needed by the size of the input. Higher levels will be built up as additional reliability is needed for longer computation time.

Remarks 3.7

1. This note is similar to Remark 3.2, but is even more important, as the error term does not even decrease with increasing t.

2. The initialization via *Init* avoids building up an infinite hierarchy in the initial configuration, leaving this to self-organization. But it introduces a huge asymmetry by coloring the left of the input with -1 and the right with 1. This has an effect similar to making the space infinite in only one direction, and placing the input into the beginning.

⌟

4   Codes

For the moment, let us focus on the task of remembering a single constant-size field of a cellular automaton, called

<div align="center">

Main-bit.

</div>

The intention is that in the initial configuration, each cell's *Main-bit* is the same, and the transition function should try to keep this field constant. We mentioned in Section 1.2 that in one dimension, even this simple task will require the construction and maintenance of some non-local organization, as this seems the only way to eliminate large islands. The first idea is to store pieces of information redundantly in segments, which we will call colonies. But then colonies will be organized into supercolonies, and so on. To simplify this idea, we will say that colonies encode cells of another cellular automaton (which can have its own colonies...).

4.1 Colonies

A colony is a fixed-size segment of cells that are supposed to cooperate.

Definition 4.1 (Colony) Let x be a site and Q a positive integer. The set of Q sites $x + i$ for $i \in [0, Q)$ 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.

Definition 4.2 (Address) An *address field* is a field of our cells that we will denote *Addr*: we will always have it when we speak about colonies. The value $\xi(x).$*Addr* is a binary string which will be interpreted as an integer in $[0, Q)$. ⌋

Generally, we will assume that the *Addr* field is large enough (its size is at least $\log Q$). Then we could call a certain Q-colony $\mathcal{C}$ a "real" colony of ξ if for each element y of $\mathcal{C}$ with address i we have $\xi(y).$*Addr* $= i$. (But we do not introduce such a definition, since the address field of some cells could be faulty.) 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 activity of restoring this information from the degradation caused by faults (for example with the help of some majority operation). This activity could be repeated periodically.

Definition 4.3 (Work period) For a parameter U that we will fix in each case, 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. One can indeed write such transition functions; however, 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.

We will codify another convention:

Definition 4.4 Let us denote by M_1 the fault-tolerant cellular automaton to be built. ⌐

In the automaton M_1, 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, and so on; the organization must be a *hierarchy of simulations* (more precise definitions follow).

4.2   Block codes

The notion of simulation relies on the notion of a *code*, since the way the simulation works is that the simulated history can be decoded from the simulating history. We will develop a system of codes, starting from the simplest, well-known example of a block code over strings.

Definition 4.5 (Code on strings) A code φ between two sets R, S is 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. We will be particularly interested in the example when for a positive integer Q called the *block size* and some finite sets $\mathbb{S}_1, \mathbb{S}_2$ we have $R = \mathbb{S}_2$, $S = \mathbb{S}_1^Q$. Such a code is called a *block code*. In a block code, strings of the form $\varphi_*(r)$ are called *codewords*. The elements of a codeword $s = \varphi_*(r)$ are numbered as $s(0), \ldots, s(Q-1)$.

The codes φ between sets R, S used in our simulations will have a feature similar to the acceptance and rejection of Example 4.7. The set R will always

have a subset of symbols called *vacant*, and among them a distinguished special symbol *Vac*. An element $s \in S$ will be called *accepted* by the decoding if $\varphi^*(s) \neq Vac$, otherwise it is called *rejected*. ⌟

Having more than one vacant symbol is just a matter of convenience.

A simple example code γ would be $R = \{0, 1\}$, $S = R^3$, $\gamma_*(r) = (r, r, r)$ while $\gamma^*((r, s, t))$ is the majority of r, s, t.

Remark 4.6 *The notation (φ^*, φ_*) to use for decoding and encoding is not in common use: I find it suggestive, though, since φ^* is something like an inverse function without actually being one.* ⌟

The following block code can be considered a significantly more complex, paradigmatic example of the codes we will use later.

Example 4.7 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 $s.\textit{Addr}$ and $s.\textit{Info}$ of a state $r = (s_0, \ldots, s_{11})$ in $\mathbb{S}_1$. The *Addr* field consists of the first 5 bits $s_0, \ldots, s_4$, 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 $\varphi_*(r) = (s(0), \ldots, s(30))$ of elements of $\mathbb{S}_1$ to each element r of $\mathbb{S}_2$. Let $r = (r_0, \ldots, r_{11})$. We will set $s(i).\textit{Info} = r_i$ for $i = 0, \ldots, 11$. The 5 bits in $s(i).\textit{Addr}$ will denote the number i in binary notation. This did not determine all bits of the symbols $s(0), \ldots, s(30)$ 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 $i \notin [0, 12)$ are not determined. To determine $\varphi_*(r)$ completely, we could set these bits to 0.

The decoding function is simpler. Given a word $s = (s(0), \ldots, s(30))$ we first check whether it is a "normal" codeword, and as such, has $s(0).\textit{Addr} = 0$ and $s(i).\textit{Addr} \neq 0$ for $i \neq 0$. If yes then $r = \varphi^*(s)$ is defined by $r_i = s(i).\textit{Info}$ for $i \in [0, 12)$, and the word is accepted'. Otherwise $\varphi^*(s) = 0 \cdots 0$, and the word is 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. ⌟

Example 4.8 The trivial example here 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

$$\iota_Q^*((s(0), \ldots, s(Q-1))) = s(0) \sqcup \cdots \sqcup s(Q-1),$$

28

												Age
												Addr
												Worksp
												Info
												Program

Figure 1: Three neighbor colonies with their tracks

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. ⌟

Definition 4.9 (Aggregating a field) Let φ be a block code from $\mathbb{S}_2$ to $\mathbb{S}_1^Q$, and F_1, F_2 be fields of $\mathbb{S}_i$. We will say that φ *aggregates* field F_1 into F_2 if

$$\varphi^*((s(0),\dots,s(Q-1))).F_2 = s(0).F_1 \sqcup \dots \sqcup s(Q-1).F_1.$$

Hence for $u,v \in \mathbb{S}_2$ the assumption $u.F_2 = v.F_2$ implies that for all $a \in [0,Q)$ we have

$$\varphi_*(u)(a).F_1 = \varphi_*(v)(a).F_1. \tag{4.1}$$

⌟

We will use a strengthening of the aggregating property for a particular address a.

Definition 4.10 (Controlling by a field) Assume that code φ aggregates field F_1 into F_2. We say that field F_1 *controls* address a for code φ via a function $\gamma : \mathbb{S}_1.F_1 \to \mathbb{S}_1$ if in all codewords w of the form $w = \varphi_*(u)$, we have $w(a) = \gamma(w(a).F_1)$. ⌟

In general, a symbol $w(a)$ of a codeword $w = \varphi_*(u)$ may contain information in its fields other than F_1 about u, and it is typically important for some symbols of w to do it. But there is no harm in freeing any one particular address a, say $a = 1$, from this responsibility.

4.3 Generalized cellular automata (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 $0 \le i < Q$,

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

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 reasoning about error correction that decoding should notice all parts of a configuration that form a colony, even if a shifted one. 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 $(\zeta(0), \ldots, \zeta(Q-1))$, and $\zeta^*(1)$ to the value decoded from $(\zeta(Q), \ldots, \zeta(2Q - 1))$. There is no site to correspond to the value decoded from $(\zeta(1), \ldots, \zeta(Q))$.

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.

An *medium*, or *generalized cellular automaton* is given by the following ingredients:

$$\mathbb{S}, \Lambda, B, \textit{Configs}, \textit{Histories}, \textit{Trajs}, (\mathcal{A}_t)_{t \ge 0}.$$

$\mathbb{S}$ is the set of possible local states, and it has a subset of distinguished states called *Vacant*, and among them one distinguished state called *the vacant state* $Vac \in Vacant$. (Definition 4.5 already hinted at the meaning of the vacant states.) Λ is the set of sites as introduced in Definition 2.2, in our case the product of a few sets of the form $\mathbb{Z}_m$ for finite or infinite m. The positive integer B is called the *body size*. In ordinary cellular automata, $B = 1$. *Configs* is the set of functions $\xi : \Lambda \to \mathbb{S}$ that are *configurations*. If in a configuration ξ we have $\xi(x) \notin Vacant$ then we will say that there is a *cell* at site x in ξ. For a site x, interval $[x, x + B)$ will be called the *body* of a possible cell with *base* x. In a *configuration*, cells must have non-intersecting bodies. *Histories* is the set of functions $\eta : \Lambda \times [0, \infty) \to \mathbb{S}$ that are *histories*. In a history η is is required that

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 (see Definition 2.3) for each site x;

Trajs is the set of random histories (μ, η) that are *trajectories*. The increasing system $(\mathcal{A}_t)_{t \geq 0}$ of σ-algebras was introduced in Section 2.3. The sets $\mathbb{S}$, Λ, *Configs* and *Histories* are defined by the set *Trajs* implicitly—therefore we may omit them from the notation, and write

$$Med(B, \textit{Trajs}, \{\mathcal{A}_t\}) = Med(\mathbb{S}, \Lambda, B, \textit{Configs}, \textit{Histories}, \textit{Trajs}, (\mathcal{A}_t)_{t \geq 0})$$

for a medium defined by them. We have a body size parameter B because some of our cellular automata will "live in simulation", and it is very convenient for the simulated cell's body to coincide with the interval of the "base" cells simulating it.

Definition 4.11 (Dwell period) A *dwell period* of η 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 switching times with $\eta(x, t_1) = s$. The rectangle $[x, x + B) \times [t_1, t_2)$ is the *space-time body* of the dwell period.   ⌟

Ordinary cellular automata obey some further restrictions:

Definition 4.12 (Lattice configuration) A configuration is a *lattice configuration* if all cells are at sites of the form iB for integers i. We can also talk about *lattice histories*: these have space-time bodies of the form

$$[iB, (i+1)B) \times [jT, kT)$$

for integers $i, j < k$.   ⌟

A lattice history is a history of an ordinary cellular automaton, except that cells are given a common size B possibly different from 1, and the time step is given a size T possibly different from 1.

Definition 4.13 (Deterministic cellular automaton with size parameters) A *deterministic cellular automaton*

$$CA(Tr, B, T, \Lambda)$$

is determined by parameters $B, T > 0$ and a transition function $Tr : \mathbb{S}^3 \to \mathbb{S}$. We may omit some obvious arguments from this notation. A lattice history 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 history η let us write

$$\overline{Tr}(\eta, x, t, B) = Tr(\eta(x - B, t), \eta(x, t), \eta(x + B, t)). \tag{4.2}$$

⌟

Probabilistic cellular automata and perturbations are generalized correspondingly as

$$PCA(\mathbf{P}, B, T, \Lambda), \qquad CA_\varepsilon(Tr, B, T, \Lambda). \tag{4.3}$$

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

We will also consider cellular automata in which the dwell periods do not have a fixed length:

Definition 4.14 (Variable-period cellular medium) A medium is said to have a *variable period* if in its histories, not all dwell periods have necessarily the same size. ⌋

Block codes between cellular automata In a cellular abstract medium with body size B, a colony of size Q is going to be some set of cells $x + iB$ for $i \in [0, Q)$. Thus, the union of its cell bodies occupies the interval $[x, x + QB)$. In what follows we use decoding to recognize colonies (when reasoning about a configuration).

Definition 4.15 (Overlap-free) A block code will be called *overlap-free* if for every string $(s(0), \ldots s(n-1))$, and all $i \leq n-Q$, if both strings $(s(0), \ldots, s(Q-1))$ and $(s(i+1), \ldots, s(i+Q-1))$ are accepted then $i \geq Q$. ⌋

In other words, a code is overlap-free if two accepted words cannot overlap in a nontrivial way.

The simple tripling code $\varphi_*(x) = (x, x, x)$ with majority decoding is not overlap-free, since for example if $s(1)s(2)s(3)s(4) = 0000$ then both $s(1)s(2)s(3)$ and $s(2)s(3)s(4)$ are accepted. On the other hand, the code $\varphi_*(x) = (0, x, 1, x, 2, x)$ with $\varphi^*(0, x, 1, y, 2, z) = \text{Maj}(x, y, z)$ (and vacant in other cases) is overlap-free. The code in Example 4.7 is similarly overlap-free. Overlap-free codes are used, among others, in [23].

A block code φ of block size Q can be used to define a code on configurations between cellular abstract media M_1 and M_2.

Definition 4.16 (Block code on configuration) Suppose that a configuration ξ of M_2, which is a cellular medium $AMed(QB)$, is given. Then we define the encoded configuration $\xi_* = \varphi_*(\xi)$ of M_1, which is an $AMed(B)$, by setting for each cell x of ξ and $0 \leq i < Q$,

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

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

$$s = (\xi(x), \xi(x+B), \dots, \xi(x+(Q-1)B)). \qquad (4.4)$$

We define $\xi^*(x) = \xi'(x)$ if the latter is vacant or there is no y closer than QB to x with $\xi'(y)$ non-vacant; otherwise $\xi^* = Vac$. ⌟

Proposition 4.17 *Given a block code $\varphi = (\varphi_*, \varphi^*)$, its extension to configurations is also a code in the sense that the equation*

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

still holds.

The proof of this statement is immediate.

As we know most configurations cannot be obtained by encoding. Those that can merit a special name.

Definition 4.18 (Code configuration) A configuration ξ is called a *code configuration* if $\varphi_*(\varphi^*(\xi)) = \xi$ and the decoded configuration $\varphi^*(\xi)$ covers the space with adjacent cells. ⌟

For example, if the code is the simple tripling $\gamma_*(x) = (x, x, x)$ mentioned above, then any code configuration would consist of triply repeated symbols.

Let φ be an overlap-free code. If $\xi = \varphi_*(\zeta)$ is a code configuration with $\xi^* = \varphi^*(\xi)$ then $\xi^*(x)$ is nonvacant only at positions x where $\zeta(x)$ is nonvacant. If ξ is not a code configuration then it may happen that in ξ^*, the cells will not be exactly at a distance QB apart. Our construction still garantees that the distance of cells in $\varphi^*(\xi)$ is at least QB. This situation can be taken as one of the justifications for the notion of cellular abstract media.

4.4 Block simulations

A simulation is defined as a code allowing to decode the trajectory of one cellular automaton from another.

Definition 4.19 (Block simulation between deterministic cellular automata) 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 4.7: 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 history η of M_1, we define the history $\eta^* = \varphi^*(\eta)$ of M_2 by setting

$$\eta^*(\cdot, t) = \varphi^*(\eta(\cdot, t)). \tag{4.5}$$

We will say that the code φ is a *simulation* if for each configuration ξ of M_2, for the (unique) trajectory η of M_1, such that $\eta(\cdot, 0) = \varphi_*(\xi)$, the history η^* is a trajectory of M_2. ⌟

We can view φ_* as an encoding of the initial configuration of M_2 into that of M_1. Our requirements say that from every trajectory of M_1 with a "good" initial configuration (that is a code configuration), the simulation-decoding results in a trajectory of M_2.

Example 4.20 Let us show one particular way in which the transition function Tr_1 can make the code φ a simulation. 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 4.7, this may happen as follows. Let us denote by r_{-1}, r_0, r_1 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_{-1} and r_1 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. ⌟

Example 4.21 (Aggregated transition) Here is 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 Tr^Q is $\mathbb{S}^Q$. Let $\mathbf{r}_j = (r_j(0), \ldots, r_j(Q-1))$ 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 $Tr^Q(\mathbf{r}_{-1}, \mathbf{r}_0, \mathbf{r}_1)$, defining an *aggregated cellular automaton* $CA(Tr^Q, Q, Q)$ where we used the notation in Definition 4.13.

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 4.8 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 $F, G, \cdots \subseteq \mathsf{All}$ 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 $\mathbf{r}.F = r(0).F \sqcup r(1).F \sqcup \cdots \sqcup r(Q-1).F$.

We may allow only less communication with the neighbors, and then modify the above as follows to a function $Tr^{Q,w}$ as follows, where we call δ the *slowdown rate*, where we assume that *both δQ and $1/\delta$ are integers*. The state space is still $\mathbb{S}^Q$. Let $\mathbf{r}_j = (r_j(0), \ldots, r_j(Q-1))$ 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 *only wQ times* to get a string of Q symbols of $\mathbb{S}$: this is the value of $Tr^{Q,\delta}(r_{-1}, r_0, r_1)$. This defines the cellular automaton $CA(Tr^{Q,\delta}, Q, \delta Q)$. The function Tr^Q becomes the special case $Tr^{Q,1}$. Denoting by $s.\mathsf{slice}^{\delta,-1}$, $s.\mathsf{slice}^{\delta,1}$ the leftmost and rightmost $\delta|Q|$ symbols of a string s in $\mathbb{S}^Q$, then in fact $Tr^{Q,\delta}(\mathbf{r})$ depends only on $r_{-1}.\mathsf{slice}^{\delta,1}, r_0, r_1.\mathsf{slice}^{\delta,-1}$. ⌟

To simulate n steps of a transition function Tr over an interval $[a, b)$ start with a larger interval $[a - n, b + n)$. After step t, we only need to work on the interval $[a - n + t, b + n - t)$.

Cellular automata are "computationally universal" in the informal sense that Turing machines are: arbitrary computations can be implemented on appropriate cellular automata. We will return to this point later, in Section 9.1. Also, there are cellular automata that are universal in a sense similar to universal Turing machines. The notion of block simulations give rise to a particularly simple such universality property.

Definition 4.22 (Intrinsically universal cellular automata) A transition function Tr is *intrinsically 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 $CA(Tr, 1, 1)$. ⌟

In the literature, intrinsic universality is defined in a slightly more general way: see [29]. There are some extremely simple cellular automata that are

computationally universal but are probably not intrinsically so. The most famous example is Cook's theorem for the computational universality of Rule 110 in [9].

Theorem 4.1 (Intrinsically universal cellular automata) *There is an intrinsically universal transition function.*

Here, we only give a proof sketch of existence. A particularly simple intrinsically universal cellular automaton is defined in [29]: it is one-dimensional, nearest-neighbor, with 4 states.

Sketch of proof: This theorem is proved somewhat analogously to the theorem on the existence of universal Turing machines. If the intrinsically 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_{-1}, r_0, r_1) 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, 34] ☐

Note that an intrinsically universal cellular automaton cannot use codes similar to Example 4.7. 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 intrinsically universal then the various simulations in which it participates will have arbitrarily large colony sizes.

The size Q of the simulating colony in the above proof sketch 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 $\|\mathbb{S}_2\|$). We will only deal with such automata, and will develop a simple language to describe their programs. (Any intrinsically universal transition function will simulate these with correspondingly small Q and U.)

4.5 A 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 local states in addition to the ones introduced earlier.

The automaton M_1 is not intrinsically universal, so 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 [2]. But we will see later that intrinsic 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 computationally universal cellular 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.

---

[2]We could avoid non-constant size fields, but since it is not necessary, for simplicity we keep them

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".

4.6 General simulations

The general notion of abstract media allows a very general definition of simulations:

Definition 4.23 (General simulation) A *simulation* of medium M_2 by medium M_1 having the same system $\{\mathcal{A}_t\}_{t \geq 0}$ of σ-algebras is given by a pair

$$\Phi = (\varphi, \Phi^*) \tag{4.6}$$

where Φ^* is a mapping of the set of histories of M_1 into those of M_2 (the decoding), φ_* is a mapping of the set of configurations of M_2 to the set of configurations of M_1 (the encoding for initialization). We will assume to φ_* always belongs also a decoding function φ^*, so they form a code $\varphi = (\varphi_*, \varphi^*)$. Denote

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

We require that for each configuration ξ of M_2, each trajectory of η of M_1 with $\eta(\cdot, 0) = \varphi_*(\xi)$ the decoded history η^* is a trajectory of M_2 with $\eta^*(\cdot, 0) = \xi$. ⌟

The actual simulations we will use will all have a certain locality property:

Definition 4.24 (Local simulation) A simulation will be called *non-anticipating* if for each t the random function $\eta^*(\cdot, t)$ is measurable in the σ-algebra $\mathcal{A}_t$ (this says that it does not depend on the future). Let σ be the shift on one-dimensional configurations and histories: that is $(\sigma\xi)(x) = \xi(x + 1)$, $(\sigma\eta)(x, t) = \eta(x+1, t)$. A simulation (φ, Φ^*) is called *shift-invariant* if both φ and Φ^* commute with the shift. It is called *local* if there is a finite space-time rectangle $V^* = I \times (u, 0]$ with $u \geq 0$ such that $\Phi^*(\eta)(w, t)$ depends only on $\eta((w, t) + V^*)$. A local simulation is both non-anticipating and shift-invariant.

All our simulations will be local, unless stated otherwise. It follows from the non-anticipating property that $\eta^*(\cdot, 0)$ depends only on $\eta(\cdot, 0)$, and therefore the simulation always defines a decoding function

$$\varphi^*$$

on configurations (as already stipulated in Definition 4.23). If $u = 0$ and so the configuration $\eta^*(\cdot, t)$ depends only on the configuration $\eta(\cdot, t)$, then the simulation will be called *memoryless*. ⌟

Corollary 8.11 gives an example of non-local simulation. Locality implies that a simulation is determined by a function defined on the set of configurations over V^*. Our eventual simulations will not be memoryless: the decoding looks back on the history during $(t - u, t]$ but, being non-anticipating, still does not look ahead. For a memoryless simulation, the simulation property is identical to the one we gave at the beginning of Section 4.4.

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 history 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 probabilities 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 Section 4.5, 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).

Trickle-down 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. Let us define the nature of this feedback via the notion of trickle-down. We want the cell state simulated by a colony to determine *some properties* (fields) of the simulating cells in the colony; however, we want to allow some other fields to vary. For example there could be an *Age* field that varies during any work period, while the information of the big cell represented by the colony should be reflected in its cells—via the code—all the time.

Definition 4.25 (Trickle-down) Let $\Phi = (\varphi, \Phi^*)$ be a simulation as in (4.6) whose encoding φ_* is a block code with block size Q, between abstract cellular media M_k ($k = 1, 2$). Let $D_k > 0$ ($k = 1, 2$) be some parameters, and F_1 a field of the medium M_1. We say that Φ has the *ε-trickle-down property* of field F_{k+1} into field F_1 with respect to D_1, D_2 if the following holds for every configuration ξ of M_2 and every trajectory (μ, η) of M_1 with $\eta(\cdot, 0) = \varphi_*(\xi)$.

Recall the notation $\preceq$ in Definition 3.4. Let $\eta^1 = \eta$, $\eta^2 = \Phi^*(\eta)$. 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. For $k = 1, 2$ let $\mathcal{E}_k(x)$ be the event that $\eta^k(x, t)$ is non-vacant and $\eta^k(x, t).F_k$ does not change during $(t_0 - D_k, t_0 + D_k]$. Let $\mathcal{E}'_1$ be the event that $\mathcal{E}_1(x_1)$ holds and also

$$\eta(x_1, t).F_1 \succeq \varphi_*(\eta^*(x_2, t))(a).F_1$$

during this time. Then $\mathsf{P}(\mathcal{E}_2(x_2) \setminus \mathcal{E}') < \varepsilon$. ⌟

The term "trickle-down" is used since information from a higher level cell is used to determine something in a lower-level cell. 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 $\eta^*(x_2, t')$ and taking the a-th symbol of the code-word.

There is yet another, simple error-resistance property of a medium worth spelling out: namely, the time until the initial state does not change, except with small probability.

Definition 4.26 (Initial stability) A cellular medium M is called *initially stable* for an initial configuration ξ with parameters (ε, T) if for each trajectory η of M with $\eta(\cdot, 0) = \xi$, for each site x, we have

$$\mathsf{P}\{\exists t < T \ \eta(x, t) \neq \eta(x, 0)\} < \varepsilon.$$

⌟

Initial stability is a trivial property for lattice cellular automata:

Example 4.27 Consider the perturbation $M = CA_\varepsilon(Tr, B, T)$ of a 1-dimensional deterministic cellular automaton. Then for any configuration ξ and arbitrary

$\delta > 0$ the medium M is initially stable for initial configuration ξ and parameters (δ, T). Indeed, by definition no cell is changing it state before time T. ⌋

For variable-period cellular automata, and other media in which the length of dwell periods is not fixed in advance, the property is less trivial.

5 Hierarchy

The reliable cellular automaton we are building is working in a hierarchical way. Before understanding a dynamically working hierarchy, it is helpful to understand thoroughly a static hierarchy, for example a hierarchically constructed initial configuration.

5.1 Hierarchical codes

In the present work, formally, a hiearchy will be defined via a "composite code". Let us introduce the basic operation of the hierarchical structure arising in an amplifier.

Definition 5.1 (Composition of codes) 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, that is the cell body sizes are in the corresponding relation. The code $\varphi \circ \psi$ is called the *composition* of φ and ψ. ⌋

Here is a detailed example.

Example 5.2 Let M_1, M_2, M_3 have cell body sizes $1, 31, 31^2$ respectively. Let us use the code φ from Example 4.7. 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 grouped 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 $-15 + 31j$

| Prog* | Info* | Worksp* | Addr* | Age* | Age
Addr
Worksp

Info

Prog |

Figure 2: Fields of a cell simulated by a colony

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. ⌟

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

Definition 5.3 (Hierarchical code) A *hierarchical code* is given by a sequence $\mathbb{S}_k, \varphi_{k*}, (k \geq 1)$ where $\mathbb{S}_k$ is an alphabet and φ_{k*} is a code on configurations. In the present work we will also assume that φ_{k*} always comes from a block code

$$\varphi_{k*} : \mathbb{S}_{k+1} \to \mathbb{S}_k^{Q_k}.$$

Since $\mathbb{S}_k$ and Q_k are implicitly defined by φ_{k*} we can refer to the code as just $(\varphi_{k*})_{k \geq 1}$. Consider a hierarchical code $(\varphi_{k*})_{k \geq 1}$ with state sequence $(\mathbb{S}_k)$ and block sizes (Q_k). If for each $\mathbb{S}_k$ a field F_k is also given, and the code φ_{k*} is aggregating from F_k to F_{k+1} as in Definition 4.9, then the hierarchical code $(\varphi_{k*})_{k \geq 1}$ will be called *aggregating* for the field sequence (F_k). If also F_k controls address 1 via a function γ_k (as by Definition 4.10) then sequence of

triples

$$\mathbf{H} = (\varphi, \mathsf{F}_k, \gamma_k)_{k \geq 1} \tag{5.1}$$

will be called a *centrally aggregating hierarchical code*, or shortly, a *code complex*. ⌟

Definition 5.4 Given a hierarchical code as above, for a configuration ξ we define the higher-order configurations ξ^k by the recursion

$$\begin{aligned} \xi^1 &= \xi, \\ \xi^{k+1} &= \varphi_k^*(\xi^k). \end{aligned} \tag{5.2}$$

The configuration ξ is called a *code configuration* for the given hierarchical code if ξ^k is a code configuration for the code φ_k for each k in the sense of Definition 4.18. ⌟

Constructing code configurations The construction of a code configuration for a hierarchical code seems to require a composition $\varphi_{1*} \circ \varphi_{2*} \circ \cdots$. What is the meaning of this? We will want to compose the codes "backwards", that is in such a way that from a configuration ξ^1 of some medium 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 $B_3 = Q_1 Q_2$, and so on. Such constructions are not unknown, they were used for example to define "Morse sequences" with applications in group theory as well in the theory of quasicrystals ([23, 32]). All known nonperiodic tiling sets are also based on a similar hierarchical construction.

Definition 5.5 Suppose that a code configuration ξ of a hierarchical code $(\varphi_k)_{k \geq 1}$ with block sizes Q_k is given. The sizes of the higher-order cells are defined by $B_1 = 1$, $B_{k+1} = Q_k B_k$. For a finite or infinite n, let

$$K_0(n) = \sup_{B_k \leq n} k. \tag{5.3}$$

If $n < \infty$ then this is the largest k for which a k-cell can fit into $\mathbb{Z}_n$. As long as $k < K_0(n)$ the origin of our space Λ will be contained in some cell x_k of level k. Since it is in our power to define the code configuration ξ, for simplicity we will always arrange that x_k is the second cell (the cell with address 1) of the colony coding cell x_{k+1}. In general, we could use any addresses $0 < a_k < Q - 1$ instead of setting them always to 1, but this is not important now. Setting $a_k = 1$ gives

$$x_1 = 0, \quad x_k = -B_1 - \cdots - B_{k-1} \text{ for } k > 1.$$

⌟

In a hierarchical code as in Definition 5.5, the sequence of states

$$s_k = \xi^k(x_k)$$

obeys some restrictions:

Definition 5.6 (Fitted sequence) A finite or infinite sequence $(s_1, s_2, \dots)$ will be called *fitted* to a code complex $\mathbf{H}$ as in (5.1) if

$$\varphi_{k*}(s_{k+1})(1) = s_k \tag{5.4}$$

for all k. ⌋

Every fitted sequence allows the construction of a code configuration for the hierarchical code. (If we used addresses a_k then the condition above would be $\varphi_{k*}(s_{k+1})(a_k) = s_k$, and the proposition below would still hold.)

Proposition 5.7 *For a fitted sequence $(s_k)_{k \geq 1}$ there are configurations ξ^k of M_k over $\mathbb{Z}$ such that for all $k \geq 1$ we have*

$$\varphi_{k*}(\xi^{k+1}) = \xi^k,$$
$$\xi^k(x_k) = s_k.$$

Proof. Let N (finite or infinite) be the size of our space, and let

$$\xi_k^k \tag{5.5}$$

be the configuration of M_k which has state s_k at site x_k and arbitrary states at all other sites $x_k + xB_k$, with the following restriction in case of a finite space size N. Let ξ_k^k be non-vacant only for $k \leq K_0 = K_0(N)$ and $\xi_{K_0}^{K_0}(x_{K_0} + z)$ non-vacant only if

$$0 \leq zB_{K_0} < N. \tag{5.6}$$

For $1 \leq i < k$ let

$$\xi_k^i = \varphi_{i*}(\varphi_{(i+1)*}(\cdots \varphi_{(k-1)*}(\xi_k^k) \cdots)). \tag{5.7}$$

We have

$$\xi_{k+1}^k(x_k) = \varphi_{k*}(\xi_{k+1}^{k+1}(x_{k+1}))(1) = \xi_k^k(x_k) \tag{5.8}$$

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

Though the code configuration 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.

Here is how finite and infinite sequences will be encoded. For the following two definitions, let **H** be a code complex as in (5.1).

Definition 5.8 (Encoding of infinite sequences) Let $\rho \in (\mathbb{S}_1.\mathsf{F}_1)^{\mathbb{Z}}$ be an infinite sequence, we define its encoding $\psi_*(\rho)$ as follows. Define for each k the following segment of ρ using the notation of Definition 5.5:

$$\rho_k = (\rho(x_k), \rho(x_k + 1), \ldots, \ldots, \rho(x_k + B_k - 1)).$$

The sequence $s_k = \gamma_k(\rho_k)$, $k = 1, 2, \ldots$ is fitted, allowing via Proposition 5.7 to encode ρ into an infinite code configuration

$$\xi = \xi^1 = \psi_*(\rho).$$

⌟

This construction leaves plenty of freedom for the codes φ_k. Indeed, the tracks of a codeword $w = \varphi_{k*}(s_{k+1})$ other than F_k can store all kinds of structural and error-correcting information about s_{k+1}, and it is only its symbol $w(1)$ that is restricted to $\gamma_k(w(1).\mathsf{F}_1) = \gamma_k(\rho_k) = s_k$.

A sequence ρ of length $n < \infty$ will be encoded into a cell of an appropriate level. Recall the notation x_k from Definition 5.5.

Definition 5.9 (Encoding of finite sequences) Given a code complex **H** as above and a string $\rho \in \Sigma_0^n$ we define its encoding $\xi = \psi_*(\rho) \in \mathbb{S}^{\infty}$ which has the property that except for an interval of size $\leq 2n$, $\xi(x) = q_{\mathbb{S}}$ with $q_{\mathbb{S}}$ as in Definition 3.6. Let $k = K_0(n)$ as in (5.3). Create the string

$$\rho' = *^{-x_k} \rho *^l$$

where $-x_k$ stars come at the front and the l stars at the back are the fewest to make $|\rho'| = mB_k$ an integer multiple of B_k. Now create a string $r_k \in \mathbb{S}_k^m$ as follows. First let $r_k' = q_{\mathbb{S}}^m$, and then obtain r_k by replacing the track $r_k'.\mathsf{F}_k$ in it with ρ'. Define the string $r_{k-1} = \varphi_{i*}(r_k) \in \mathbb{S}_{k-1}^{mQ_{k-1}}$. Similarly, for each $1 \leq i < k$ let $r_i = \varphi_{(i+1)*}(r_{i+1})$. By definition of the codes φ_{i*}, string r_i consists of $mQ_{k-1}Q_{k-2}\cdots Q_i$ symbols of $\mathbb{S}_i$. Then r_1 consists of mB_k symbols of $\mathbb{S} = \mathbb{S}_1$. Finally obtain ξ as follows: $\xi(x) = r(x - x_k)$ for $0 \leq x - x_k < mB_k$ and $q_{\mathbb{S}}$ otherwise. ⌟

Now the symbols of ρ occupy the places $\xi(x).\mathsf{F}_1$ for $x \in [0, n)$, within the string r_1 representing a number $m \leq Q_k + 1$ of k-cells. Outside these k-cells the configuration consists of 1-cells with one and the same state $q_{\mathbb{S}}$.

5.2 Amplifiers

An amplifier is a hierarchical code whose member codes are also simulations. Recall general simulations in Definition 4.23.

Definition 5.10 (Amplifier) 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 somewhat analogous to renormalization groups in statistical physics. So far, we have not seen in the present work any nontrivial example of simulation other than between deterministic cellular automata, so the idea of an amplifier seems far-fetched at this moment.

We want the simulations in our amplifiers to have all the good properties introduced earlier:

Definition 5.11 (Trickle-down and initial stability for amplifiers) Suppose that an amplifier $\mathbf{\Psi} = (M_k, \Phi_k)_{k \geq 1}$ is given along with the sequences of parameters D_k, ε_k'', ε_k, and fields F_k. Assume that each φ_k aggregates field F_k into F_{k+1}, as in Definition 4.9.

1. $\mathbf{\Psi}$ has the *trickle-down* property if for each k, the simulation Φ_k has the trickle-down property of F_{k+1} to F_k with parameters D_k, D_{k+1} and ε_k'', as in Definition 4.25.

2. Suppose that the sequence $\varepsilon_k > 0$, $k = 1, 2, \dots$ is given with $\sum_k \varepsilon_k < 1/6$, and also a configuration ξ of M_1 that is a code configuration of the code sequence (φ_k) (as introduced in Definition 5.3). We say that $\mathbf{\Psi}$ has the *initial stability property* with respect to the initial configuration ξ and parameters $((\varepsilon_k, T_k))_{k \geq 1}$ if for each k the medium M_k has the initial stability property for initial configuration ξ^k and parameters $(\varepsilon_k, 2D_k)$.

⌟

The initial stability property for amplifiers is not as trivial as for lattice media. Consider the case when $M_1 = CA_\varepsilon(Tr, \mathbb{Z}, B, T)$. As the trajectories of medium M_2 are obtained by decoding from trajectories of M_1, their initial stability is not apriori guaranteed.

5.3 Information storage: proof from an amplifier assumption

The following lemma will be proved later in the paper.

Lemma 5.12 (Initially stable trickle-down amplifier) *We can construct the following objects.*

a) *Parameters $\varepsilon_k < \varepsilon_k''$, B_1, Q_k, D_k with $D_{k+1} \geq D_k$ for $k \geq 1$, with $\sum_k \varepsilon_k'' < 1/6$.*

b) *Media M_k with local state spaces $\mathbb{S}_k$, with a distinguished field F_k.*

$$M_1 = CA_{\varepsilon_1}(Tr_1, B_1, T_1, \mathbf{Z}), \qquad (5.9)$$

and an amplifier $\mathbf{\Psi} = ((M_k), (\Phi_k))$ with simulations $\Phi_k = (\varphi_k, \Phi_k^*)$ where the codes φ_k are block codes with blocksize Q_k that are aggregating field F_k into field F_{k+1} as per Definition 4.9, and where F_k controls address 1 as in Definition 4.10.

c) *For each infinite sequence $\rho \in \Sigma_0^*$ an initial configuration $\xi^1 = \psi_*(\rho)$ of M_1 as obtained in Proposition 5.7 such that for each k the sequence ξ^k covers the space with non-vacant cells of M_k.*

d) *$\mathbf{\Psi}$ is trickle-down and initially stable for ξ, with the parameter sequences (F_k), (D_k), (ε_k''), (ε_k).*

Note that (5.9) is required only for M_1; the media M_k for $k > 1$ will be more complex objects, not simply cellular automata. Let us use this lemma to prove Theorem 3.3.

Proof of Theorem 3.3. We will use the amplifier defined in the above lemma. Let $\rho \in \Sigma_0^{\mathbb{Z}}$, and ξ as in Lemma 5.12. Let η^1 be a trajectory of the medium M_1 with $\eta^1(\cdot, 0) = \xi$, and η^k be defined by the recursion $\eta^{k+1} = \Phi_k^*(\eta^k)$. Let (x_1, t_0) be a space-time point. There is a sequence of points $x_1, x_2, \ldots$ such that x_{k+1} is a cell of $\eta^{k+1}(\cdot, 0)$ containing x_k in its body. There is a first n with $t_1 < D_n$. Let $\mathcal{E}_k$ be the event that $\eta^k(x_k, t)$ is non-vacant and

$$\eta^k(x_k, t).\mathsf{F}_k = \xi^k(x_k).\mathsf{F}_k$$

during $(t_0 - D_k, t_0 + D_k]$. The theorem follows from the bounds on $\sum_k \varepsilon_k$ and $\sum_k \varepsilon_k''$ and from

$$\mathsf{P}\left(\neg(\mathcal{E}_1 \cap \cdots \cap \mathcal{E}_n)\right) \leq \varepsilon_n + \sum_{k=1}^{n-1} \varepsilon_k''. \qquad (5.10)$$

To prove this inequality, use

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

By the construction, $\eta^n(x_n, 0).\mathsf{F}_n = \xi^n(x_n)$. The initial stability property implies $\mathsf{P}\left(\neg\mathcal{E}_n\right) \leq \varepsilon_n$. Let us show $\mathsf{P}\left(\mathcal{E}_{k+1} \setminus \mathcal{E}_k\right) \leq \varepsilon_k''$. The assumption $\mathcal{E}_{k+1}$

says

$$\eta^{k+1}(x_{k+1}, t).F_{k+1} = \xi^{k+1}(x_{k+1}).F_{k+1}. \tag{5.11}$$

Let a be the address of cell x_k of M_k in the colony simulating cell x_{k+1} of M_{k+1}. By the definition of of ξ^k we have $\varphi_{k*}(\xi^{k+1}(x_{k+1}))(a) = \xi^k(x_k)$. The assumption that φ_k aggregates F_k into F_{k+1} and (5.11) implies, as shown in (4.1), $\varphi_{k*}(\eta^{k+1}(x_{k+1}, t))(a).F_k = \xi^k(x_k).F_k$. The trickle-down property says that except with probability ε_k'', we have $\varphi_{k*}(\eta^{k+1}(x_{k+1}, t))(a).F_k = \eta^k(x_k, t).F_k$. during $(t_0 - D_k, t_0 + D_k]$, which completes the proof. $\square$

5.4 Error-correcting codes

Let us define the parts of our block code that deal with error-correction.

Definition 5.13 (Error-correcting code) A block code $\varphi = (\varphi_*, \varphi^*)$ with $\varphi^* : R^Q \to S$ will be called is *t-error-correcting* if for all $x \in \Sigma$, $y \in \Sigma^Q$ we have $\varphi^*(y) = x$ whenever y differs from $\varphi_*(x)$ in at most t symbols. It is (β, t)-*burst-error-correcting* if for all $x \in S$, $y \in R^Q$ we have $\varphi^*(y) = x$ whenever y differs from $\varphi_*(x)$ in at most t intervals of size $\leq \beta$. For such a code, we will say that a word $y \in R^Q$ is (β, r)-*compliant* if it differs from a codeword of the code by at most r intervals of size $\leq \beta$.

Assume that strings R and S are in $\{0, 1\}^*$. In the context of error-correcting codes, elements of R will be called the *symbols* of the codewords, as opposed to *bits*, since each symbol may consist of several bits. ⌟

Example 5.14 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 that are *systematic*: 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. For correcting a single error, in the tripling method outlined in Section 4.5, the error check bits are simply the twice repeated information bits. ⌟

Example 5.15 Here we define the linear code we will be using in later construction (a generalization of the so-called Reed-Solomon code, see [7]). In its application in our paper only its basic quantitative properties are used, so the details can safely be skipped at first reading.

Let our codewords (information symbols and check symbols together) be binary strings of length Nl for some l, N. The *symbols* of our codewords,

binary strings of length l, will be interpreted as elements of the finite field $GF(2^l)$ and thus, each binary string c of length Nl will be treated as a vector $(c(0), \ldots, c(N-1))$ 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 a positive integer. The codewords are those vectors c that satisfy the equations

$$\sum_{i=0}^{N-1} \alpha_i^j c(i).F^k = 0 \ (j = 1, \ldots, 2t) \tag{5.12}$$

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. 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 demonstrates that for the correction of error in any $\le t$ symbols, only $2t$ error-check symbols are needed. (Robert Solovay noticed that in our applications, with a little deterioration of efficiency, even the most brute-force decoding algorithm is sufficient. We will never have to correct more than some constant number c of errors, so the complexity of decoding would never go much over n^c steps.)

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

$$S_j = \sum_i (c_i + e_i)\alpha_i^j = \sum_i e_i \alpha_i^j = \sum_r Y_r X_r^j \tag{5.13}$$

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. 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 $r = 1, \ldots, \nu$, and any $j = 1, \ldots, 2t - \nu$:

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

Hence, summing for r,

$$0 = \sum_{s=0}^{\nu} \Lambda_s (\sum_r Y_r X_r^{j+\nu-s}) = \sum_{s=0}^{\nu} \Lambda_s S_{j+\nu-s} \ (j = 1, \dots, 2t - \nu) \qquad (5.14)$$

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 $\nu = 1, 2, \dots, t$, see if M_ν 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 (5.13) and see if the resulting corrected vector C satisfies (5.12). 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 $\mathrm{GF}(2^l)$. This finite field can be efficiently represented as the set of remainders with respect to an irreducible polynomial of degree l over $\mathrm{GF}(2)$, 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 $\mathrm{GF} < (2)$ for any s. So, the scheme works for all l of the form $2 \cdot 3^s$. ⌟

A hierarchical code like in Proposition 5.7 leaves some space for error-correction, but it needs to be economical, so we indeed need a code like Example 5.15.

Example 5.16 Let us use the notation of Proposition 5.7 with $a_k = 1$ for all k. Our goal is to give more details of state spaces $\mathbb{S}_k$ and codes φ_k satisfying the conditions of that Proposition. See Figure 3. The error check symbols for F_k are on a track *Redun*^{k}. As the address $a_k = 1$ is controlled by F_k, the cell with address 1 is not used for error checks. The other parts of the information needed to represent the big cell are on track L^k. It contains in its different segments the fields L^{k+1}, *Redun*$^{k+1}$, *Work*$^{k+1}$ of the big cell represented by the colony, as well as its own error check symbols. There is also a collection of tracks shown here as a single track *Work*^{k}: it contains for example *Addr*^{k}, *Mail*^{k}. ⌟

F^k				
		F^{k+1}		
$Redun^k$	Error-check symbols for the track F^k			
L^k	L^{k+1}	$Redun^{k+1}$	$Work^{k+1}$	Error check for this track
$Work^k$	Work tracks			
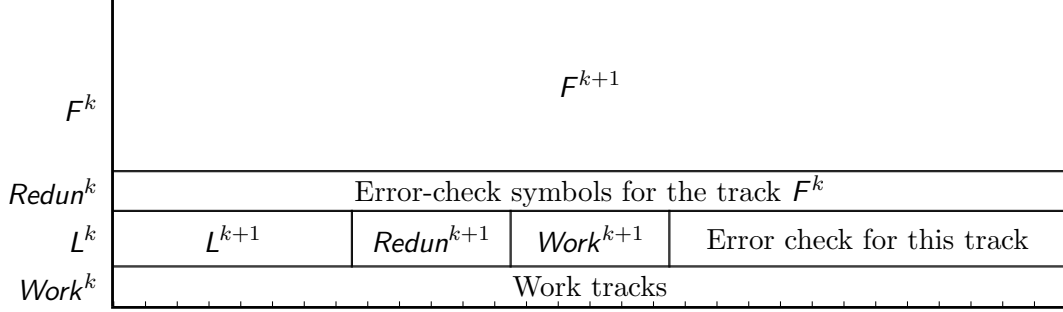
Figure 3: Error-correcting code in a shared field

In any implementation of the above example all fields other than the aggregated field F^k must be relatively quite narrow. Recall that $Cap_k = \|\mathbb{S}_k\|$ is the number of bits in a state of M_k.

Proposition 5.17 *In a hierarchical code (φ_k) let H_k denote any track of M_k that is not needed to restore information in a cell of M_{k+1} from the colony coding it. (It may contain error-checks or other administrative information like addresses.) Then $\sum_k |H_k|/Cap_k < 1$.*

Proof. Let $h_k = |H_k|/Cap_k$, $r_k = 1 - h_k$. The state of a cell of M_{k+1} must be represented by the colony even when we exclude the track H_k, so

$$Cap_{k+1}(r_{k+1} + h_{k+1}) \leq Q_k Cap_k r_k = Cap_{k+1} r_k,$$
$$r_{k+1} + h_{k+1} \leq r_k,$$
$$h_{k+1} \leq r_k - r_{k+1},$$

allowing to estimate $\sum_k h_k$ by a telescoping sum. $\qquad\qquad\square$

5.5 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 like its 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 Section 4.5. 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.

Less formally, "structure" does not represent any data for 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.

The only kind of disruption whose correction can be attempted along the lines of traditional error-correcting codes and repetition is the one of kind 1): a way of its correction was indicated in Section 4.5. The three other kinds are new; we will deal with them in different ways.

On 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 fast, 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, we "hard-wire" the program into the

transition function of each cell. We will not lose computational universality this way: as the hard-wired program itself can simulate some fixed cellular automaton on one of its tracks (which will be called the *Payload* track).

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. These configurations, viewed as "illegal" earlier, will be "legalized" now; this way they can be eliminated with their own active participation, provided we have rules (trajectory conditions) applying to them. This is the main reason for the introduction of generalized cellular automata.

The generalized cellular automaton introduced this way 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 reason about structures that arise in simulations by a random process.

6 Results for the finite space

This section overviews the main theorems concerning reliable computation with cellular automata in discrete time.

6.1 Relaxation time and ergodicity

Ergodicity means forgetting eventually everything about the initial configuration. We quantify "eventually" here by the notion of a relaxation time.

We start with some notation.

Notation 6.1 Let $\Lambda(n)$ be an increasing sequence of finite subsets of Λ with $\bigcup_n \Lambda(n) = \Lambda$, for example

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

if $\Lambda = \mathbb{Z}^d$. We can view an element $\mathbf{s}$ of $\mathbb{S}^{\Lambda(n)}$ as a vector whose components are indexed with the elements of $\Lambda(n)$. For a measure ν over configurations, denote

$$\nu(\mathbf{s}) = \nu\{\xi : \xi(x) = s_x \text{ for all } x \in \Lambda(n)\}. \tag{6.1}$$

For $n = 0$, we have the special case $\nu(s) = \nu\{\xi : \xi(0) = s\}$.

If (μ, η) is a random trajectory of a probabilistic cellular automaton then let μ^t be the distribution of the configuration $\eta(\cdot, t)$. ⌋

Definition 6.2 (Weak convergence) A sequence ν_k of measures over $\mathbb{S}^\Lambda$ *weakly converges* to measure ν if $\lim_k \nu_k(\mathbf{s}) = \nu(\mathbf{s})$ for all n and for all $\mathbf{s} \in \mathbb{S}^{\Lambda(n)}$. ⌋

In other words, the ν_k-probabilities of any event determined by a finite number of sites converge to its ν-probability. Let us proceed to the definition of ergodicity. There is a linear operator P determined by the transition function $\mathbf{P}(s, \mathbf{r})$ giving $\mu^{t+1} = P\mu^t$:

Definition 6.3 (Markov operator, invariance) Assuming for simplicity, $\Lambda = \mathbb{Z}$, we define the *Markov operator P* on measures by

$$(P\mu)(\mathbf{s}) = \sum_{\mathbf{r}} \prod_{j=-n+1}^{n-1} \mathbf{P}(s_j, (r_{j-1}, r_j, r_{j+1}))\mu(\mathbf{r}) \tag{6.2}$$

where the summation goes over all possible strings $\mathbf{r} = (r_{-n}, \ldots, r_n) \in \mathbb{S}^{2n+1}$. Given a Markov operator, 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 [35]) 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.

Definition 6.4 (Ergodicity) A probabilistic cellular automaton is called *ergodic* if

a) It has only one invariant measure ν.

b) For every possible measure μ^0 over configurations, $\mu^t = P^t\mu^0$ converges weakly to ν.

⌋

In other words, all information about the initial configuration will be eventually lost. A noisy cellular automaton, whenever the set of sites is finite, is a finite Markov chain with all positive transition probabilities. This is ergodic by a well-known elementary theorem of probability theory. If the set of sites is infinite then noisiness does not imply even ergodicity.

Remark 6.5 *No examples are known of noisy cellular automata over an infinite space that satisfy a) but not b). In one dimension and continuous time there is no such example, as shown in [27].* ⌋

The first example of a non-ergodic noisy cellular automaton was given by Toom. (See for example [35].)
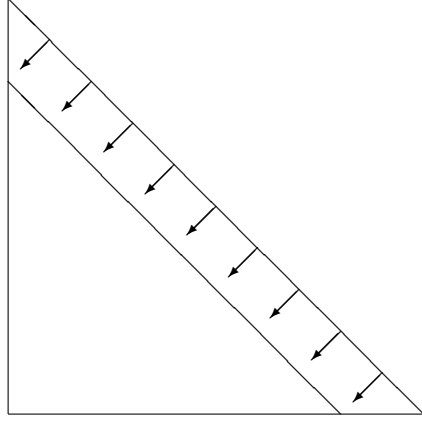
Figure 4: The Toom rule's effect on a large triangular island

Definition 6.6 (Toom rule) A deterministic cellular automaton 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 the Toom rule 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 Section 2.5 and is hence non-ergodic.

Remark 6.7 *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 Section 2.3) such that $R' \subseteq R_\varepsilon$, that is 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.* ⌋

Relaxation time as a measure of information loss What is the relevance can results on infinite cellular automata for 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, consider a (finite or infinite) two-dimensional space Λ. We need a notion of distance for measures.

Definition 6.8 (Variation distance) Let us define the variation distance for measures μ and ν are restricted to $\mathbb{S}^{\Lambda(n)}$:

$$d^n(\mu, \nu) = \sum \{|\mu(\mathbf{s}) - \nu(\mathbf{s})| \text{ for all } \mathbf{s} \in \mathbb{S}^{\Lambda(n)}\}.$$

⌟

The maximum distance 2 means that the two measures have disjoint support. Let us qualify distance even more as a function of the size of the finite two-dimensional spaces that we will be considering:

Definition 6.9 For the set of sites

$$\Lambda_m = \mathbb{Z}_m \times \mathbb{Z}_m$$

where m can be finite or infinite, suppose that the local transition matrix (for simplicity, with nearest-neighbor interaction) gives rise to an ergodic Markov process with Markov operator P_m. Let

$$D_m^n(t) = \bigvee_{\mu, \nu} d^n(P_m^t \mu, P_m^t \nu),$$

where the dependence on the transition matrix is understood but is not shown.

⌟

It is easy to see that this is equal to

$$\bigvee_{\mathbf{r}, \mathbf{s}} d^n(P_m^t \delta_{\mathbf{r}}, P_m \delta_{\mathbf{s}})$$

for the measure $\delta_{\mathbf{s}}$ concentrated on configuration $\mathbf{s}$. The function $D_m^n(t)$ is monotonically decreasing in t since for example for invariant μ,

$$d_m^n(P_m^{t+u} \nu, \mu) = d^n(P_m^t(P_m^u \nu), \mu).$$

P_m is ergodic if and only if for each finite configuration $\mathbf{s}$ and each measure ν we have $P_m^t \nu(\mathbf{s}) \to \mu(\mathbf{s})$. By the weak compactness of the space of measures, this is equivalent to saying that

$$\lim_t D_m^n(t) = 0$$

holds for all n.

Definition 6.10 (Relaxation time) We will call the function

$$r_m(n, \delta) := \sup\{t : D_m^n(t) > \delta\} < \infty$$

the *relaxation time*.

⌟

Ergodicity implies $r_m(n, \delta) < \infty$ for all δ. This function is obviously increasing in n (defined only for $n \leq (m-1)/2$) and decreasing in 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 the unique invariant distribution μ_m is not separated well from any $P_m^t \nu$ it soon becomes fairly close to all of them. Therefore we will generally consider δ fixed.

Relaxation time as a function of space size   If $m < \infty$ then the medium is always ergodic. Assume now that the medium is also ergodic for $m = \infty$. This has a serious implication on the relaxation times for finite m:

Lemma 6.11 *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 ergodic 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 $D_m^n(t)$, it is enough to prove that $D_m^n(r) \leq D_\infty^n(r)$. Take a measure ν over configurations of Λ_m, this will give rise to some measure ν_∞ over configurations of period m in Λ_∞ in a natural way, where ν_∞ is such that for all $n < (m-1)/2$ and all $\mathbf{s} \in \mathbb{S}^{\Lambda(n)}$ we have $\nu_\infty(\mathbf{s}) = \nu(\mathbf{s})$. Then, $r < (m-1)/2 - n$ implies $2n + 1 + 2r < m$ and therefore via (6.2) we have

$$P_m^r \nu(\mathbf{s}) = P_\infty^r \nu_\infty(\mathbf{s}).$$

$\square$

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

Forgetfulness: a variant of ergodicity   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).

Example 6.12 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 the one that is 1 in the even places and 0 in the odd places. For $b = 0, 1$ let μ_b be the measure concentrated on ξ_b, and let P be the linear operator obtained from some 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. ⌋

Definition 6.13 We can call the Markov operator P *strongly not forgetful* if it has two disjoint (weakly) closed translation-invariant sets of measures. ⌋

Our non-ergodic examples will all be strongly not forgetful.

6.2 Information storage and computation

We recall the existence of three-dimensional reliable cellular automata.

Definition 6.14 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 Section 6.1 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. [3]

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

$$\zeta(i, j, n, t) = \xi(n, t). \tag{6.3}$$

Thus, each cell of $CA(Tr)$ is repeated m^2 times, on a whole "plane" (a torus for finite m) in Λ. ⌋

The following has been proved in earlier work:

Proposition 6.15 (Gács-Reif) *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*

---

[3]The papers [19] and [14] use a neighborhood instead of H that makes some proofs easier: the northern, south-eastern and south-western neighbors.

of $CA(Tr)$ over $\mathbb{Z}_N$, if the trajectory ζ of $CA(Tr')$ is defined by (6.3) 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 $w \in \Lambda$

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

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 (6.3)) an arbitrary computation (for example 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, that is it depends on the *size* $N \cdot L$ of the computation. The decoding is even simpler: if a plane of Λ 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 [13]. In its current form, the theorem was proved in [5] using an adaptation of the techniqe of [35].

Theorem 3.1 shows that one-dimensional noisy and strongly not forgetful probability operators exist. Section 5 gave some detail on the kind of constructs going into the proof. The actual proof 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). Once the basic structure (an amplifier, as asserted in Lemma 5.12) is in place, the simulations in it support arbitrary computational actions and allow the formulation of several other theorems. Theorem 3.3 asserts the possibility of storing an infinite string ρ—provided the space is infinite. Now we can give more detail on the nature of the encoding ψ_* used there: it will be one like in Definition 5.8. We don't give a finite version: that one can be seen as a special case of the finite version of Theorem 6.1 below, when the computation is a trivial one outputting the input. Here are some definitions needed before Theorem 6.1. If ρ is a configuration in $\Sigma^{\mathbb{Z}}$ where Σ is the standard computing alphabet, then we say that ρ is *supported by* interval I if $\rho(x) = \# \cdots \#$ or vacant for all $x \notin I$. Recall $Init_{\psi_*}(\rho)$ in Definition 3.6.

We defined the function $h_1(t, c)$ in (3.4).

Theorem 6.1 (Reliable computation in finite or infinite space) *Let Tr be a standard computing transition function over an alphabet Σ, and $\delta > 0$ a constant. There is*

- *a transition function Tr' with a state space $\mathbb{S}$ having fields Input, Output;*

- *a code $\psi_* : \Sigma_0^* \to \mathbb{S}^*$ with $|\psi_*(\rho)| \leq 2|\rho|$,*

- *constants $c_1, c_2 > 1$*

such that the following holds for all sufficiently small ε. Let N be the size of our space, and $\rho \in \Sigma_0^*$ with $2|\rho| < N$, $\varepsilon < \varepsilon_0$. Let $\zeta(x,t)$ be any trajectory of Tr with $\zeta(x,0).\textit{Input} = \rho(x)$ for $x \in [0, |\rho|)$ and filled with $*$'s otherwise. Let η be any trajectory over the space $\mathbb{Z}^N$ of an ε-perturbation of Tr', with $\eta(\cdot, 0) = Init_{\psi_*}(\rho)$. For all sufficiently small ε, for all t for which $\zeta(\cdot, t)$ does not use more space than N, all $t' > t \cdot h_1(t, c_1)$, all $x \in \mathbb{Z}$ we have

$$\mathsf{P}\{\zeta(x,t).\textit{Output} \not\preceq \eta(x,t').\textit{Output}\} < \delta + t'\varepsilon^{h_0(N,c_2)}.$$

As we see the space occupied by our reliable simulation is just a constant times more than the original space (how much larger is $\|\mathbb{S}\|$ than $\|\Sigma\|$). The time delay is more significant: we can recover the computation result $\zeta(x,t)$ only at times $t' > t \cdot h_1(t, c_1)$.

The above theorem strengthens the result of [13] in some ways.

- The need for the decoding of the computation result is eliminated: it appears in the same place as in ζ, due to an economical encoding.

- The encoding of the input depends only on the input, not on the size of the computation. This is achieved by lifting, as indicated after Theorem 3.1.

7 More restrictions on media

Here we impose further requirements on our media

$$Med(\mathbb{S}, \Lambda, B, \textit{Configs}, \textit{Histories}, \textit{Trajs}, (\mathcal{A}_t)_{t \geq 0})$$

from Section 4.3.

7.1 Trajectories

Here, we will show in what form the set of trajectories of a medium will be given. The kind of conditions defining what is a trajectory in a medium will be of the form

$$g(\eta) \leq b,$$

where the function g is an event function as in Definition 2.11.

Example 7.1 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 $g(\eta) = 1$ 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 the interval $[t_0 - \varepsilon, t_0)$ and $\eta(x, t_0) = s$. Thus, the event $g(\eta) = 1$ marks a certain kind of transition in the window.   ⌟

The set *Trajs* of a medium will be given by many such conditions. Let us fix an arbitrary set

$$\mathbb{T} \tag{7.1}$$

called the set of *condition types*. Let

$$Rectangles$$

be the set of bounded space-time rectangles of the form $[a, b) \times (u, v]$. For a type $\alpha \in \mathbb{T}$ and a rectangle V, a *local condition* will be defined as a pair

$$(\alpha, V). \tag{7.2}$$

Two local conditions are *disjoint* when their rectangles are. The conditions defining a trajectory will be given by two functions:

$$g : \mathbb{T} \times Rectangles \times Histories \to \{0, 1\}, \quad h : \mathbb{T} \to [0, 1], \tag{7.3}$$

where $g(alpha, V, \eta)$ is for each α, V an event function in the sense of Definition 2.11, and $b(\alpha) \in [0, 1]$ is a bound: so each condition will have the form

$$\mathsf{E} \, g(\alpha, V, \eta) \le b(\alpha). \tag{7.4}$$

The medium is then defined as

$$M = Med(\mathbb{S}, \Lambda, \mathbb{T}, Configs, Histories, \{\mathcal{A}_t\}, g(\cdot, \cdot, \cdot), b(\cdot)), \tag{7.5}$$

where all arguments inferrable from the context can be omitted. An event defined by $g(alpha, V, \eta)$ occurs when η *violates* a certain local condition. For technical reasons we we always include two special condition types: the *killer* type ω, that does not allow anything, and the *dummy* type δ, that allows everything:

$$g(\omega, V, \eta) = b(\omega) = 0, \quad g(\delta, V, \eta) = b(\delta) = 1. \tag{7.6}$$

For an empty "rectangle" $\emptyset$ we stipulate $g(\alpha, \emptyset, \eta) = 0$ for all $\alpha \ne \delta$, and $g(\delta, \emptyset, \eta) = 1$.

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 or infinite spaces Λ 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 $M_1, M_2, \dots$ then $g_i(), b_i()$ automatically refer to M_i.

Recall the definition $\boldsymbol{\pi}_t$ from (2.4).

Definition 7.2 (Trajectory) A random history η will be called a *trajectory* of medium M given in (7.5) if for each time $u \geq 0$, for each set of disjoint local conditions

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

with $(\forall i) \inf \boldsymbol{\pi}_{\mathrm{t}} V_i \geq u$ (where $\inf \emptyset = \infty$), we have

$$\mathsf{E}\left[\prod_i g(\alpha_i, V_i, \eta) \,\middle|\, \mathcal{A}_{<u}\right] \leq \prod_i b(\alpha_i). \tag{7.7}$$

⌐

This says that we can multiply the probabilities of violating local conditions $\mathsf{E}\, g(\alpha_i, V_i, \eta) \leq b(\alpha)$ in disjoint windows, in other words: these violations happen "independently". The media we are interested in have the following three types of condition:

1) One type recognizes in V a certain event called "damage", and has $b = \varepsilon$ for some small ε;

2) One type recognizes the event that the transition does not occur there according to a certain transition function. This has $b = 0$, thus prohibiting this possibility.

3) One type corresponds to some "coin tossing" event, and has $b = 0.5 + \varepsilon'$ for a suitable ε'.

The following example will show how a discrete-time probabilistic cellular automaton (PCA) fits into this framework.

Example 7.3 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 $\alpha(s, \mathbf{r})$. The condition of this type will say that the transition rules are obeyed at times of the form nT. Namely for cell x, time t of the form nT with $n > 0$, let

$$g\big(\alpha(s, \mathbf{r}),\ \{x\} \times (t - T, t],\ \eta\big)$$
$$= \{\eta(x + iB, t - T/2) = r_i\ (i = -1, 0, 1),\ \eta(x, t) = s\}, \tag{7.8}$$
$$b(\alpha(s, r)) = \mathbf{P}(s, \mathbf{r}). \tag{7.9}$$

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

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

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.

⌐

The new conditions introduced below are "loosened-up" versions of the above ones since it is not always desirable to be as restrictive. Instead of a bound on the probability of getting into a local state s, we may just want a bound on the probability of hitting a certain set of local states K'' once it is known that we were going to hit a certain bigger set of local states K'.

Definition 7.4 (Local state subset functions) In 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

$$K(\mathbf{r}) \subseteq \{s \neq r_0 : \mathbf{P}(s, \mathbf{r}) > 0\}. \tag{7.10}$$

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

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

For all $K', K'' \in \mathbf{K}$ with $K' \supseteq K''$ (that is $K'(\mathbf{r}) \supseteq K''(\mathbf{r})$ 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 maximum conditional probability, over all $\mathbf{r}$, of getting into the set $K''(\mathbf{r})$ provided we get into the set $K'(\mathbf{r})$.

In case of a continuous-time cellular automaton $CCA(\mathbf{R}, B)$, for any functions $K, K', K'' \in \mathbf{K}$ with $\mathbf{R}(K, \mathbf{r}) = \sum_{q \in K} \mathbf{R}(q, \mathbf{r})$ we define the corresponding quantity

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

⌐

Let us form local conditions with the help of these concepts:

Example 7.5 In case of a probabilistic cellular automaton $PCA(\mathbf{P}, B, T)$, for each pair of such functions $K', K'' \in \mathbf{K}$ with $K' \supseteq K''$ let us form the type

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

For each cell x and times $u \leq v$ 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 $\eta(x,t) \in \overline{K}'(x, t-T, \eta)$ and in it, we have $\eta(x,t) \in \overline{K}''(x, t-T, \eta)$. Also, let $b(\alpha) = c(K', K'')$. The condition of type α bounds the probability that as soon as we get into the set K' we also get into the set K''. ⌐

The proposition below is an immediate consequence of the definitions:

Proposition 7.6 *If M is the medium defined by the conditions above then all trajectories of $PCA(\mathbf{P}, B, T)$ are trajectories of M.*

Example 7.7 In case of a continuous-time cellular automaton $CCA(\mathbf{R}, B)$ we define each $\alpha(K', K'')$, $V = \{x\} \times (u, v]$ and $g(\alpha, V, \eta)$ as in Example 7.5, but now for all rational $u < v$. ⌐

Theorem 7.1 *Every trajectory of $CCA(\mathbf{R}, B)$ is a trajectory of the medium defined in the above example.*

Proof sketch. As in the time-discretization in Section 2.4, 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 $s \neq r_0$ and $1 - \delta \sum_{s' \neq r_0} \mathbf{R}(s', \mathbf{r})$ otherwise. Then a transition to the limit $\delta \to 0$ in Proposition 7.6 completes the proof. The transition uses the fact that trajectories of $M(\delta)$ converge in distribution to trajectories of $CCA(\mathbf{R}, B)$. □

Of course in both Proposition 7.6 and in Theorem 7.1, instead of the first time with η switching into K' we could have asked about the second time, or the third time, etc.

7.2 Strong trajectories

In one part of our construction and proof, for self-organization, in Section 21.2, we need a seemingly stronger condition on trajectories than the one in Definition 7.2: namely, we must consider sets of local conditions that are randomly chosen. Of course, they cannot be chosen completely randomly, as then we could just choose the places where the faults occur. But we will allow conditions chosen in a non-anticipating way, that is via *stopping times*. Recall the notion of a set of condition types $\mathbb{T}$ in (7.1), (7.2) and (7.3).

Here, we will specialize, for our needs, the notion of stopping times from the theory of stochastic processes. Let $\mathcal{A}_t$ be a filtration over a probability space, as in Definition 2.12.

Definition 7.8 (Stopping time) A real random variable σ with values in $[0, \infty)$ is a *stopping time* 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 $A \cap \{\sigma \leq t\} \in \mathcal{A}_t$. ⌐

Definition 7.9 (Constraint process) For some time interval $J \subset (0, \infty]$, assume that there is a stopping time $\sigma \in J$, and random variables $\alpha(\eta) \in \mathbb{T}$,

$V(\eta) \in Rectangles$ measurable in $\mathcal{A}_\sigma$, with

$$\boldsymbol{\pi}_{\mathrm{t}} V^\sigma(\eta) \subseteq J \cap [\sigma, \infty].$$

The tuple $(J, \sigma, \alpha(\eta), V(\eta))$ will be called a *constraint process*. ⌟

Definition 7.10 (Strong trajectory) A random history η will be called a *strong trajectory* if for each system of disjoint constraint processes

$$(J, \sigma_i, \alpha_i(\eta), V_i(\eta))_{i \in N}$$

(where $V_i(\eta)$ are disjoint for each η), h an event function over $\mathcal{A}_{<\boldsymbol{\pi}_{\mathrm{t}} J}$, we have

$$\mathsf{E}\left(h \prod_i g(\alpha_i(\eta), V_i(\eta), \eta)\right) \leq \mathsf{E}\, h\, \mathsf{E} \prod_i b(\alpha_i(\eta)). \tag{7.11}$$

⌟

This says, just as for ordinary trajectories, that we can multiply the probabilities of violating local conditions in disjoint windows—but requires that this be even true if the condition types and windows are chosen by random stopping rules. It may seem strange that there is dependence on a random α_i on both sides, but note that $\alpha_i(\eta)$ depends only on what happened before σ_i, while $g(\alpha_i(\eta), V_i(\eta), \eta)$ depends on what happens in $V_i(\eta)$, whose bottom is after σ_i.

Proposition 7.11 *The trajectories of the cellular automaton $PCA(\mathbb{P}, B, T)$ of Example 7.5 are strong.*

Proof sketch. The statement follows by a standard argument from the strong Markov property of the Markov process $PCA(\mathbf{B}, B, T)$ (see [28]). □

Proposition 7.12 *Proposition 7.6 and Theorem 7.1 can be strengthened: all trajectories of $PCA(\mathbf{B}, B, T)$ and $CCA(\mathbf{R}, B)$ are* strong *trajectories of the corresponding media.*

The proofs are routine.

7.3   Canonical simulations

We will restrict the kind of simulations among media from its general form given in Definition 4.23 to a kind of mapping that sets up some relations between the local conditions of the media and makes it easier to prove it is a simulation.
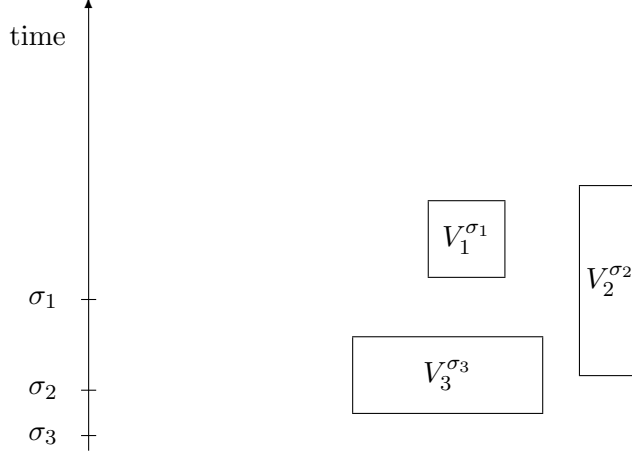
Figure 5: A system of disjoint random local conditions

Definition 7.13 Let M, M^* be media where the functions g, b define the constraints on M, and g^*, b^* define those for M^*, and let φ^* be a mapping from the histories of M into those of M^*: we will write $\eta^* = \varphi^*(\eta)$. We say that φ^* is a *canonical simulation map* if the following holds. Let (α, V) be any local condition for M^* with $\alpha \in \mathbb{T}^*$. Then there is a system of constraint processes

$$\left(\boldsymbol{\pi}_{\mathrm{t}} V, \tau_{i,j}, \beta_{i,j}(\eta), W_{i,j}(\eta) \right) \tag{7.12}$$

with $i = 1, \ldots, n(\alpha, V)$, $j = 1, \ldots, k(\alpha, V, i)$ with the following properties. For each fixed i, the subsystem containing elements for that i forms a system of disjoint constraint processes with $W_{i,j}(\eta) \subseteq V$, and for almost all η we have

$$g^*(\alpha, V, \eta^*) \le \sum_i \prod_j g(\beta_{i,j}(\eta), W_{i,j}(\eta), \eta),$$

$$b^*(\alpha) \ge \sum_i \prod_j b(\beta_{i,j}(\eta)). \tag{7.13}$$

⌟

Theorem 7.2 *If φ^* is a canonical simulation map then for each strong trajectory η of M, the expression $\varphi^*(\eta)$ defines a strong trajectory of M^*.*

Proof. Let η be a strong trajectory of M and

$$(J, \sigma_l, \alpha_l(\eta^*), V_l(\eta^*))_{l \in N} \tag{7.14}$$

a system of disjoint random local conditions in M^*. Let h be an event function measurable in $\mathcal{A}_{<\inf J}$. We must show

$$\mathsf{E}\left(h\prod_l g_l^*(\alpha_l(\eta^*), V_l(\eta^*), \eta^*)\right) \leq \mathsf{E}\, h\, \mathsf{E}\prod_l b^*(\alpha_l^{\sigma_l}(\eta^*)).$$

By the assumption, for each value (a, v) of the random $(\alpha_l(\eta^*), V_l(\eta^*))$ there is a system of constraint processes

$$\left(\boldsymbol{\pi}_t v, \tau_{l,i,j}(a, v), \beta_{l,i,j}(a, v, \eta), W_{l,i,j}(a, v, \eta)\right) \tag{7.15}$$

for $i = 1, \ldots, n(a, v)$, $j = 1, \ldots, k(a, v, i)$, with the properties defined for canonical simulations, so $\tau_{l,i,j} \in \boldsymbol{\pi}_t v$. This implies

$$\tau_{l,i,j}(\alpha_l(\eta^*), V_l(\eta^*)) \geq \sigma_l \tag{7.16}$$

for all l, i, j. We will use the shorthand notation

$$\bar{W}_{l,i,j}(\eta) = W_{l,i,j}(\alpha_l(\eta^*), V_l(\eta^*), \eta), \tag{7.17}$$

and $\bar{\tau}_{l,i,j}$, $\bar{\beta}_{l,i,j}(\eta)$ means similarly not showing the dependence on α_l, V_l.

Claim *For each l, i, j, the tuple*

$$(J, \bar{\tau}_{l,i,j}, \bar{\beta}_{l,i,j}(\eta), \bar{W}_{l,i,j}(\eta))$$

is a constraint process.

Proof. Let us first show that $\bar{\tau}_{l,i,j}$ is a stopping time. Let

$$E_{a,v} \Leftrightarrow (\alpha_l(\eta^*), V_l(\eta^*)) = (a, v).$$

We have

$$\bar{\tau}_{l,i,j} \leq t \Leftrightarrow \bigvee_{a,v} \tau_{l,i,j}(a, v) \leq t \wedge E_{a,v}. \tag{7.18}$$

From (7.16) we get that $\bar{\tau}_{l,i,j} \leq t$ implies $\sigma_l \leq t$, therefore

$$\bar{\tau}_{l,i,j} \leq t \wedge E_{a,v} \Leftrightarrow \bar{\tau}_{l,i,j} \leq t \wedge \sigma_l \leq t \wedge E_{a,v}.$$

By the definition of stopping times, $\{\tau_{l,i,j}(a, v) \leq t\} \in \mathcal{A}_t$. Because (7.14) is a constraint process for each l, we have also

$$\{\sigma_l \leq t \wedge E_{a,v}\} \in \mathcal{A}_t, \tag{7.19}$$

hence the countable union (7.18) is also in $\mathcal{A}_t$. Similarly, for all $(b, w) \in \mathbb{T} \times \textit{Rectangles}$, the event

$$\bar{\tau}_{l,i,j} \leq t \wedge (\bar{\beta}_{l,i,j}(\eta), \bar{W}_{l,i,j}(\eta)) = (b, w) \tag{7.20}$$

can be written as

$$\bigvee_{a,v} \big(\sigma_l \leq t \wedge E_{a,v} \wedge \bar{\tau}_{l,i,j} \leq t \wedge (\beta_{l,i,j}(a, v, \eta), W_{l,i,j}(a, v, \eta)) = (b, w) \big).$$

We have seen (7.19), and because (7.15) is a system of constraint processes, also

$$\{ \bar{\tau}_{l,i,j} \leq t \wedge (\beta(a, v, \eta), W_{l,i,j}(a, v, \eta)) = (b, w) \} \in \mathcal{A}_t,$$

therefore (7.20) is in $\mathcal{A}_t$. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\square$

By (7.13) we have, with $\bar{\alpha}_l = \alpha_l(\eta*)$, $\bar{V}_l = V_l(\eta^*)$ and the notation in (7.17)

$$g^*(\bar{\alpha}_l, \bar{V}_l, \eta^*) \leq \sum_{i=1}^{n} \prod_{j=1}^{k} g(\bar{\beta}_{l,i,j}(\eta), \eta), \bar{W}_{l,i,j}(\eta), \eta). \tag{7.21}$$

Here we did not make n dependent on $\bar{\alpha}_l$, $\bar{V}_l$, nor k dependent on these and i, setting them to the maxima of all the possible values. Recall the special types ω, δ defined in (7.6). Whenever $i > n(\bar{\alpha}_l, \bar{V}_l)$, we set $\beta_{l,i,j} = \omega$, meaning that the ith additive term is omitted. And whenever $j > k(\bar{\alpha}_l, \bar{V}_l, i)$, we set $\beta_{l,i,j} = \delta$, meaning that the jth multiplicative factor is omitted. Now

$$\prod_l b^*(\bar{\alpha}_l) \geq \prod_l \sum_i \prod_j b(\bar{\beta}_{l,i,j}(\eta)) = \sum_{i(\cdot)} \prod_{l,j} b(\bar{\beta}_{l,i(l),j}(\eta))$$

where the last sum is over all functions $i(\cdot)$ with $i(l) \in [1, n]$. Similarly, by (7.21)

$$g^*(\bar{\alpha}_l, \bar{V}_l, \eta^*) \leq \sum_{i(\cdot)} \prod_{l,j} g(\bar{\beta}_{l,i,j}(\eta), \bar{W}_{l,i,j}(\eta), \eta).$$

Therefore it is sufficient to show for a fixed function $i(\cdot)$,

$$\mathsf{E} \big(h \prod_{l,j} g(\bar{\beta}_{l,i(l),j}(\eta), \bar{W}_{l,i(l),j}(\eta), \eta) \big) \leq \mathsf{E} \, h \, \mathsf{E} \prod_{l,j} b(\bar{\beta}_{l,i(l),j}(\eta)).$$

For any l, i, the rectangles $\bar{W}_{l,i,j}(\eta)$ defined in (7.17) are all disjoint, and belong to $V_l(\eta^*)$. Also the sets $V_l(\eta^*)$ are disjoint, therefore all rectangles $\bar{W}_{l,i(l),j}(\eta)$ are disjoint. The Claim has shown that the system

$$\big(J, \bar{\tau}_{l,i(l),j}, \bar{\beta}_{l,i(l),j}(\eta), \bar{W}_{l,i(l),j}(\eta) \big)_{l \in N}$$

is a system of disjoint constraint processes for M, and hence the proof is finished by the strong trajectory property of η. □

From for ease of reading, we omit the qualifier "strong", but all trajectories we will deal with are strong trajectories.

Here is a trivial but illustrative example.

Example 7.14 Let M_1 be the ε-perturbation $CA_\varepsilon(Tr, B_1, T_1)$ of the cellular automaton with $\mathbb{S}_1 = \{0, 1\}$ and the trivial transition function $Tr(x, y, z) = 0$ for all x, y, z. Then η is a trajectory if for $t > 0$, $\eta(x, t)$ are random variables such that for any u and any finite set S of space-time points (x, t) with $t \geq u$, the conditional probability that $\eta(x, t) = 1$ in all $(x, t) \in S$ no matter how the $\eta(x', t')$ are fixed for all $t' < t$, is bounded by $\varepsilon^{|S|}$. The set of constraint types consists of a single element, $\mathbb{T}_1 = \{\alpha_1\}$ with $b_1(\alpha_1) = \varepsilon$. For the event function we have $g_1(\alpha_1, V, \eta) = 1$ if and only if for some (x, t) of the form (mB_1, nT_1) with integers m, n and $n > 0$, $V = [x, x + B_1) \times (t - T_1, t]$, and $\eta(x, t) = 1$.

Let M_2 be a medium with $\mathbb{S}_2 = \{0, 1\}$. For integers $Q_1, U_1 > 0$ let $B_2 = Q_1 B_1$, $T_2 = U_1 T_1$. The histories will be of the form $\eta(x, t)$ with $x = mB_2$, $t = nT_2$ for $n \geq 0$. The set of constraint types consists of a single element, $\mathbb{T}_2 = \{\alpha_2\}$ with $b_2(\alpha_2) = 3Q_1 U_1 \varepsilon^2$. For the event function we have $g_2(\alpha_2, V, \eta) = 1$ if and only if for some (x, t) of the form (nB_1, nT_1) with integers m, n and $n > 0$, $V = [x - B_2, x + 2B_2) \times (t - 2T_2, t]$, and $\eta(x, t) = 1$.

Let us define the simulation φ^* as follows. If $\eta^* = \varphi^*(\eta)$ then $\eta^*(x, t) = 1$ if and only if (x, t) has the form (mB_2, nT_2) with integers m, n and $n > 0$, and in $V = [x - B_2, x + 2B_2) \times (t - 2T_2, t]$ there are at least two elements $(x_1, t_1) \neq (x_2, t_2)$ with $\eta(x_i, t_i) = 1$. To show that this is a simulation we define the system of local conditions

$$\big(\beta_{i,j}(\alpha_2, V), W_{i,j}(\alpha, V)\big)$$

as follows. These are nonempty if and only if for some (x, t) of the form (mB_2, nT_2) with integer m, n and $n > 0$, $V = [x - B_2, x + 2B_2) \times (t - 2T_2, t]$. In this case, let $P = \{p_1, \ldots, p_N\}$ be a list of all distinct pairs $\{(x_1, t_1), (x_2, t_2)\}$ in V, where $N = \binom{6Q_1 U_1}{2}$. For $p_i = \{(x_{i,1}, t_{i,1}), (x_{i,2}, t_{i,2})\}$, let $\beta_{i,j}(\alpha_2, V) = \alpha_1$ for all i, j, and

$$W_{i,j}(\alpha_2, V) = [x_{i,j}, x_{i,j} + B_1) \times (t_{i,j} - T_1, T_{i,j}].$$

Then $g_2(\alpha_2, V, \eta^*)$ is the event that $\eta(x, t) = 1$ for at least two pairs $(x, t) \in V$, and this is upper-bounded by

$$\sum_i \prod_j g_1(\alpha_1, W_{i,j}(\alpha_2, V))$$

as required. Also

$$b_2(\alpha_2) = 3Q_1 U_1 \varepsilon^2 > N\varepsilon^2 = \sum_{i=1}^{N} \prod_{j=1}^{2} g_1(\alpha_1, W_{i,j}(\alpha_2, V))$$

as required. ⌟

Frequency of switching times   Adding certain conditions does not really change some media.

Definition 7.15 (Injective canonical simulation) We will 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.

We will illustrate canonical simulations on the the following example. The simulating medium M_1 is the *CCA* of Example 7.7. The conditions to add impose lower and upper bounds on the frequency of switching times of M_1. Recall the set of functions **K** in Definition 7.4.

Definition 7.16 (Subset-function switching times) 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 history η a K-*switching-time* of a cell x if for all $t < u$ sufficiently close to u we have $\eta(x, t) \notin \overline{K}(x, t, \eta)$ and $\eta(x, u) \in \overline{K}(x, t, \eta)$, that is we have just jumped into the target set determined by $K(\cdot)$. For some rational constant $D > 0$, integers $k \geq 0$ and $j = -1, 1$ let

$$g\big(\alpha(K, D, k, j), \; \{x\} \times (u, u + D], \; \eta\big) \tag{7.22}$$

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. ⌟

The statement below estimates the probability of having a number of switches significantly below the lower bound on their expected number, or above the upper bound.

Proposition 7.17 *Let us define, with $a_j = a_j(K)$ and every possible $h > 0$:*

$$b(\alpha(K, D, a_j D + jh\sqrt{D}, j)) = 1 \wedge \frac{a_j}{h^2}, \qquad (7.23)$$

except that in case $j = 1$ and $a_1 D + h\sqrt{D} = 1$ we set it to $a_1 D$ when the latter is smaller. Let $b(\alpha(K, D, k, j)) = 1$ for all other k. For each transition rate matrix $\mathbf{R}$ and function $K \in \mathbf{K}$, the trajectories of $CCA(\mathbf{R}, B)$ are also trajectories of the medium obtained by adding all these conditions defined by $\alpha(K, D, \dots)$, $b(\alpha(\dots))$.

Proof. The number of K-switching times during $(u, u + D]$ is dominated by a Poisson random variable X_1 with parameter $a_1 D$. We use approximation as in Definition 2.17. Thus, for a sufficiently small $\delta > 0$, let $M_\delta = PCA(\mathbf{P}, B, \delta)$ with $\mathbf{P}(s, \mathbf{r}) = \mathbf{R}(s, \mathbf{r})\delta$ when $s \neq r_0$ and $1 - \delta \sum_{s' \neq r_0} \mathbf{R}(s', \mathbf{r})$ otherwise. If $A(D, \delta)$ is the random variable counting the number of switches in M_δ during $(u, u + D]$ then it is clearly dominated by the random variable $B(D, \delta)$ that is the number of switches when a switch takes place with probability $a_1 \delta$ every time. As $\delta \to 0$ the distribution of this variable $B(D, \delta)$ converges to that of X_1, which then dominates the number of K-switching times. Similarly, the number of K-switching times dominates a Poisson random variable X_{-1} with parameter $a_{-1} D$. The mean and variance of X_1 is $a_1 D$, hence by the Chebyshev inequality

$$\mathsf{P}\{X_1 \geq a_1 D + h\sqrt{D}\} \leq \frac{a_1 D}{h^2 D} = \frac{a_1}{h^2}.$$

Similarly, $\mathsf{P}\{X_{-1} \leq a_{-1} D - h\sqrt{D}\} \leq \frac{a_1}{h^2}$. In the special case when $a_1 D + h\sqrt{D} = 1$ we have the simpler bound $\mathsf{P}\{X_1 \geq 1\} = 1 - e^{-a_1 D} < a_1 D$. $\qquad \square$

7.4 Primitive variable-period media

The media defined here will serve as the bottom level of our amplifier.

Definition 7.18 (Dwell period lower bound) We say that the number $T_\bullet > 0$ is a *dwell period lower bound* of a history 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 $M_0, M_1, \dots$ we will construct eventually, all abstract media $M_1, M_2, \dots$

will have a dwell period lower bound, only the continuous-time probabilistic cellular automaton 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).

Definition 7.19 (Primitive variable-period medium) The medium

$$Prim\text{-}var(Tr, B, T_\bullet, T^\bullet, \varepsilon).$$

is defined as follows. 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 2.15, is a record with two fields, *Det* and *Rand* where *Rand* consists of a single bit. To simplify the upper-bounding of dwell periods, we assume that the transition function has the property

$$Tr(r_{-1}, r_0, r_1).Det \neq r_0.Det, \tag{7.24}$$

that is the deterministic part of a cell will change at every transition. In a more general setting later this will be called "time marking", see Condition 10.9. For a history η, site x let $a \geq 0$ be a rational number. If $a > 0$ then let $\sigma_1 = \sigma_1(x, a, \eta)$, and σ_2 be the first and second switching times $t > a$ of η in x. If $a = 0$ then let $\sigma_1 = 0$ and let σ_2 be the first switching time > 0. Let us list the different types of local condition.

   a) Conditions of type $\alpha(dw\text{-}p\text{-}lb)$ imposing $T_\bullet$ as a dwell period lower bound. We have $b(\alpha(dw\text{-}p\text{-}lb)) = \varepsilon$, and

$$g\big(\alpha(dw\text{-}p\text{-}lb),\ \{x\} \times (a, a + T^\bullet],\ \eta\big)$$

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

   b) Conditions of type $\alpha(dw\text{-}p\text{-}ub)$ imposing $T^\bullet$ as a dwell period upper bound. We have $b(\alpha(dw\text{-}p\text{-}ub)) = \varepsilon$, and

$$g\big(\alpha(dw\text{-}p\text{-}ub),\ \{x\} \times (a, a + T^\bullet],\ \eta\big)$$

   is the event 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 $b(\alpha(comput)) =$

ε, 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 (4.2).

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:

$$b(\alpha(coin, j)) = 0.5 + \varepsilon,$$
$$g\big(\alpha(coin, j),\ \{x\} \times [a, a + 2T^\bullet],\ \eta\big) = \{\eta(x, \sigma_2).Rand = j\}.$$

When there is no σ_2 then $g(\alpha(coin, j), \cdot, \cdot) = 0$, so this condition has no effect.

⌟

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 7.20 All trajectories of the ordinary deterministic cellular automaton $CA(Tr, B, T)$ are also strong trajectories of $Prim\text{-}var(Tr, B, T, T, 0)$.

⌟

Theorem 7.3 (Simulation by CCA) *For medium*

$$M = Prim\text{-}var(Tr, 1, T_\bullet, 1, \varepsilon)$$

with (7.24) *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 strong trajectory of* M *for each trajectory* η *of* $CCA(\mathbf{R}, 1)$.

Proof. The proof for trajectories is a straightforward application of Proposition 7.17, and could be modified with some formalism to hold also for strong trajectories. For an appropriate n to be chosen later, let $\mathbb{S}' = \mathbb{S}^3 \times \{0, \ldots, 3n - 1\}$. If $s' = (\mathbf{s}, i)$ where $\mathbf{s} = (s_{-1}, s_0, s_1)$ then let $\pi(s') = s_0$. Let us define the transition rates $\mathbf{R}_n(\cdot, \cdot)$ with $\sum_{s'} \mathbf{R}_n(s', \mathbf{r}) = 3n$. Let $\mathbf{K} = \mathbf{K}_n$ be the set of functions as defined in (7.10) belonging to this set of states, assuming that all transition rates are positive. Choose the function $K \in \mathbf{K}$ as follows. For

$\mathbf{r} \in (\mathbb{S}')^3$ where $r_0 = (\mathbf{q}, i)$, where $\mathbf{q} = (q_{-1}, q_0, q_1)$, let $K(\mathbf{r})$, be the set of those $(\bar{\mathbf{q}}, i + 1)$ for which $\bar{\mathbf{q}}$ is given as follows. If $i \neq n - 1, 3n - 1$ then $\bar{\mathbf{q}} = \mathbf{q}$. If $i = n - 1$ then $\bar{q}_j = \pi(r_j)$, $j = -1, 0, 1$. Suppose $i = 3n - 1$: then $\bar{q}_j = q_j$ for $j = -1, 1$ and $\bar{q}_0.\mathit{Det} = Tr(\mathbf{q})$. This completes the definition of K which is a one-element set whenever $i \neq 3n - 1$; in the latter case it is a two-element set, with the two elements differing by the value of $\bar{q}_0.\mathit{Rand}$.

Now $\mathbf{R}_n$ is given as follows. $\mathbf{R}_n(s, \mathbf{r}) = 0$ for all $s \neq K(\mathbf{r})$ and $\mathbf{R}(K(\mathbf{r}), \mathbf{r}) = 3n$. This determines $\mathbf{R}_n(s, \mathbf{r})$ for all cases when $K(\mathbf{r})$ is a one-element set. If $K(\mathbf{r})$ is a two-element set then the transition rates into each of these elements are equal. This completes the definition of $\mathbf{R}_n$. Thus, in a trajectory η of $\mathbf{R}_n$, each dwell period of $\pi(\eta)$ consists of $3n$ small dwell periods of η. There is a change only in the nth and $3n$th transition. In the n-th transition, cell x learns the current represented state $\pi(\eta(x + j, t))$ of its two neighbors. In the $3n$-th transition, it switches its own represented state according to the transition probabilities corresponding to the triple of $\mathbb{S}$ states it currently knows about. The actual transition rates $\mathbf{R}$ differ from $\mathbf{R}_n$ in an arbitrary way by some small amount $< \varepsilon'$. Let us show that we can choose first n and then ε' in such a way that the process $\pi(\eta)$ becomes a trajectory of M.

Let $K_0(\mathbf{r}) = \mathbb{S} \setminus (K(\mathbf{r}) \cup \{r_0\})$. A trajectory η of $\mathbf{R}_n$, in any switching point (x, t) makes a switch into the set $K(x, t-, \eta)$. If a trajectory η of $\mathbf{R}$ in a switching point (x, t) does not switch into $K(x, t-, \eta)$ we will this a *faulty switch*. Then it switches into $K_0(x, t-, \eta)$, but with a rate smaller than $\varepsilon'|\mathbb{S}|$. According to the part of Proposition 7.17 referring to bounds of the kind (7.23), by making ε' small enough we can always achieve that the following condition type $\alpha(\mathit{fault})$ can be added to the others:

$$b(\alpha(\mathit{fault})) = c\varepsilon,$$

where c is any positive constant chosen in advance and

$$g\big(\alpha(\mathit{fault}),\ \{x\} \times (a, a + 2],\ \eta\big)$$

is the event that a faulty switch occurs in x during $(a, a + 2]$. Therefore in all the local conditions for the simulated $\mathit{Prim\text{-}var}()$, when bounding the probabilities in local conditions, we can confine ourselves to the case that no faulty switch occurs in x during $(a, a + 2]$.

Consider the conditions in a). Since we can exclude faulty switches in the time period $(a, a + 1]$, a trajectory $\eta(x, t)$ of $\mathbf{R}$ will be such that when denoting $\eta(x, t) = (\mathbf{r}(x, t), i(x, t))$, then $i(x, t)$ increases by 1 modulo $3n$ in every switch. Therefore there are exactly $3n$ switching times of η between any two switching times of $\pi(\eta)$. Let $d = (1 - T_\bullet)/2$. If $\pi(\eta)$ has two switching

times $\sigma_1 < \sigma_2$ during $(a, a+1]$ that are closer then $T_\bullet$ then either they are in $(a, a+1-d]$ or in $(a+d, a+1]$. Consider the first case, the second case is similar. Then η has at least $3n$ switching times during $(a, a+1-d]$. By choosing n large enough and noting that the rate of $\mathbf{R}$ is (arbitrarily close to) $3n$, we can use Proposition 7.17 to bound the probability of this event by $\varepsilon/3$. The probability for the other case can also bounded by $\varepsilon/3$ and the probability of a faulty switch can also bounded by $\varepsilon/3$. We thus get the bound ε in the conditions a) by canonical simulation. A similar reasoning handles condition b).

Now consider condition c). Again, we can assume that no faulty switch occurs during $(a, a+2]$. There are exactly $3n$ switching times of η between σ_1 and σ_2. According to the definition of $\mathbf{R}_n$, the transition will occur according to Tr, with the neighbor values that were read at the nth switching time. Now an argument similar to the proof of a) shows that this nth switching time is in $(\sigma_1, \sigma_2 - T_\bullet/2]$.

Finally, the possibility of adding condition d) on $\pi(\eta)$ is a straightforward consequence of Theorem 7.1. Indeed, let $K'(\mathbf{r}) = \mathbb{S} \setminus \{r_0\}$ and let $K''(\mathbf{r})$ be the set of elements $(\mathbf{q}, 0) \in K'(\mathbf{r})$. Thus, jumping into K'' means making a switch in $\pi(\eta)$. At that switch, the transition rates into the possibility $q_0.\mathsf{Rand} = 1$ and $q_0.\mathsf{Rand} = 0$ are within ε' of $1/2$. Therefore Theorem 7.1 is applicable. □

Generalizing the results to continuous time The first publication showing the possibility of reliable computation with a continuous-time medium (in two dimensions) is [38]. Here, we formulate the new results for variable-period one-dimensional information storage and computation. The following theorems generalize Theorems 3.1, 3.3 and 3.4 allowing a variable-period medium $Prim\text{-}var(Tr', 1, T_\bullet, 1, \varepsilon)$ with $T_\bullet = 0.5$ in place of the perturbed discrete-time automaton $CA_\varepsilon(Tr')$. See Corollary 7.21 below for the version for interacting particle systems.

Theorem 7.4 *Theorem 3.1 also holds when we allow a perturbed primitive variable-period medium $Prim\text{-}var(Tr', 1, 0.5, 1, \varepsilon)$ with strong trajectories instead of a perturbed discrete-time cellular automaton.*

Theorem 7.5 *Theorem 3.3 also holds (with the same encoding ψ_*) and strong trajectories when we allow a primitive variable-period perturbation instead of a discrete-time perturbation.*

Theorem 7.6 *Theorem 6.1 holds (with the same encoding ψ_*) also with $Prim\text{-}var(Tr', 1, 0.5, 1, \varepsilon)$ and strong trajectories in place of $CA_\varepsilon(Tr')$.*

Theorem 7.3 implies the following.

Corollary 7.21 *In Theorems [7.4], [7.5], [7.6], we can replace Tr' with a rate matrix $\mathbf{R}$, and $Prim\text{-}var(Tr', 1, T_\bullet, 1, \varepsilon)$ with a CCA with rates coming from an arbitrary ε-perturbation of $\mathbf{R}$. This proves, in particular, Theorem [3.2].*

In what follows when we talk about trajectories we will always mean strong trajectories.

8 Synchronization

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.

Definition 8.1 (Totally asynchronous cellular automata) We 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

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

⌋

To analyze synchronization, some more concepts are needed.

Definition 8.2 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

$$\eta(\cdot, t+1) = Tr(\eta(\cdot, t), U). \tag{8.1}$$

Definition 8.3 (Update set) Let the *update set*

$$U(t, \eta) \tag{8.2}$$

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 the trajectory η.

Notation 8.4 (Indicator function) For any set A, let us use the indicator function

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

⌟

Definition 8.5 (Effective age) For given history η, we define the function $\tau(x, t) = \tau(x, t, \eta)$ as follows:

$$\begin{aligned} \tau(x, 0) &= 0, \\ \tau(x, t+1) &= \tau(x, t) + \chi(x, U(t, \eta)). \end{aligned} \tag{8.3}$$

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. ⌟

Definition 8.6 (Invariant histories) 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

$$\eta(x, t) = \zeta(x, \tau(x, t, \eta), \xi). \tag{8.4}$$

⌟

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. The following notation will be useful:

Notation 8.7 Denote

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

⌟

Definition 8.8 (Commutative transition) 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 [15] proves the theorem that if a transition function is commutative then it has invariant histories. In Theorem 8.1 below, we will give a simple example of a universal commutative transition function. For that example, the theorem can be proved much easier.

Notation 8.9 We will denote the smallest absolute-value remainders

$$b \operatorname{amod} m \tag{8.5}$$

with respect to a positive integer $m > 2$, defined by the requirement $-m/2 < b \operatorname{amod} m \leq m/2$.
⌟

Theorem 8.1 (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.

Define $s' = Tr_1(s_{-1}, s_0, s_1)$. If there is a $j \in \{-1, 1\}$ with $(s_j.Age - s_0.Age) \operatorname{amod} U < 0$ (that is some neighbor lags behind) then $s' = s_0$ that is 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
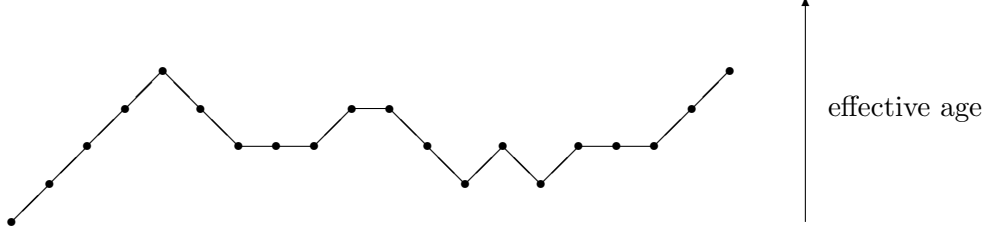
Figure 6: The Marching Soldiers scheme. The effective age of neighbor sites differs by at most 1.

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. $\square$

Definition 8.10 (Marching soldiers) The commutative medium of the above proof will be called the *marching soldiers* scheme. ⌐

The name comes from the similarity of its handling of the *Age* field to a chain of soldiers marching ahead in which two neighbors do not want to be separated by more than one step.

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 (amount of communication with a neighbor in one step) smaller than $\|\mathbb{S}\|$.

Corollary 8.11 (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 $T_{\bullet 1} \leq T_1^{\bullet}$, is a simulation of $CA(Tr_2)$ by Prim-var($Tr_1, 1, T_{\bullet 1}, T_1^{\bullet}, 0$).*

Proof. Let Tr_1 be the commutative transition function given by Theorem 8.1, with the fields F, G. Let ξ_2 be an arbitrary configuration of $\mathbb{S}_2$ and let ξ_1 be a configuration of $\mathbb{S}_1$ defined in the statement of the same theorem. Let η_1 be a trajectory of $Prim\text{-}var(Tr_1, 1, T_{\bullet 1}, T_1^\bullet, 0)$, with the starting configuration ξ_1.

An update set $U(t, \eta_1)$ similar to (8.2) 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 (8.3):

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

With these, let us define

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

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

The simulation in this theorem is not a local one in the sense defined in Section 4.6 since it changes the time scale. For an analysis of such simulations, see [5].

9 Some simulations

In section, we build up the technique we will use in defining cellular automata and simulations.

9.1 Functions defined by programs

Recall the definition of a standard computing transition function and medium as introduced in Section 6.2.

Definition 9.1 (Computation result) For a standard computing medium with 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 the interval $[1, s-1]$ (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 the interval
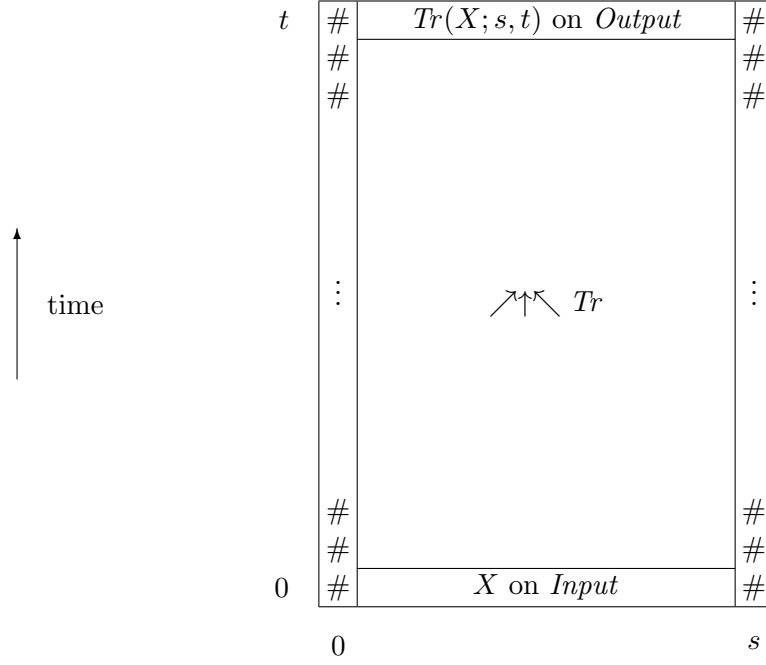
Figure 7: Definition of $trans(X; s, t)$

$[1, s - 1]$. The string w on the **Output** track on $[1, s - 1]$ will be called the *result* of the computation, denoted by

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

⌟

Since the output is monotonic in standard computing media, the result will not change at later times.

Universal cellular automata were introduced in Definition 4.22. We will need, however, also some bounds on the time and space used by the universal simulation.

Definition 9.2 (Efficient universality) The standard computing transition function Tr_0 is *efficiently universal* if for every standard computing transition

function Tr, there is a string $\mathbf{prog}$ and constants c, d such that for all strings X and positive integers s, t we have

$$Tr(X; s, t) = Tr_0(\mathbf{prog} \sqcup X; c \cdot s, d \cdot t)$$

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 9.1 *There are efficiently universal standard computing transition functions.*

Sketch of the proof. This theorem can be proven similarly to Theorem 4.1. 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. □

This theorem allows some standardization.

Definition 9.3 We fix an efficiently universal standard computing transition function

$$Univ \tag{9.1}$$

for the definition of various string functions. In view of Theorem 8.1, we can (and will) also assume that $Univ$ is commutative as in Definition 8.8. For a string function $f(X)$ defined over some domain E we will say that $\mathbf{prog}$ is a program for f with *time complexity bound t* and *space complexity bound s over E* if the relation

$$Univ(\mathbf{prog} \sqcup X; s, t) = f(X)$$

holds. ⌟

9.2 The rule language

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. This subsection defines a language for describing a transition function. Our purpose is twofold.

Convenience We want a method that is more convenient than giving a complete transition table.

Self-simulation Our self-correcting cellular automaton will simulate another self-correcting cellular automaton, identical to itself, or just similar to it: for simplicity, we refer to this now as "self-simulation". Self-simulation in our case has some peculiar requirements.

Let us note that self-simulation by itself is not mysterious. A universal cellular automaton U would use the program p of any cellular automaton with transition function Tr_p to simulate it. Of course, U itself has a program, q, so it could simulate itself as well, if we give it the program q. But it seems redundant to give U its own program, and actually it is better not to, since the written program is exposed to errors. How to define conveniently a machine that simulates (a modified version of) itself without needing any program? There are various ways of doing it, but each involves a notion of a program that is not a simple transition table, but rather has some ability to refer to itself: see the remark 9.4 below.

Remark 9.4 (Self-simulation) *Let us denote the output of a universal machine (it does not matter, whether it is a Turing machine or cellular automaton) computing the universal partial recursive function $u(p, x)$, where p is the program, and x is the input. Let τ be an arbitrary computable transformation of strings into strings. Kleene's Fixpoint Theorem (Recursion Theorem) says that there is a program q with the property that for all x we have $u(q, x) = u(\tau(p), x)$. What does this theorem have to do with our problem of self-simulation? Suppose that we have some program p, and a transformation $\tau(p)$ of p that carries out a simulation of the action of p, with possibly some modifications and extra properties (say, error-corrections, and the change of some parameters). The theorem says that there is a program q which already acts as $\tau(q)$.*

One intuitive proof of the fixpoint theorem goes as follows. Assume that we have a programming language that has at least the following ingredients: 1) We can define functions (procedures) in it. 2) It contains a function $\mathtt{U(p, x)}$ that computes $u(p, x)$, that is it interprets the string p as a program on input x. 3) It contains a function $\mathtt{MyText()}$,[4] which returns the string that is the text of the program. One implementation of such a function in a modern computer uses the fact that the program is stored at some standard place in memory: it is enough to go there and read it out. Let $\mathtt{T(p)}$ be the expression of the transformation $\tau(p)$ in our programming language: then the fixedpoint

---

[4]The parentheses (), used for example in the programming language C, are a useful reminder that even when it has no explicit arguments, the function depends on the internal state of the machine at the time of execution.

program is

$$q = \texttt{U}(\texttt{T}(\texttt{MyText}()), \mathbf{x})$$

(the $\mathbf{x}$ *in the program is just a symbol in the program, not something the program* q *depends on).* ⌋

A transition table can be seen as a sequence of *rules*: each rule could have the following form, for all possible triples $(a, b, c) \in \mathbb{S}^3$:

> **if** $r_{-1} = a$ **and** $r_0 = b$ **and** $r_1 = c$ **then**
> $\quad s \leftarrow d$

where $d = Tr(a, b, c)$. The most important generalization of this sort of rule involves fields. All the fields of our state will be defined in advance— as subintervals of the interval $[0, \|\mathbb{S}\|)$. Also, in our rule description, it is convenient to refer to the current cell and its neighor in a standardized way. Our rules, being just a description of the transition function, will always apply to a particular cell.

Notation 9.5 For a field F, if η is the current configuration and x is a site then we will sometimes write

$$F(x) \text{ for } \eta.F(x).$$

We will write

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

for the site j steps from site x. In a semi-formal description of a rule, $\mathbf{x}$ refers to the cell to which it is applied (the "current" cell). In the condition as well as the action of the rule, a field $F(\mathbf{x}) = \eta.F(\mathbf{x})$ will simply be written as F. Denote

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

⌋

The notation $\mathbf{x}$ can always be replaced with references to fields of the current cell and its neighbors. For example, for field *Addr*, instead of writing "**if** $\eta.Addr(\mathbf{x}) = 0$ **and** $\eta.Addr(\vartheta_j(\mathbf{x})) = 1$" we will write "**if** $Addr = 0$ **and** $Addr^j = 1$". Here is an example rule of this simplest form:

> **if** $Age > 0$ **and** $Addr < 100$ **then**
> $\quad Mail \leftarrow Mail^{-1}$

This means that if (at the site $\mathbf{x}$ under consideration) the *Age* field's value is 0 and the *Addr* field's value is less than 100 then the *Mail* field of the left

neighbor $\mathbf{x} - B$ should be copied into the *Mail* field of cell $\mathbf{x}$. We allowed in (2.7) the possibility of referring to a field by simply a pair of numbers i, j. In the rule language, i, j could themselves be specified by some (fixed) fields. Here is a more general definition of rules:

Definition 9.6 (Rules) An *elementary rule* has the following form:

> **if** C_1 **and** $\cdots$ **and** C_k **then**
> $\quad F \leftarrow f(F_1^{j_1}, \ldots, F_n^{j_n})$

Here F, F_i are fields (possibly specified as bit segments like in (2.7)), $j_i \in \{-1, 0, 1\}$. Each of the conditions C_i has the form $a_1 F_1^{j_1} + \cdots + a_n F_n^{j_n} \leq b$ with integer constants a_i, b. (When the condition is not applicable, the rule does not change the state.) The function $f(x_1, \ldots, x_n)$ must be computable on *Univ* in linear time. This allows for many functions: for example, if x, y are integers in binary notation then $x \cdot y$, $\lfloor x/y \rfloor$, $x \bmod y$ are all computable in linear time by known cellular automata algorithms, see for example [25].

A *rule* is a sequence of elementary rules. The transition function will be defined by a set of rules. In case some rules contradict each other, they will be ordered by explicit *priority*. Rules will sometimes be associated with certain fields. The lowest-priority rule referring to a field is called the *default*. ⌟

We will introduce several shorthand notations for expressing rules: these can always be translated into a rule according to Definition 9.6.

The most important shorthand is a branching conditional:

> **if** C_1 **then**
> $\quad A_1$
> **else if** C_2 **then**
> $\quad A_2$
> **else if** C_3 **then**
> $\quad A_3$
> **else**
> $\quad A_4$

We can combine several rules.

Definition 9.7 (Combination of rules) When a sequence of rules is combined, this is a *parallel* combination: all the rules apply to the same transition, and do not refer to consecutive times. Sometimes we may use the notation

$$R_1 \parallel R_2 \parallel \ldots \parallel R_n$$

for a combination of n rules. As a shorthand, we may also combine rules in series: the rule

$$R_1; R_2$$

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

In both parallel and serial combination, we will refer to the rules R_i as *sub-rules*, when they will always be invoked ("called") by other rules. Still, for a better flow of discussion we will frequently refer to sub-rules as just "rules". Sub-rules may have *parameters* that are specified at the time of the call. Rule $P(i)$ with parameter i can be viewed as a different rule for each possible value of i. ⌟

For example, whenever we write

```
Retrieve;
Eval
```

then this can be translated, using appropriate constants t_i, into the following, where `Retrieve` and `Eval` are sub-rules:

> **if** $t_1 \leq \text{\textit{Age}} < t_2$ **then**
>   `Retrieve`
> **else if** $t_2 \leq \text{\textit{Age}} < t_3$ **then**
>   `Eval`   // where `Eval` was defined in (6)

This example also shows a formal way to add comments to a rule definition, on any line after a #. An example for a simple parameterized sub-rule would be `Read-Mail`(j) for $j \in \{-1, 1\}$:

$$\text{\textit{Mail}} \leftarrow \text{\textit{Mail}}^j$$

for the field *Mail*.

Definition 9.8 (Conditionals and various kinds of "for") Sometimes the conditions of a conditional statement are indexed in a regular way, then we may use the shorthand **cfor** as below:

> **cfor** $j \in \{-1, 1\}$ **do**
>   **if** $\text{\textit{Age}}^j = 0$ **then**
> $\text{\textit{Age}} \leftarrow 0$

So this rule tests the conditions in order, carries out the command of the first satisfied condition and skips the rest. Several conditions that are to be tested one after the other will be combined using the construct **cond**. This can be seen as an alternative to the "**if**...**then**...**elsif**..." construct.

> **cond**
>> **cfor** $j \in \{-1, 1\}$ **do**
>>> **if** $Age^j = 0$ **then**
>>>> $Age \leftarrow 0$
>>
>> **cfor** $j \in \{-1, 1\}$ **do**
>>> **if** $Addr^j \not\equiv Addr + j \pmod{U}$ **then**
>>>> $Kind \leftarrow Latent$

Both the parallel and the consecutive combination of rules can be over an indexed set, and then we get the constructs **pfor** and **for**:

> **pfor** $(k, d) \in Mail\text{-}ind$ **do**
>> $Mail_{k,d} \leftarrow Mail\text{-}to\text{-}receive(k, d)$

Here, the rule is carried out simultaneously for all fields indexed by the indicated values k. On the other hand in the **for** example the application is consecutive:

> **for** $i = a$ **to** b **do**
>> $\langle$some rule referring to $i\rangle$

Another use of **for** can be this:

> **for** n steps of Age **do**
>> $\langle \cdots \rangle$

which, started at some $Age = t$ will mean that the commands $\langle \cdots \rangle$ will run under the condition $t_1 \leq Age < t + n$. The **repeat** construct could be translated into a **for**:

> **repeat** k **times**
>> $\langle \cdots \rangle$

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.

Definition 9.9 (The "let" construct) We may use temporary constants or functions in the course of defining a rule, as in this example:

$$\textbf{let } f(i) \leftarrow \textsf{Addr} + \textsf{Age}^i$$

⌟

Example 9.10 Here is a description of the transition function given in the proof of Theorem 8.1 (Asynchronous Simulation).

Algorithm 9.1: rule $\textsf{March}_0$

> **pfor** $j \in \{-1, 0, 1\}$ **do**
>> **let** $r(j) \leftarrow \textsf{Cur}^j$ if $\textsf{Age}^j = \textsf{Age}$, and $\textsf{Prev}^j$ otherwise
>
> **if** $\forall\, j \in \{-1, 1\}\ (\textsf{Age}^j - \textsf{Age}) \bmod U \geq 0$ **then**
>> $\textsf{Prev} \leftarrow \textsf{Cur}$
>>
>> $\textsf{Cur} \leftarrow Tr_2(r(-1), r(0), r(1))$
>>
>> $\textsf{Age} \leftarrow \textsf{Age} + 1 \bmod U$

⌟

Some primitives will always be available in the rule language. For example, for a field $\textsf{F}$, its width $|\textsf{F}|$ can be used. Moreover, a *slice* of the field as a string of bits $\textsf{F} = (f_0, \dots, f_{n-1})$ is available as well as $\textsf{F}[i{:}j]$ as indicated in (2.7). We will always have a field $\textsf{Addr}$. As before, a *colony* is a segment of cells with addresses growing from 0 to $Q - 1$ modulo some constant Q. Continuing to develop shorthands, it is useful to have notation for talking about parts of a colony:

Definition 9.11 (Location) A *location* is defined by a pair $(\textsf{F}, I)$ where $\textsf{F}$ is a field, and I is an interval of addresses in the colony, or by a constant-size union of such pairs $(\textsf{F}_j, I_j)$. We will denote the track $\textsf{F}$ over interval I as

$$\textsf{F}(I).$$

The name of a location will generally begin with an underscore, like

$$\textsf{\_Info}.$$

The set of addresses (typically an interval) of a location L will be denoted by $space(L)$. ⌟

The example in the above definition is important enough to be fixed:

Definition 9.12 (Info location) The location on the *Info* track containing the represented string is denoted by *␣Info*. ⌟

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*.

Remark 9.13 *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, $0^m 1^n$ has one run of 0's and a run of 1's. Let us also be given a location *loc*. Then $\texttt{Write}(S, loc)$, writing the string S to the location *loc*, can be written as a conditional sub-rule:

Algorithm 9.2: sub-rule $\texttt{Write}(0^m 1^n, \mathsf{F}([a, a + m + n)))$

    if $a \leq \mathsf{Addr} < Q \wedge (a + m)$ **then**
        $\mathsf{F} \leftarrow 0$
    else if $a + m \leq \mathsf{Addr} < Q \wedge (a + m + n)$ **then**
        $\mathsf{F} \leftarrow 1$

Definition 9.14 (Name definitions) The rule language can contain some definitions of names for constant strings of symbols, of the form

$$Param_1 = s_1, \ \ Param_2 = s_2, \ldots \tag{9.2}$$

where s_i are some strings. The names $Param_i$ are allowed to be used in the rules. This has the advantage that even if we use, say, $Param_1$ twice in a rule, this does not make the rule much longer, even if the value of the parameter is a long string.

Parameters will have descriptive names: say, we might write *Height* for $Param_1$, when $Param_1$ indicates the level in a hierarchy of simulations. Parameters will always be used via the function

$$\texttt{Write-param}(Param, loc),$$

which writes the string value of parameter *Param* to location *loc*. This can be implemented as a subrule, via the primitive access function

$$Param_i(j),$$

referring to the jth symbol of parameter i. There will also be a special parameter:

$$My\text{-}rules,$$

whose value is the string that is the sequence of all the rules. This will implement self-reference in the style of Remark 9.4. ⌟

Theorem 9.2 (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* (9.1) *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 interpr-coe $(|P| + 1)^2 \|\mathbb{S}\|$ *and space* interpr-coe $(|P| + \|\mathbb{S}\| + 1)$.

A detailed proof would be tedious, but routine. Essentially, each line of a rule program is interpreted in linear time on inputs that are 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*.

Later, in Section 19.3, we will add some other, simple features to the *Interpr* string: interpreting certain special commands.

9.3   A basic block simulation

The simulation described just demonstrates the use of the notation and introduces 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 4.7. The construction will be summarized in Theorem 9.3 below.

Along the way, we introduce some more shorthand notation in writing the rules that can also be incorporated into the rule language—without invalidating Theorem 9.2 (Rule Language).

Overall structure   The transition function $Tr_2 : \mathbb{S}_2^3 \to \mathbb{S}_2$ to be simulated is given by a rule program $\textit{Trans-prog}_2$. 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 Section 4.1. 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 $|\textit{Info}| = 2$, that is the *Info* track contains only symbols from the standard alphabet, introduced in Definition 3.3. The field *Cpt* will be used much of the time like the cells of a standard computing medium, so its size is the capacity $\|\mathbb{S}_{Univ}\|$ 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 the rest of *Cpt* is under the control of the rule `Eval`. The whole program can be summarized as follows:

Algorithm 9.3: Basic simulation program

```
Retrieve;
Eval;
Update
```

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

$$\textit{Copy}(m, loc_1, loc_2)$$

that copies, from the colony with base $x - mQ$, the location loc_1 to location loc_2 of the current colony. To describe the rule `Copy` we use a framework a little more general than what would be needed here, with a variable-time version in mind.

Definition 9.15 (Locations for the retrieval rule) The rule `Retrieve` uses the locations $\_Retrieved_m$ to deposit the string it retrieves from the neighbor colony number $m \in \{-1, 0, 1\}$. ⌟

With the help of the copy rule, we can formulate retrieval now as follows:

Algorithm 9.4: rule `Retrieve`

> **for** $m \in \{-1, 0, 1\}$ **do**
> $\quad$ $Copy(m, \_Info, \_Retrieved_m)$

There will be several mail tracks, indexed by the following values:

$$Mail\text{-}ind = \{(k, d) : k \in \{-1, 0, 1\}, d \in \{-1, 1\}\}. \tag{9.3}$$

The meaning of the two values $-1, 1$ of d is different for $k \in \{-1, 1\}$ and $k = 0$. If $k \in \{-1, 1\}$ then $d = 1$ is for a track to *send to* the destination colony in direction k, and $d = -1$ is for a track to *receive from* that colony. If $k = 0$ then the track with $d \in \{-1, 1\}$ is for sending *left* or *right* within the present colony.

In an ordinary cellular automaton, we could move a string on the mail track for a certain number of steps and then copy it to another track in a single step. But in a variable-time medium, we cannot rely on such timing. Therefore with each piece of information, the mail track will also carry the address of the place it is coming from, and this will allow to know when and where to deposit it without relying on timing. Field $Mail_{k,d}$ has subfields

$$Toaddr, Info.$$

For simplicity, let $|Mail_{k,d}.Info| = |Info|$. For adjacent cells x, y with $j = y - x$, and their colonies x^* and y^* (defined from their $Addr$ field) we define the predicate

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

Thus, the function $Edge_{-1}(x)$ depends implicitly on the current values of the address fields in the configuration η. We have, $Edge_{-1}(x) = 0$ if $x - 1$ is in

the same colony as x, it is 1 if $x - 1$ and x are right and left endcells of their respective colonies, and it is ∞ otherwise (abnormal case, with "inconsistent" address field values). The mail track $Mail_{k,d}$ of cell x will read, from the neighbor in direction $j = j(k, d)$, from mail track

$$peer(k, d) = (k, d')$$ (9.4)

defined as follows.

- If $|k| = 0$, $Edge_{-d}(x) = 0$ then $j = -d$, $d' = d$.
- If $|k| = 1$, $Edge_{-kd} = 0$ then $j = -kd$, $d' = d$.
- If $|k| = 1$, $Edge_{kd} > 0$ then $j = kd$, $d' = -d$.

In all other cases, the values $j(k, d)$, $peer(k, d)$ are not defined. In words: the mail is passed along the same mail track except that when it crosses an edge then it goes from the sending track of the sending colony to the receiving track of the receiving colony. Formally we define

$$Mail\text{-}to\text{-}receive(k, d) = Mail^{j(k,d)}_{peer(k,d)}.$$ (9.5)

as the mail to be received into $Mail_{k,d}$ provided $peer(k, d)$ is defined, and $Undef$ otherwise. The one-step sub-rule $\texttt{Move-mail}$ gets mail from the neighbor cell:

Algorithm 9.5: sub-rule $\texttt{Move-mail}(k, d)$

$Mail_{k,d} \leftarrow Mail\text{-}to\text{-}receive(k, d)$ if the latter is defined.

A cell will typically copy the information to be sent into the subfield $Mail_{k,d}.Info$ and at the same time, the receiver cell's address into $Mail_{k,d}.Toaddr$. In the copy rule here, j 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.

Algorithm 9.6: sub-rule $\texttt{Copy}(m, F([a, a + n)), G([b, b + n)))$

    let $d \leftarrow 1, d' \leftarrow -1$ if $m \neq 0$ and $d' = d \leftarrow \text{sign}(b - a)$ otherwise
(1) **if** $Addr \in [a, a + n)$ **then** $Mail_{m,d}.(Toaddr, Info) \leftarrow (Addr + b - a, F)$
    repeat $2Q$ **times**
        $\texttt{Move-mail}(m, d)$
        if $Addr \in [b, b + n)$ **and** $Addr = Mail_{m,d'}.Toaddr$ **then**
            $G \leftarrow Mail_{m,d'}.Info$

Remark 9.16 *Here we assumed that the source and destination locations are on tracks of the same width, and have the same lengths. We can write a*
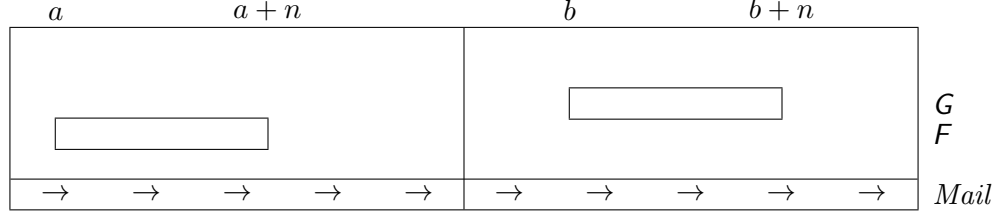
Figure 8: $Copy(-1, F([a, a+n)), G([b, b+n)))$

more general version of Copy *using a loop. For example assume that the source location, of length* n*, is on a track* F *that is 5 times wider than its destination location* G *of length* $5n$*. Then we can split the field* F *into 5 subfields* $F_1, \ldots, F_5$*. More generally, a track that has* m *times the width can be treated as into* m *tracks of the same width. Here* m *can even be a variable computed from the widths of the fields in question, using the primitive* $F[i{:}j]$ *as defined in* (2.7)*. We can do the copying in 5 iterations, copying each subtrack separately, or (this will be our actual choice later) we can post each cell onto the mail track in 5 steps. The rule* Move-mail *can proceed in parallel with all these iterations.*

Similarly, one can write a program for copying a longer and narrower location into a shorter and wider one. Also, instead of giving the locations as arguments, we may just give the name of a field where they are described. In applications we will assume this more general implementation. ⌟

Definition 9.17 (Indirectly given locations) The description $F_1([a_1, a_2))$ of a location can fit into a single field G_1 of some cell. If for example argument loc_1 of the rule $Copy(m, loc_1, loc_2)$ is given by a field G_1 this way, then we write $Copy(m, G_1*, loc_2)$. This will only happen if $m = 0$, that is the copying proceeds within one colony. It will be assumed that field G_1 of each cell of the colony contains the same information loc_1. Therefore the rule can be written just as above, except that some of its parameters are read now from the field G_1. ⌟

The evaluation rule The rule Eval controls the track Cpt. Before describing it we need to define some locations.

Definition 9.18 (Locations for evaluation) Here are a few locations used in the rule Eval:

- Track *Cpt* is used for universal computation, with sub-tracks *Cpt.Input* and *Cpt.Output*.

- The location for the interpreter on the *Cpt.Input* track is denoted by *_Interpr*.

- The location for the program to be executed by the universal medium *Univ* is denoted by *_Prog*.

- The locations for the parameters of the program are denoted by *_Param$_i$*, $i = 1, 2, \ldots$, as needed,

- The location of the arguments of the function to be executed on the *Cpt* track are *_Arg$_m$* for $m \in \{-1, 0, 1\}$.

- The location of the output of the computation on the *Cpt.Output* track is *_Sim-output*. We assume it to be the same interval as that of location *_Info*.

⌟

We will assume that the program to be simulated is written as a string value of parameter *Trans-prog$_2$* of our program. The first steps of `Eval` get the arguments, then write the interpreter (a constant string), followed by writing the program `Trans-prog`$_2$ of the transition function to be simulated:

$$\texttt{Write}(Interpr, \_Interpr);$$
$$\texttt{Write-param}(Trans\text{-}prog_2, \_Prog);$$

But if $Tr_1 = Tr_2$ (self-simulation) is desired, then write *My-rules* in place of *Trans-prog$_2$* above. In both cases, follow with $\texttt{Write-param}(Param_i, \_Param_i)$ and copying *_Retrieved$_m$* to *_Arg$_m$* for each m. Then the rule

$$\texttt{Initialize} \tag{9.6}$$

writes $* \cdots *$ to the *Output* track and the rest of the cells (including the end-cells) of the *Cpt.Input* track, and 0's to the track *Cpt*\(*Cpt.Input*∪*Cpt.Output*). Then the rule

$$\texttt{Interpret}$$

applies, for a sufficient number of steps, the transition function *Univ* to the *Cpt* track. According to Theorem 9.2 (Rule Language), the computation finishes in

$$\mathsf{interpr\text{-}coe}\,(|Trans\text{-}prog_2| + 1)^2 \|\mathbb{S}_2\|$$

Algorithm 9.7: sub-rule `Eval`

> `Write`$(Interpr, \_Interpr)$;
> `Write-param`$(My\text{-}rules, \_Prog)$;
> **for** $i = 1$ **to** N **do**
>   `Write-param`$(Param_i, \_Param_i)$
> **for** $m = -1, 0, 1$ **do** `Copy`$(0, \_Retrieved_m, \_Arg_m)$
> `Initialize`;
> `Interpret`

steps, so this number of iterations is sufficient. So the whole rule `Eval` is (for the case of self-simulation) in Algorithm 9.7, for N parameters:

The subrule

$$\texttt{Update}$$

in Algorithm 9.3 copies the track *Cpt.Output* into track *Info* for all addresses in $space(\_Info)$.

Summary in a theorem   Before formulating what will be accomplished by the above program, let us define some encodings.

Definition 9.19 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

$$\texttt{Trans-prog}_2, \|\mathbb{S}_1\|, \|\mathbb{S}_2\|, Q, U \tag{9.7}$$

will be restricted as follows.

Condition 9.20 The following inequalities must be obeyed.
Cell Capacity Lower Bound

$$\|\mathbb{S}_1\| \geq c_1 \lceil \log U \rceil + \|\mathbb{S}_{Univ}\| + c_2$$

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:

$$|\textit{Info}| = 2,$$
$$|\textit{Mail}_i| = |\textit{Mail}_i.\textit{Toaddr}| + |\textit{Mail}_i.\textit{Info}| = \lceil \log Q \rceil + 3,$$
$$|\textit{Cpt}| = \|\mathbb{S}_{Univ}\|,$$
$$|\textit{Addr}| = |\textit{Age}| = \lceil \log U \rceil.$$

Colony Size Lower Bound

$$Q \geq \mathsf{interpr\text{-}coe}\left(\|\mathbb{S}_2\| + |\texttt{Trans-prog}_2| + \log U + 1\right).$$

With the field sizes as agreed above, this provides sufficient space in the colony for information storage and computation.

Work Period Lower Bound

$$U \geq 3Q + \mathsf{interpr\text{-}coe}\left(|\texttt{Trans-prog}_2| + 1\right)^2 \|\mathbb{S}_2\|.$$

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 9.3 (Basic Block Simulation) *There are strings* $\texttt{Sim-prog}_1$, $\texttt{Sim-prog}_0$ *such that the following holds. If* $\texttt{Trans-prog}_2$ *with parameters* $\|\mathbb{S}_1\|$, $\|\mathbb{S}_2\|$, Q, U *satisfies the above inequalities, and* $\texttt{Trans-prog}_2$ *defines a transition function* Tr_2, *then* $\texttt{Sim-prog}_1$ *is a rule program with parameters*

$$\texttt{Trans-prog}_2, \|\mathbb{S}_1\|, \|\mathbb{S}_2\|, Q, U,$$

defining a transition function Tr_1 *such that* $CA(Tr_1)$ *has a block simulation of* $CA(Tr_2, Q, U)$.

If $\|\mathbb{S}_1\|, \|\mathbb{S}_2\|, Q, U$ *satisfy the above inequalities, then* $\texttt{Sim-prog}_0$ *is a rule program with parameters* $\|\mathbb{S}_1\|$, Q, U, *defining 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 $\texttt{Sim-prog}_1$ and $\texttt{Sim-prog}_0$ explicitly.

10 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. The definition can be compared to the primitive variable-period media in Section 7.4. The main distinguishing feature is the notion of damage.

Remark 10.1 *The paper [13] used communication only between adjacent neighbors. It used a special mechanism for preventing accidental intrusions into an intact colony: growth in a zigzag pattern. Here this is achieved by communication between non-adjacent neighbors, enabling a colony of "stronger" cells to defend itself from the accidental intrusion by "weaker" cells. However, to simulate such a property on higher levels, it will also be used to pass information other than strength to non-adjacent neighbors.* ⌋

10.1 Damage

In a robust medium, being in the space-time set *Damage* defined below excuses the site for "not following the rules".

Definition 10.2 (Damage) For the media to be defined, robust media, we introduce a special set of states *Bad* ⊆ *Vacant*. For a history η we define the *damage set*

$$Damage(\eta) = \{(x,t) : \eta(x,t) \in Bad\}.$$

For a space configuration, the damage is defined similarly. For a site x a time interval I is *damage-free* if $\eta(x, \cdot)$ is not in *Bad* during I. ⌋

Damage points can be viewed as holes in the fabric of the lawful parts of a trajectory. When $(x,t) \in Damage(\eta)$ then in the neighborhood of (x,t), we will not be able to make any predictions of η, that is in some sense, η behaves completely "lawlessly" there. 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.

Definition 10.3 (Damage map) We will use the following constants:

$$\gamma = 5, \quad \beta = 2\gamma + 5. \tag{10.1}$$

Let us define the rectangle

$$V = [-B/2, B/2) \times (-T^\bullet/2, T^\bullet/2], \tag{10.2}$$

then the corresponding rectangle V^* is defined using $B^*, T^{\bullet *}$. Two space-time points are said to be *too close* if they are contained in a single copy (space-time translation) of βV. Two sets A_1, A_2 are *too far* if no copy of γV^* intersects both. A subset A of the damage is called an *island* if it is covered by a translation of βV and is too far from the rest of damage. In a simulation $\eta^* = \Phi^*(\eta)$ between two robust media, the set $Damage(\eta^*)$ is obtained after we remove all islands of $Damage(\eta)$. We call this definition of $Damage(\eta^*)$ the *damage map* of simulation Φ^*. ⌟

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*. The *Computation Property* constrains the kinds of events that can occur under the condition that the damage does not intersect a certain rectangle. The *Restoration Property* bounds the probability of occurrence of damage in the middle of some window. It depends on a parameter ε, and 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 10.4 (Restoration Property) Let $b(\alpha(restor)) = \varepsilon$. Further, for any pair (x, t) let

$$g\big(\alpha(restor),\ (x, t) + (\gamma + 2)V,\ \eta\big)$$

be the event function for the event that there is damage in $\eta((x, t) + V)$. ⌟

The Restoration Property says that damage, that is the occasional obstacle to applying the Computation Property, has small conditional probability of occurrence in the middle part $(x, t) + V$ of any rectangle of the form $(x, t) + \gamma V$. The property will hold automatically on the lowest level by the property of the medium that the transition rule is only violated with small probability.[5]

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

---

[5]In the model of [14], the restoration property 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.

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.

Provided $Damage(\eta^*)$ is not nearby, $\eta^*(x, t)$, will be essentially defined by a block code φ as

$$\varphi^*(\eta(x + [0, QB - 1], \ t)).$$

However, in the interest of stabilization there will be some look-back in time: the simulation will not be memoryless.

Lemma 10.5 (Simulation Damage Probability Bound) *Let M, M^* be media with parameters ε, ε^*, with $B^* \geq B$, $T^{\bullet*} \geq T^\bullet$, whose local condition system includes the Restoration Property. Let a simulation Φ^* be defined between them which assigns damage in M^* according to the damage map of Definition 10.3. There is a constant c_{Dam} such that if*

$$\varepsilon^* \geq c_{Dam}((B^*/B)(T^{\bullet*}/T^\bullet)\varepsilon)^2 \tag{10.3}$$

then Φ^* is a deterministic canonical simulation map for damage, as defined in 7.3.

The proof observes 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 $O(\varepsilon^2)$ times the number of such pairs in a rectangle V^*.

Proof. It is enough to show an expression of the form (7.13) for local conditions of type *restor* in M^*. Let η be a trajectory of M. Consider the event that $Damage(\eta^*)$ intersects the middle part $W' = (x, t) + V^*$ of rectangle

$$W = (x, t) + (\gamma + 2)V^*,$$

with β, γ defined in (10.1), then of course $Damage(\eta)$ intersects it, too. In what follows, by "damage" we mean the set $Damage(\eta)$. We claim that the damage in W cannot be covered by a copy W'' of βV. Suppose namely that it can: then it can also be covered by a copy of $W' + \beta V$. However, the rectangle $W' + \beta V$ is too far from the complement of W. Then so is W'', so it is an island and as such it would have been deleted by the damage map, contradicting the assumption that its elements belong to $Damage^*$.

We found two damage points p_1, p_2 in W that are not too close. Let us fix some partition of the space into copies of V. Let $\mathcal{V}$ be the set of those elements of this partition intersecting W. For each point p in space-time, let $U(p)$ be the $U \in \mathcal{V}$ with $p \in U$. Let $U'(p) = U(p) + (\gamma + 1)V$. The fact that p_1, p_2 are not too close implies that the two copies $U'(p_1)$ and $U'(p_2)$ of

$(\gamma + 2)V$ are disjoint. Indeed, if they intersected then some copy of $2(\gamma + 2)V$ would contain both, but due to (10.1) then they would be too close.

Let E be the set of pairs U_1, U_2 in $\mathcal{V}$ with $U_1' \cap U_2' = \emptyset$. We found

$$g^*(restor, (\gamma + 2)V^*, \eta^*) \leq \sum_{(U_1, U_2) \in E} \prod_{j=1,2} g\big(restor, U_j', \eta\big),$$

therefore $\mathsf{E}\, g^*(restor, V^*, \eta^*) \leq |E|\varepsilon^2$. Since counting bounds $|E|$ by $c_{Dam}((B^*/B)(T^{\bullet*}/T^\bullet))^2$ for an appropriate constant c_{Dam}, by the assumption (10.3) both conditions of a weak canonical simulation in (7.13) are satisfied.

$\square$

10.2   Computation

Before giving the Computation Property, let us introduce some details of the structure of a robust medium. Cells sometimes have to be erased 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.

Neighborhood structure   We extend the meaning of the neighbor function $\vartheta_j(x)$ introduced in Notation 9.5.

Notation 10.6 (Non-adjacent neighbors) Let us fix a history η. For a (non-vacant) cell x, in direction $j = \pm 1$ there is at most one cell at distance smaller than $2B$: if it exists and there is no damage between it and x then we denote it also by $\vartheta_j(x, t, \eta)$. 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 $\eta(\vartheta_j(x, t, \eta), t) = Vac$.   ⌟

Having an extended notion of neighbors, we can define the notion of transitions for robust media.

Definition 10.7 (Transition of a robust medium) The transition function in a robust medium has the form

$$Tr(\mathbf{s}, \mathbf{a}). \tag{10.4}$$

Here $\mathbf{s} = (s_{-1}, s_0, s_1)$, $\mathbf{a} = (a_{-1}, a_1)$ have the following meaning. s_0 is the state of the cell whose transition will happen, and for $j = \pm 1$ s_j is the state

of its (not necessarily adjacent) neighbors in direction j. Further $a_j = 1$ if the neighbor in direction j is adjacent and 0 otherwise. The transition does not depend on a_j if either s_j or s_0 is vacant. Note that a_j is in fact $a_j(x, t, \eta)$, but we may omit the arguments that are obvious from the context. In analogy with (2.6), let

$$\overline{Tr}(\eta, x, t) = Tr((\vartheta_{-1}(x), x, \vartheta_1(x)), (a_{-1}(x), a_1(x))).$$

The symbol η will be omitted from $Tr(\eta, x, t)$ when it is obvious from the context. ⌟

Properties of the transition function   The state space of a robust medium will be required to have some minimal structure, and every transition function will be required to have certain properties. In Definition 10.2 it has been mentioned already that the set of states has a subset Bad. The value of the transition function Tr will not be defined if any of its argument is Bad and we could even say that this defines the set Bad.

Definition 10.8 A robust medium will be denoted by

$$Rob(Tr, B, T_\bullet, T^\bullet, \varepsilon, \varepsilon').$$

The first bit of the state of a cell will be the field *Rand*; we will write

$$Det = All \setminus Rand.$$

⌟

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 impose a number of conditions on the transition functions in robust media.

Condition 10.9 (Time Marking) If a cell is not vacant, then the transition function always changes it. ⌟

This condition is similar to (7.24), and is helpful for nontrivial media with a strict dwell period upper bound.

Condition 10.10 Non-vacant cells will have a property (typically represented by a field) of being *latent* or not. Newborn cells will be latent. The transition function will not make a difference between a vacant and latent neighbor. ⌟

This condition will help attributing all information obtained from the neighbors to the same moment of time even when a vacant neighbor turns to a latent one.

Transitions between vacant and non-vacant state are handled in a special way.

Condition 10.11 (Cling to Life) A cell will only be erased if it may disturb a close non-aligned neighbor: namely, suppose that $u_0 \neq Vac$ and $Tr(\mathbf{u}, \mathbf{a}) = Vac$. Then there is a $j = \pm 1$ with $u_j \neq Vac$ and $a_j = 0$. ⌟

Definition 10.12 (Creators, emergence) Recall the form of the transition function in (10.4). We will call a pair $(j, v) \in \{-1, 1\} \times \mathbb{S}$, a *potential creator* of a non-vacant state s if $s = Tr(\mathbf{u}, \mathbf{a})$ where $u_j = v$, $u_0 = u_{-j} = Vac$. The expectation is that an adjacent cell with state s will be created by the cell with state v determined by the state u_j of the creator *independently of* the state u_{-j}.

Consider a history η, with a switching time σ of cell x when $\eta(x, t)$ turns from vacant to non-vacant. We call $y = x + jB$ a *creator* of x for time σ if $(j, \eta(y, \sigma-))$ is a potential creator of $\eta(x, \sigma)$, further $\eta(y, t)$ is non-vacant for $t \in (\sigma - T_\bullet/2, \sigma + T_\bullet)$. (So a creator is supposed to survive the creation.) On the other hand, if there is no t in $(\sigma - T_\bullet/2, \sigma]$ when the interval $[x - 2B, x + 3B)$ intersects the body of any cell, we will say that the cell x *emerged* at time σ. ⌟

The computation property The condition called the Computation Property has a form similar to the definition of primitive variable-period media in Section 7.4.

Definition 10.13 (Special switching times) For a history η, cell x and number $a \geq 0$ let

$$\sigma_1, \sigma_2, \sigma_0$$

be defined as follows. The values σ_1, σ_2 are the first two switching times of x in $(a, a + 2T^\bullet]$, but σ_2 is defined only if $\eta(x, \sigma_1)$ is non-vacant. 0 is considered a switching time if $\eta(x, 0)$ is not vacant. On the other hand, σ_0 is the first switching time of x after $a + T^\bullet$ but defined only if $\eta(x, a + T^\bullet)$ is vacant. 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). ⌟

Referring in an event function to, say, σ_2 directly is a shorthand. We will consider the times σ_i only in places where no damage occurs, which allows us to lower-bound all dwelling times by $T_\bullet$. Then all requirements referring to σ_i can be expressed instead in terms of some arbitrary sequence of rational numbers $t_i = t_0 + iT_\bullet$. For example the requirement $\sigma_2 - \sigma_1 < T^\bullet$ can be expressed by prohibiting for each rational t, and for n with $T^\bullet/n < T_\bullet$, that all $\eta(x, t + id) = \eta(x, t)$ hold for $i = i, \ldots, n$. The requirement $\mathsf{P}\{\eta(x, \sigma_2).Rand =$
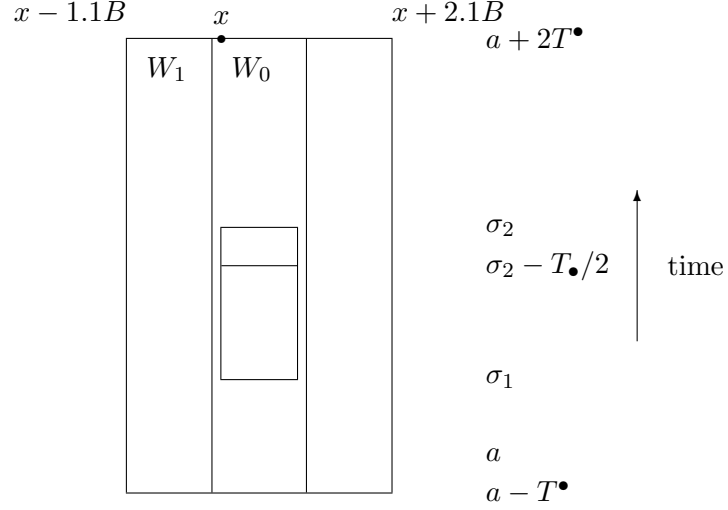
Figure 9: To the computation property. The large rectangle is $W_1(x, a)$. The rectangle of the same height but width $1.2B$ around the cell body at point x is $W_0(x, a)$. The small rectangle between times σ_1 and σ_2 is the cell work period under consideration. Its lower part is the observation interval.

$j\} \le 1 + \varepsilon'$ can be expressed by

$$\mathsf{P}\{\eta(x, t_1) \ne \eta(x, t_0) \wedge \eta(x, t_1).\mathsf{Rand} = j\} \le 1 + \varepsilon'.$$

for $t_0 < t_1 < t_0 + T_\bullet$.

For a transition to work satisfyingly, the cell as well as its neighbors must have been damage-free for some time before the transition. But it is convenient to require that a transition not fail completely even if only the cell itself is damage-free, not its neighbors. The following space-time rectangles will play a special role.

Definition 10.14 [Rectangles used in the computation property] Let

$$I_0(x) = [x - 0.1B, x + 1.1B), \quad I_1(x) = [x - 2.1B, x + 3.1B),$$
$$W_j(x, a) = I_j(x) \times (a - 2T^\bullet, a + 3T^\bullet] \quad (j = 0, 1),$$
$$f_j(x, a, \eta) = \text{the event function for } \{Damage(\eta) \cap W_j(x, a) = \emptyset\} \quad (j = 0, 1).$$

⌟

Let us write

$$Damage = Damage(\eta), \quad Damage^* = Damage(\eta^*).$$

Definition 10.15 (Affecting) We will say that damage *affects* cell (x, t) *directly*, if it intersects with $I_0(x)$ at time t, with $I_j(x)$ as in Definition 10.14. It affects a cell *via neighbors*, if it intersects $I_1(x)$. ⌐

Below we formulate Condition 10.16 (Computation Property), which beside the Restoration Property is the other crucial requirement on the trajectory. It implies that if a cell is not affected directly during a certain time interval, then it makes a legal transition. It may still not make the transition required by the transition function: for that, it must not be affected even via neighbors. With these notions, we are ready to spell out the computation property. Condition 10.16e) refers to a notion called "atomicity" in the field of distributed computing: the fact that despite the complexity and prolonged nature of the interaction between a colony and its neighbors, its simulation result can be viewed as the result of an observation of the states of the three represented big cells in a single moment.

Condition 10.16 (Computation Property) This property consists of several condition types. For each type α used here except $\alpha(rand, j)$, we have $b(\alpha) = 0$, that is the corresponding events $g(\alpha(\cdot), W, \eta)$ are simply prohibited in the trajectory. On the other hand,

$$b(\alpha(rand, j)) = 0.5 + \varepsilon' \quad (j = 0, 1).$$

a) Coin-tossing, for $j = 0, 1$:

$$g\big(\alpha(rand, j), \ W_0(x, a), \ \eta\big) = f_0(x, a, \eta)\{\exists \sigma_2 \wedge \eta(x, \sigma_2).\mathsf{Rand} = j\}.$$

b) The length of dwell periods must be between $T_\bullet$ and $T^\bullet$:

$$g\big(dw\text{-}p\text{-}bd, \ W_0(x, a), \ \eta\big) = f_0(x, a, \eta)h(dw\text{-}p\text{-}bd, x, a, \eta)$$

where $h(dw\text{-}p\text{-}bd, x, a) = 1$ 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) Whenever $W_0(x, a)$ is damage-free, the transition at σ_2 is a legal one:

$$g\big(legal\text{-}comp, \ W_0(x, a), \ \eta\big) = f_0(x, a, \eta)\{\exists \sigma_2 \wedge \neg legal(\eta(x, \sigma_2-), \eta(x, \sigma_2))\}.$$

d) Whenever $W_1(x, a)$ is damage-free and σ_0 exists, x either has a creator or is emerging (see Definition 10.12).

$$g\big(newborn, \ W_1(x, a), \ \eta\big) = f_1(x, a, \eta)\{\exists \sigma_0 \wedge \neg h(newborn, x, a, \eta)\}$$

where $h(newborn, x, a, \eta) = 1$ if x has a creator for time σ_0, or $\eta(x, \sigma_0)$ is emerging. Also, the transition in σ_0 is a legal one:

$$g\big(legal\text{-}birth,\ W_0(x, a),\ \eta\big) = f_0(x, a, \eta)\{\exists \sigma_0 \wedge \neg legal(\eta(x, \sigma_0-), \eta(x, \sigma_0))\}.$$

e) Whenever $W_1(x, a)$ is damage-free, the transition function applies, based on observation at some (unspecified) time point during the *observation interval* ("atomicity"):

$$g\big(trans,\ W_1(x, a),\ \eta\big) = f_1(x, a, \eta)\{\exists \sigma_2 \wedge \neg h(trans, x, a, \eta)\}$$

where $h(trans, x, a, \eta) = 1$ if there is a $t' \in (\sigma_1, \sigma_2 - T_\bullet/4]$ with

$$\eta(x, \sigma_2).\mathsf{Det} = Tr(\eta, x, t').\mathsf{Det}.$$

f) A cell cannot stay vacant if it has would-be creators for a long time and has no neighbor potentially blocking the creation:

$$g\big(no\text{-}birth,\ W_1(x, a),\ \eta\big) = f_1(x, a, \eta)h(no\text{-}birth, x, a, \eta)$$

where $h(no\text{-}birth, x, a, \eta) = 1$ if the following conditions hold:

 i) $\eta(x, t)$ is vacant for all $t \in [a, a + 3T^\bullet]$;
 ii) 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;
 iii) there is no non-vacant $\eta(y, t)$ with $0 < |y - x| < B$, $t \in [a, a + 3T^\bullet]$.

⌟

Our conditions do not require the state of an emerging cell to be completely determined by the transition function (other than that it is latent, as said in Condition 10.10). In fact as we will see, during self-organization, the state of an emerging big cell may depend on the information in the germ cells creating the colony representing it.

Let us take a peek ahead to self-simulation, since at this point, condition 10.16f) seems too strong. Indeed, suppose that a colony of Q cells of size B simulates a higher-order cell of size QB in a medium η^* in such a way that these higher-order cells must obey the conditions of a robust medium again. Now assume again that $\eta^*(x, t)$ is vacant for a long time, and for all this time, there is a potential creator in $x - QB$, while there is no (y, t) with $0 < |y - x| < QB$ with non-vacant $\eta^*(y, t)$. Can then indeed a new colony be created in x? There could in principle be another obstacle, namely that there is a y with $0 < |y - x| < QB$ such that there is a colony during all this time

starting in $y + QB$ (that is $\eta^*(y + QB, t)$ is nonvacant) and is trying to create a colony with starting point y. The attempts of colony creation from left and right interfere. Our solution is simple: growth from the left will get priority over growth from the right. In this way, one of the conflicting attempts will succeed.

Though it is desirable to illustrate the computation property on a number of special cases, currently, we will give only the barest minimum of these.

Example 10.17 (Special cases) To obtain a deterministic cellular automaton as a special case of robust media, set $T_\bullet = T^\bullet$, $r = 1$, $\varepsilon = 0$, require that the histories have empty damage and that the transition function not give a vacant value.

The connection between primitive variable-period media and robust media will be set up via a trivial simulation. Let $M_1 = Prim\text{-}var(\mathbb{S}_1, \Lambda, Tr_1, B, T_\bullet, T^\bullet, \varepsilon)$. We make the additional assumptions that the transition function Tr_1 satisfies the time marking Condition 10.9. We define a simple simulation Φ^* by this medium of the robust medium

$$M_2 = Rob(\mathbb{S}_2, \Lambda, Tr_2, B, T_\bullet, T^\bullet, \varepsilon, \varepsilon, 1).$$

Set $\mathbb{S}_2 = \mathbb{S}_1 \times \{0, 1\}$, identify $\mathbb{S}_1$ with $\mathbb{S}_1 \times \{0\}$, and denoting $\mathbb{S}_1' = \mathbb{S}_1 \times \{1\}$, let $Vacant_2 = Vacant_1 \cup \mathbb{S}_1'$, $Bad_2 = Bad_1 \cup \mathbb{S}_1'$. Thus, some new bad states are introduced by turning on a new bit which we can call the "bad bit". The transition function Tr_2 is essentially the same as Tr_1: it ignores the bad bit. It also satisfies Condition 10.11 trivially: it never destroys or creates cells.

Let g_1 be the event function used in the definition of medium M_1. 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 $s = Vac$. If $g_1(\alpha, \{x\} \times (t', t], \eta) = 1$ for some $t' < t$ and some $\alpha \neq coin$ then $s = (\eta(x, t), 1)$ (as some condition is violated, the bad bit is turned on), else $s = (\eta(x, t), 0)$.

It can be verified that this is indeed a simulation. ⌋

Peeking ahead to simulations, Lemma 10.5 proves the Restoration Property (Condition 10.4) for $\eta^* = \varphi^*(\eta)$ whenever η is a trajectory of M. When the computation property above is applied to a cell x of η^*, we are given a rational number a satisfying $f_1(x, a, \eta) = 1$. Since we have to prove this property for η^*, it will be assumed throughout the rest of the construction that $Damage^*$ does not intersect the area we are reasoning about. Hence all damage belongs to islands as in Definition 10.3, and for simplicity, we will identify the damage with the islands (that is assuming it fills them out).

11 Amplifiers

In the present section, when we talk about media without qualification we understand robust media. Recall amplifiers from Definition 5.10. We will also need a trickle-down property, as in Definition 5.11. Then a lemma analogous to Lemma 5.12 can be formulated, and a proof for Theorem 7.4 can be provided analogously to the proof of Theorem 3.1.

Our eventual goal is to find an amplifier (M_k, Φ_k), where each medium M_k will also simulate a some computation with a given transition function, and damage probability bound ε_k, for a fast decreasing sequence ε_k. We introduce some parameters of the sequence (M_k) of media that will satisfy conditions sufficient for the existence of an amplifier. These conditions are similar to the inequalities in the conditions of Theorem 9.3.

Recall from Definition 10.8 that all robust media M_k have the fields *Det* and *Rand* whose behavior is governed by the computation and restoration properties.

Notation 11.1 Only field *Det* will be subdivided into subfields: therefore subfields $\eta.\textsf{Det}.\textsf{F}$ will be denoted simply as $\eta.\textsf{F}$, without danger of confusion.
⌋

Definition 11.2 (Amplifier complex) Consider an amplifier (M_k, Φ_k) where $\Phi_k = (\varphi_{k*}, \Phi_k^*)$. For a sequence of fields $(\textsf{Payload}_k)$ and parameters D_k, ε_k'', our amplifiers will use a code complex

$$\mathbf{H} = (\varphi_k, \textsf{Payload}_k, \gamma_k)_{k \geq 1}$$

as in (5.1), and they will have in addition the trickle-down property as in Definition 5.11 for the fields $\textsf{F}_k = \textsf{Payload}_k$. We will call such an amplifier an *amplifier complex.*
⌋

In any trajectory η of M_1 at space-time point (x, t) the hierarchy might only be built up to a certain highest level k which may increase as time proceeds. The cells on this level will have the kind *Germ* (on the field *Kind* see later in Section 12), and the useful computation of M_k will be carried out on their $\textsf{Payload}_k$ track. This computation will be an aggregated version $Pl\text{-}trans_k$ of a certain fixed cellular automaton with transition function $Pl\text{-}trans_1$. It will have monotonic output in the sub-track $\textsf{Output}_1$, and correspondingly $Pl\text{-}trans_k$ will have monotonic output on $\textsf{Output}_k$.

Condition 11.3 We require transition function $Pl\text{-}trans_1$ to be commutative in the sense of Definition 8.8.
⌋

As shown in Theorem 8.1, this requirement is not strong: the transition can still be, for example, universal; but it makes its simulation simpler as we don't have to worry about the order of updating (for about the effect of delays due to a temporary lack of cells at the edges). The trickle-down property will trickle down the latest result of $Pl\text{-}trans_k$ to the lowest level.

Definition 11.4 (Carrying the payload) In a robust medium M, recall the transition function $t = Tr(\mathbf{r}, \mathbf{a})$ from (10.4). with input vector $\mathbf{r} = (r_{-1}, r_0, r_1)$ and the adjacency bits $\mathbf{a} = (a_{-1}, a_1)$. We will say that it *carries* the payload transition function *Pl-trans* if the condition $t \neq Vac$, $r_0.\mathsf{Kind} = Germ$, $\mathbf{a} = (1, 1)$ implies $t.\mathsf{Payload} = Pl\text{-}trans(\mathbf{r}.\mathsf{Payload})$. ⌟

Thus, the payload field is controlled by its own transition function in a germ provided that the simulation does not command to eliminate it. In a non-germ cell, its value will trickle down from the large cell its colony simulates.

The following concept will also be needed for the definition of amplifiers. Here, and always from hence, when we refer to a natural number k as a binary string we always mean its standard binary representation.

Definition 11.5 (Uniform program) 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 are the parameters involved. The definitions

$$\mathbb{S}_k = \{0, 1\}^{Cap_k}, \quad B_k = \prod_{i < k} Q_i$$

are not new: Cap_k is the number of bits in a state in $\mathbb{S}_k$, and B_{k+1} is the size of a colony consisting of Q_k cells of size B_k. Recall that $T_{\bullet k} < T_k^\bullet$ are the lower and upper bound on the dwell periods. Let

$$D_k = 2T_k^\bullet \tag{11.1}$$

for the parameters D_k in trickle-down (Definition 5.11). For simplicity, we will require

$$\lambda := \sup_k T_k^\bullet / T_{\bullet k} < 2. \tag{11.2}$$

The number

$$U_k' \tag{11.3}$$

approximates the (variable) number of dwell periods of M_k in a work period of simulation of a cell of M_{k+1}, (only approximates, as the size of dwell periods is not fixed). For a constant

$$R > 0 \tag{11.4}$$

to be fixed large enough later, let

$$T_{\bullet k+1}/T_{\bullet k} = U_k'(1 - RQ_k/U_k'), \quad T_{k+1}^{\bullet}/T_k^{\bullet} = U_k'(1 + RQ_k/U_k'). \tag{11.5}$$

We will see that U_k'/Q_k grows fast, and thus both $T_{\bullet k}$ and $T_k^{\bullet}$ grow almost like $\prod_{i<k} U_i'$, only $T_{\bullet k}$ grows slightly slower, and $T_k^{\bullet}$ slightly faster. Now (11.2) can be achieved for example by setting $U_k' \geq ck^2 Q_k$ for some large c.

Due to the aggregation property and Proposition 5.17, only a small fraction of the capacity of each cell can be used for computation and communication during a work period. The new parameter

$$w_k \tag{11.6}$$

called *bandwidth rate* will approximately indicate this fraction. We assume that

$$P_k = 1/w_k \tag{11.7}$$

and $w_k Q_{k-1}$ are integers, and therefore so is $w_k B_k$. The payload transition function on level k will be an aggregated version of *Pl-trans$_1$* with slowdown, as in Example 4.21:

$$Pl\text{-}trans_k = Pl\text{-}trans_1^{B_k, w_k}.$$

Thus, a cell of M_k of body B_k, simulated by B_k cells of M_1, should contain all the payload of the simulating cells, and the payload transition function will be slowed down by a factor w_k.

The colony will be processed in chunks of $w_k Q_k$ cells, and we want to be able to correct a burst of constant number off cells in each of these. This requires that information be stored with a redundancy of at least a few cells per chunk. So we devote a portion

$$\delta_k = R w_k/Q_k$$

of the information to redundancy. Let us fix some arbitrary positive initial constants:

$$T_{\bullet 1} \leq T_1^{\bullet} \leq 1.5 T_{\bullet 1},$$
$$\varepsilon_1' < 1/R^2,$$
$$\varepsilon = \varepsilon_1.$$

The interpretation of the parameters $\varepsilon_k, \varepsilon_k', \varepsilon_k''$ has been seen before. In the definition of robust media in Section 10, ε_k bounds probability of occurrence of damage, and $1/2 + \varepsilon_k'$ bound the probability of an outcome of the random coin. The bound ε_k'' has been seen in connection with trickle-down in Definition 5.11. Here are the recursive definitions:

$$\varepsilon_{k+1} = c_{Dam}(Q_k U_k' \varepsilon_k)^2,$$
$$\varepsilon_k'' = 4 Q_k U_k' \varepsilon_k,$$
$$\varepsilon_k' = \varepsilon_1' + \sum_{i=1}^{k-1} \varepsilon_i''. \tag{11.8}$$

The formula for ε_{k+1} is natural in view of Lemma 10.5, which introduces the constant c_{Dam}. The definition of ε_{k+1}' takes into account the limited ability to simulate a coin-toss with the help of other coin-tosses. The parameter ε_k'' bounds the probability that there is any k-level damage at all during the work period of a colony of k-cells.

Let us choose

$$w_k = 1/Rk^2,$$
$$Q_k = R^{k+1},$$
$$U_k' = RQ_k/w_k, \tag{11.9}$$
$$Cap_k = B_k(1 + \delta_k),$$

where we omitted integer parts. Let us by induction, prove for sufficiently small ε:

$$\varepsilon_k \leq \varepsilon^{2^{k-2} + 2^{(k-3)/2}}. \tag{11.10}$$

For $k = 1$, the statement gives $\varepsilon \leq \varepsilon$. For $k > 1$, using the inductive assumption,

$$Q_k U_k' = R^{2k+4} k^2$$
$$\varepsilon_{k+1} \leq \varepsilon^{2^{k-1} + 2^{(k-1)/2}} \cdot c_{Dam} R^{4k+8} k^4$$
$$= \varepsilon^{2^{k-1} + 2^{(k-2)/2}} \cdot c_{Dam} R^{4k+8} k^4 \varepsilon^{2^{(k-2)/2}(2^{1/2} - 1)}.$$

For small enough ε, the last factor is less than 1. From here, it is easy to see that ε_k'' also converges to 0 with similar speed.

The parameters introduced satisfy the requirements below, which can be compared to the corresponding conditions for Theorem 9.3.

Complexity Upper Bounds All parameters in *Frame* are uniform sequences with complexity coefficient R.

Capacity Lower Bound Cells should be able to hold numbers comparable to the size of the colony and the work period within a cell, and the colony must represent the state of the big cell with some redundancy:

$$Cap_k \geq R \log U'_k, \tag{11.11}$$

$$Q_k Cap_k \geq Cap_{k+1}(1 + \delta_{k+1}). \tag{11.12}$$

Bandwidth Bounds

$$w_k Cap_k \geq R \log U'_k, \tag{11.13}$$

$$w_k Q_k Cap_k > w_{k+1} Cap_{k+1}, \tag{11.14}$$

$$w_{k+1} Q_k \geq R(1/w_k + 1/w_{k+1}). \tag{11.15}$$

(11.13) says that numbers comparable to the size of the colony and the work period should fit within the space bound $w_k Cap_k$. (11.14) says that a track of a colony of relative width w_k should be able to hold the portion w_{k+1} of the state of the represented big cell, for processing. The easily satisfiable inequality (11.15) (as Q_k will grow much faster than $1/w_k$) will be used in Section 19.1.

Work Period Lower Bound There must be enough dwell periods in a work period to perform the necessary computations of a simulation:

$$U'_k \geq R \cdot Q_k/w_k. \tag{11.16}$$

Error Upper Bound The following bound relates colony and work-period size to the error probability, and will help in the recursive error estimates:

$$\varepsilon_k^{0.2} \leq 1/RQ_k U'_k. \tag{11.17}$$

It will be easy to satisfy in our examples, since Q_k, U'_k will grow only exponentially and ε_k will decrease super-exponentially.

Lemma 11.6 *The parameters satisfy the conditions above.*

Proof. Conditions (11.11), (11.13), (11.15) and (11.16) are satisfied for large R. For inequality (11.12), assuming $R > 2$, note first that

$$\frac{\delta_{k+1}}{\delta_k} = \frac{Q_k w_k}{Q_{k+1} w_{k+1}} = \frac{(k+1)^2}{k^2 R}.$$

$$Cap_{k+1}(1 + \delta_{k+1}) = B_{k+1}(1 + \delta_{k+1})^2$$
$$= B_{k+1}(1 + \delta_{k+1}(2 + \delta_{k+1}))$$
$$= B_{k+1}(1 + \delta_k \frac{(k+1)^2}{k^2 R}.(2 + \delta_{k+1}))$$

The multiplier of δ_k on the right-hand side is smaller than 1 for large R. Condition (11.14) reduces to

$$1 + \delta_k > (1 + \delta_{k+1})k^2/(k+1)^2,$$

which is true. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\square$

For later reference, here are explicit expressions for B_k and for a quantity falling between $T_{\bullet k}$ and $T_k^\bullet$:

$$B_k = \prod_{i<k} Q_i = R^{k(k+1)/2-1},$$
$$T_k^\bullet \stackrel{*}{=} \prod_{i<k} U_i' = B_k R^{k-1}((k-1)!)^2 = R^{(k+1)(k+2)/2-2}((k-1)!)^2. \tag{11.18}$$

Lemma 11.7 (Amplifier) *For large enough R there is a uniform amplifier complex with the parameters introduced above, which also obeys the following conditions:*

a) *Medium*

$$M_k = Rob(Tr_k, B_k, T_{\bullet k}, T_k^\bullet, \varepsilon_k, \varepsilon_k'). \tag{11.19}$$

   is robust.

b) *The damage map of the simulation Φ_k is given as in Definition 10.3.*

c) *The function Tr_k carries the payload transition function Pl-$trans_k$ in the sense of Definition 11.4.*

Lemma 11.7 implies a variable-period generalization of Lemma 5.12 with $D_k = T_k^\bullet$, which, via essentially the same proof as the one after Lemma 5.12, implies Theorem 7.5 for the infinite space. All properties but the initial stability property of an abstract amplifier (defined in Lemma 5.12) 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 bounds $T_{\bullet k}, T_k^\bullet$. Therefore, if η^k is a trajectory of M_k and $t < T_k^\bullet$ 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 $\{x\} \times (0, T_k^\bullet]a$. This can be bounded by the Restoration Property.

12 Outline of the program

Our eventual goal is to prove Lemma 11.7 (Amplifier). From now on, we fix the level k of the simulation hierarchy, refer to medium M_k as M and to M_{k+1} as M^*. The subscript will be deleted from all parameters of 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*. 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 Section 9.3. We will introduce a fair number of fields but they are all relatively small.

Let us fix some conventions on the handling of fields and the different parts of the transition function. According to Section 11, the medium M has a special field called *Payload*. Together with some other fields, it can be grouped into a field we could call *Info*: in a colony, the *Info* track contains the string that encodes the state of the represented cell via an error-correcting code. Some other fields like *Addr*, will service the functioning of the simulation program.

Definition 12.1 (Info track) Field *Info* consists of the subfields *Info.Payload*, *Info.Pl-redun*, *Info.Util* (referred to mostly as just *Payload*, *Pl-redun*, *Util*). The track *Payload* contains the intended original information, The track *Pl-redun* contains "error check bits" for the track *Payload* as in Example 5.14, while the *Util* track contains other parts of the represented information of the simulated colony (along with its own error checks). ⌟

In what follows we define the function Tr: the function $Pl\text{-}trans$ is given in advance. The field *Payload* is updated via Definition 11.4: thus, Tr determines the next value of *Payload* only in case of non-germ cells via trickle-down.

For the error-correcting code, the track *Payload* $\cup$ *Pl-redun* will be subdivided into "packets" such that the parity check bits for each packet will be computed separately. This is done because the *Payload* track will be much wider than the work tracks, so it cannot be processed in its entirety at once on them.

Cells have an address field *Addr* which determines the only *colony* (Q-colony) to which the cell belongs. A colony $\mathcal{C}(y)$ has base y. The *Age* field of a cell, called its *age*, can have values in $[0, U)$, where $U = U'/p_1$ (which we can assume to be integer without loss of generality). Here U' is the parameter U'_k

introduced in (11.3), and p_1 is a constant to be introduced in Definition 12.10 below.

12.1 Cell kinds

Definition 12.2 (Adjacency) Recall the transition function $Tr(\mathbf{r}, \mathbf{a})$ where $\mathbf{a} = (a_{-1}, a_1)$ says whether the left and right neighbor is adjacent. In our rule language we will refer to a_j as

$$\text{adj-nb}(j).$$

⌟

We will continue to understand by the word *cell* a non-vacant, non-damaged site.

Definition 12.3 (Colonies) The values of the address field *Addr* will vary in $[-Q, 2Q)$. The *colony* of a cell x is the set $C = \{z + iB : i \in [0, Q)\}$ whose starting cell is $z = x - (\textit{Addr}(x) \bmod Q)B$. We will also say that x *belongs* to colony C. The *originating colony* of a cell x is the colony D whose starting cell is $x - \textit{Addr}(x)B$. We will also say that x *originates* at colony D. The cells whose colony is their originating colony, will be called *inner cells*, the other ones will be called *outer cells*. Two neighbor cells will be called *space-consistent* if they originate at the same colony. ⌟

A certain property of cells, called their "kind", plays an important role in determining their behavior.

Definition 12.4 (Cell kinds) Cells will be of a few different *kinds*, distinguished by the field

$$\textit{Kind}.$$

The possible kinds are listed in (12.1) below. *Member* The kind of non-latent, non-germ, cells is determined by *Age* and *Addr*. Since it will be possible to determine from the age of a cell whether it has kind *Channel* or *Growth*, We will sometimes use therefore just one value

$$\textit{Ext}_j$$

in place of $\textit{Growth}_j$ and $\textit{Channel}_j$. The kinds of cells are ordered by a property called *strength* as follows:

$$
\begin{aligned}
\textit{Vac} &< \textit{Latent} < \textit{Germ} < \textit{Channel}_{-1} < \textit{Channel}_1 \\
&< \textit{Growth}_{-1} < \textit{Growth}_1 < \textit{Member}.
\end{aligned}
\tag{12.1}
$$

If a cell needs to be created, overlapping in body a another one then if the new cell is stronger, the weaker one will be erased. This is the only way a cell can turn vacant see the later Condition 10.11 (Cling-to-Life). Let us discuss each kind.

- The relation $Kind(x,t) = Vac$ means that there is no cell at site x at time t.

- A cell will be called *dead* if it is vacant or latent, and *live* otherwise. *Killing* a cell x means turning it latent. *Erasing* it means turning it vacant.

- The *member* cells are "inner" cells as defined above; they are the strongest kind in order to maintain the integrity of colonies.

- Cells of kind Ext_j are outer cells. A *right outer* cell has addresses $\geq Q$, and a *left outer* cell has addresses < 0.

- Left channel cells are weaker than right channel cells which are weaker than growth cells. This way, a right-growing channel will erase the left-growing one if it is in its way, and information need not be transmitted between two channels. Also, left channel cells can grow only until address $-Q + 1$, and right ones only until $2Q - 2$: thus a channel cannot be extended to cover a full colony.

- *Germ* cells exist at the highest level of the hierarchy, in the sense that they are not part of any colony simulating some higher-level cell. They also have the function to carry out a computation, the *payload* of the whole construction. The member cells on lower levels will receive the results of this computation by a feature of the simulation called *trickle-down*. Germ cells have addresses in $[-2Q, 3Q)$.

In the construction leading to Theorem 3.4, germ cells will play the following role. Adjacent and "consistent" germ cells of the same color will attempt to expand and form a new colony simulating a big germ cell: we call this "lifting", or "self-organization". The part of the space marked with color 0 is where the actual payload computation takes part. The parts of space to its left and right are marked by color -1 and 1 respectively. These pars are also populated by germ cells which also self-organize (this needs randomization to break the translational symmetry). The germs of the outside areas, with their appropriate colors, will suppress possible structures that would compete with the computation area.

The transitions between different kinds are limited in advance.

Condition 12.5 (Latent Cells) A vacant cell can only turn into a latent one.

The following condition helps enforce that newly created cells in the simulation are latent.

Condition 12.6 Suppose that $\varphi^*(\eta)(x,t) \notin Bad$, 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 affected by damage (as in Definition 10.15). 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 decoded from this track is latent. ⌟

12.2 A colony work period

Algorithm 12.1 describes the main stages of a colony work period. In the notation here, it sometimes does not matter much whether we call one of the rules here a "sub-rule" or not, and whether we print the semicolon, as the rule (like *Extend*) includes the conditions under which it is applied.

Algorithm 12.1: A colony work period

```
Extend ∥ Retrieve;
Compute;
Proc-payload;
Grow;
Shrink;
Finish
```

There are some idle stages between these parts to make sure that faults in one part have limited effect on other parts. Rule *Extend* is defined in Algorithm 14.9, *Retrieve* in Algorithm 19.4, *Compute* in Algorithm 19.7, *Proc-payload* in Algorithm 19.11, *Grow* in Algorithm 14.10, *Finish* in Algorithm 19.12. Shrinking (if needed) happens via the rule *Decay* defined in Algorithm 15.1.

The stages are started and ended at certain specific values of the field *Age*, which runs between 0 and U, the maximum age. Recall for some k we are defining medium M_k of and amplifier. Because of a delay parameter p_1 to be defined below in (12.7), parameter $U = U_k$ is related to the estimated number U'_k of dwell periods in a work period introduced in (11.3) as follows:

$$U'_k = p_1 U_k. \tag{12.2}$$

Here are the ages starting or ending important stages of this program:

$$\text{compute-start} = KRP^*Q,$$
$$\text{proc-payload-start} = \text{compute-start} + KRP^*Q,$$
$$\text{grow-start} = \text{proc-payload-start} + RP^*Q, \qquad (12.3)$$
$$\text{grow-end} = \text{grow-start} + 6Q\lambda,$$
$$U = \text{grow-end} + 4Q,$$

where λ was defined in (11.2), U is the number of *Age* steps in the work period, K is a constant to be specified later, the constant R was introduced in Section 11, and P was defined in (11.7).

Definition 12.7 (Expansion and retrieval periods) The interval before age compute-start is called the *retrieval period*: then the colony retrieves information from neighbor colonies. The constant K will make it substantially longer than the transition period (grow-start, U] at the end of the work period when the information to be sent to neighbor colonies would change. The age intervals (0, compute-start] and (grow-start, grow-end] for non-germ outer cells and (0, grow-end] for germ cells will be called *expansion periods*. Cells in their expansion period are called *expansion cells*. ⌟

The following short description of the major stages of a colony work period should serve for orientation. The part of the work period before `Compute` sees the following activities happening simultaneously: `Extend`, and `Retrieve`.

Expansion The rule `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 $Channel_j$. 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. Extension cells are weaker than member cells, so channel or growth cells do not normally damage another colony. The channels will be killed at the end of the computation.

Communication The rule

<div align="center"><code>Send</code></div>

defined in Algorithm 19.3, will be running most of the time, sending all needed information to the neighbors. The rule `Retrieve` records the information from the neighbors. Atomicity (Condition 10.16e)) for the simulated

medium will be guaranteed by waiting for a time when neither of the neighbor colonies is near the end of their work period—when this information might change.

Computation The subrule `Compute` computes the output of the simulated transition function and stores it on the track *Hold*. It will doom each cell of the colony if the represented cell is to be erased at the end of the work period.

Payload processing This part carries out the part of the computation not needed for simulation but for the information-processing task of the cellular automaton.

Growth If $Growing_j = 1$ then, between values grow-start and grow-end, the colony tries to extend an arm of length at most Q in direction j, making it possible to create a new colony if the encountered area is empty.

Birth A latent cell x turns immediately into a germ cell with address 0 and age 0. The *germ* then begins to grow, trying to fill 3 colonies until age grow-end. A germ that fails to grow with a certain speed will be killed off. If it succeeds then at $Age = U - 1$, the germ cells turn into member cells, thus implementing a lifting of the organization level.

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 (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. The rule `Send` will not run during this transition time.

Finish Rule `Finish`, will called at $Age = U - 1$, reduces the addresses of growth cells (if any remain) modulo Q. Outer non-germ cells and inner germ cells turn into members. Doomed cells will be killed. Otherwise, the information from the *Hold* track will be copied into the corresponding locations on the *Info* track.

Definition 12.8 The application of the rule `Finish` will also be called a *cut*. ⌟

Doomedness can be changed only in specific ways:

Condition 12.9 (Dooming) Only the following rules can change the field *Doomed* without killing the cell:

1) The sub-rules `Adapt` and `Heal` (defined later) can create doomed or non-doomed cells.

2) At $Age = \mathsf{proc\text{-}payload\text{-}start}$, we possibly doom each cell.

⌟

12.3   Timing

Different actions of the program will be carried out with different speeds; this will be achieved as follows. Some rules r have a *delay parameter p* and a timing variable $\mathit{Wait}_r \geq 0$, used as follows:

> **if** $\mathit{Wait}_r = 0$ **then** $\mathit{Wait}_r \leftarrow p$
> **else if** $\mathit{Wait}_r \geq p$ **then** r
> **else** $\mathit{Wait}_r \leftarrow \mathit{Wait}_r - 1$.

A single assignment

$$F \leftarrow_p c$$

can also be seen as a rule, with its own delay parameter p and wait variable.

Definition 12.10 Recall the definition of λ in (11.2). We introduce some delay constants $p_0 < p_1 < p_2 < p_3$ whose value will be defined below, further

$$\tau_i = (p_i + 1)T^{\bullet}. \tag{12.4}$$

⌟

Here is a summary of the roles of the delays:

p_0 Default and healing;

p_1 Computation;

p_2 Decay;

p_3 Growth.

For defining the delay constants let

$$\mathsf{heal\text{-}span} = 2\Delta + 1, \tag{12.5}$$

denote the reach of the healing operation. Let

$$p_0 = 5\lambda, \tag{12.6}$$
$$p_1 = 4\lambda p_0, \tag{12.7}$$
$$p_2 = 2\,\mathsf{heal\text{-}span}\,\lambda p_0, \tag{12.8}$$
$$p_3 = (3\lambda + 1)p_2. \tag{12.9}$$

12.4   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.

Let us outline how local repairs will be made to deal with obstacles of functioning; details follow in later sections. The rule *Purge* eliminates isolated cells, or "exposed" cells that are not important to heal. The rule *Heal* repairs a small hole of colony cells (typically after *Purge* eliminated "garbage" from it.). Germs, and arms of communication or growth will not be healed. An unrepaired hole will be slowly enlarged by the rule *Decay*, to eventually eliminate partial colonies.

An island can destroy or alter information represented on its space projection, therefore the information represented on the *Info* track will be a redundant, error-correcting code. It 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* before the final step of the work period commits to it.

The crucial Lemma 16.3 (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). This lemma enables us to reason about the process over this area in terms of big cells. It helps us for example to see 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: 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 (extension or germ cells), 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" backwards in time until time $t_n = t_0 - mQ$

with some large coefficient m. Lemma 15.5 (Skirting) shows that it is possible to lead a path around an island of damage. 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 decay rule, 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 Skirting Lemma. Lemma 15.12 (Running Gap) says that if a gap is large enough then the decay process propagates it fast, even in the presence of some damage.

– Lemma 15.17 (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 rest of the proof.

• We define those computation rules not dependent on communication with neighbor colonies.

• Lemma 20.2 (Legality) shows that the computation terminates gracefully independently of the success of communication.

• The development of colony $\mathcal{C}$ will be followed forwards to the present in Lemma 20.6 (Present Attribution) .

• Finally, the retrieval rules will be defined and the remaining part of the Computation Property will be proved.

13   Local consistency

Functional obstacles created by an island need to be recognized; we develop the tools for this in the present section.

13.1   Local maintenance

Information in a colony about the represented big cell will be corrected using decodings. But on certain tracks, inconsistency be corrected almost instantaneously.

Definition 13.1 (Locally maintained fields) Certain fields, called *locally maintained*, will be kept constant over the colony, for most of the work period. Each such field F has a *default value*. The updating will of such a field always happen at a specific age called the *update age*. Let

$$n_{\text{l-m}} \tag{13.1}$$

be the (constant) number of update ages for locally maintained fields. A cell's locally maintained field F is said to be *stable* if its age is at distance $\geq \Delta$ from the update age, where this constant is defined below in (15.3). An age n is called *stable* for F if F is stable at age n. It is called simply *stable* if it is stable for all locally maintained fields, and is also at distance $\geq \Delta$ from ages 0, compute-start and grow-end. ⌟

From the definitions above it follows that within a work period, all unstable ages are covered by $n_{\text{l-m}}$ intervals of size 2Δ each.

Example 13.2 Here are some examples of locally maintained fields.

- The Boolean track *Doomed* will be set to 1 (true) if the represented big cell must be removed (the site to become vacant), and 0 otherwise. Its default value is 0 (false). Its update age is proc-payload-start (defined in (12.3)). Cells with *Doomed* $= 1$ are called *doomed*.

- The track
$$\textit{Growing}_j \in \{0, 1\},\ j \in \{-1, 1\}$$

signifies the collective decision of the colony to grow a new neighbor colony in direction j. Its update age is proc-payload-start. The field

$$\textit{Creating}_j \in \{0, 1\},\ j \in \{-1, 1\}$$

will be used to control the creation of a new neighbor. Its default value, in a vacant or new latent cell, is 0. *Creating*$_j = 1$ makes a cell a potential creator in direction j in the sense of Definition 10.12. The value of *Creating*$_j$ of a big cell will be broadcast into the locally maintained track *Growing*$_j$ of the cells of its representing colony.

- The track *Pl-commands* will contain commands for the processing of payload: see Section 19.

⌟

Condition 13.3 (Locally maintained fields) Age 0 is an update age for all locally maintained fields other than *Doomed*. If *Doomed* $= 1$ then *Growing*$_j = 0$ for $j \in \{-1, 1\}$. ⌟

13.2 Fitting neighbors

Color   We introduced the *Color* field in Definition 3.5. It distinguishes between germ cells, but can also serve as the only information conserved by the non-ergodic medium. Even latent cells have color; when a cell becomes latent its color does not change.

Definition 13.4 Two neighbor cells $x < y$ with colors c, d will be expected to have *fitting* colors. This is expressed by the relation $Fit_1(c, d)$ which is 1 if the colors fit and 0 otherwise. Let

$$Fit_{-1}(c, d) = Fit_1(d, c).$$

We will use two examples of the fitting relation:

1) A color can be an arbitrary symbol from a finite alphabet, and $Fit_j(c, d) = 1$ if and only if $c = d$.

2) A color is an element of $\{-1, 0, 1\}$, and $Fit_1(c, d) = 1$ if $c \le d \le c + 1$. In what follows we concentrate on the second case, as the other case is treated analogously, only simpler. The function of color is in this case is the following: the area of space where payload computation is happening is marked by color 0, while the space to its left and right by colors -1 and 1 respectively.

⌟

Variant 1) will be used in the proof of Theorems 3.1, 3.2 and 7.4. Variant 2) will be used in the proof of Theorems 3.4, 7.6.

Siblings   The basic structural soundness of a colony is expressed by some local consistency conditions. Space-consistency was introduced in Definition 12.3. Time consistency will also be required, but in a continuous-time cellular automaton we cannot require all cells to have the same age. Requirement (13.2) says that the age within an extended colony is highest at the position of address $\lfloor Q/2 \rfloor$ and is non-increasing (with age difference ≤ 1 between neighbors) as we move away from it.

Definition 13.5 (Time consistency) Two cells x and $y = x \pm B$ with $|Addr(x) - \lfloor Q/2 \rfloor| < |Addr(y) - \lfloor Q/2 \rfloor|$ belong to the *same work period* if

$$0 \le Age(x) - Age(y) \le 1. \tag{13.2}$$

They *straddle a work period boundary* if $Age(x) = 0$, $Age(y) = U - 1$. If one of these cases holds then we will say they are *time-consistent*.   ⌟

The most important consistency requirement, in the concept of siblings, is a combination three kinds of consistency: of space- and time-consistency, and agreement in stable locally maintained fields.

Definition 13.6 (Siblings) Two cells $x, x + jB$ for $j \in \{-1, 1\}$ are *siblings* if one of the following properties holds.

1) They belong to the same work period, originate at the same colony (see the definition of originating colony at the beginning of Section 12.1), either both are germ cells or neither of them is. If x is a colony cell and $x + jB$ is a growth cell then **Growing**$_j(x) = 1$ is also required.

2) They belong to the same colony (see the definition of the colony of a cell at the beginning of Section 12.1), and straddle a work period boundary.

3) They have fitting colors.

4) If they are germ cells with G-*growing*(**Age**) then they agree in their **G-size** and **Dominant** fields (see Section 14.4).

⌟

The sibling relation, unlike in biology, is not transitive. It would be more appropriate, but more awkward, to use the term "half-sibling".

Definition 13.7 (Domains)

$$\text{max-depth} = \text{heal-span} \tag{13.3}$$

with value defined in (12.5). An interval of cells in which the adjacent cells are siblings will be called a *domain*. A domain of size n will also be called a *n-support* of its members. For integer $k > 0$, let us call two cells $x, x + kB$ *relatives* if we can change the states of cells $x + jB$ for $j = 1, \ldots, k-1$ in such a way that the cells $x, x + B, \ldots, x + kB$ become a domain. A colony with starting cell x will be called *full* if it is covered by a domain in such a way that **Addr**$(x + iB) \equiv i \pmod{Q}$. ⌟

So if a color is an element of $\{-1, 0, 1\}$, and $Fit_1(c, d) = 1$ if $c \leq d$ then a domain is either has just one color, or starts with a -1's, continues with 0's and ends with $1's$. All of these can be empty, except that if both -1 and 1 is present then some 0's must also be there.

For a cell x, the distance of the boundaries of a domain containing x is estimated by the concept of depth. For the purpose of depth, we only consider domains within a single colony.

Definition 13.8 (Depth) Let us call the *left depth* of a cell x the size of the largest domain of the form $\{x - iB, x - (i-1)B, \ldots, x\}$ within a single colony

containing x. The right depth is defined similarly. For example, $depth_{-1}(x) > 1$ just means that x has a left sibling.

For a locally maintained field F we define the *left depth $depth_{j,F}(x)$ for F* of a cell x the size of the largest interval of the form $\{x - iB, x - (i-1)B, \ldots, x\}$ within a single colony containing x and having constant value of F.

A cell will always try to keep track of its depths, up to the maximum possible value. It will use the fields

$$Depth_j, Depth_{j,F} \in \{1, \ldots, \mathsf{max\text{-}depth}\}, \; j \in \{-1, 1\}.$$

So if for example left depth is larger than max-depth then the cell may only see $Depth_{-1} = \mathsf{max\text{-}depth}$.   ⌋

The condition $depth_j = 1$ means that in direction j either there is a colony boundary or a domain boundary. On the other hand the condition $Depth_j = 1$ is a statement about the field $Depth_j$, maintained by observing $depth_j$ using the rule `Watch-depth` in Algorithm 13.1. Let

$$\Delta = 5 \tag{13.4}$$

denote the number of cells directly affected by damage.

Algorithm 13.1: rule `Watch-depth`

> **pfor** $j \in \{-1, 1\}$ **do**
>> **if** $depth_j = 1$ **or** $Addr = e_j$ **then** $Depth_j \leftarrow_1 1$
>>
>> **else** $Depth_j \leftarrow_1 \mathsf{max\text{-}depth} \wedge (Depth_j^j + 1)$
>>
>> **forall** locally maintained fields F **do**
>>> **if** $depth_{j,F} = 1$ **or** $Addr = e_j$ **then** $Depth_{j,F} \leftarrow_1 1$
>>>
>>> **else** $Depth_{j.F} \leftarrow_1 \mathsf{max\text{-}depth} \wedge (Depth_{j,F}^j + 1)$

Rule `Loc-maintain` locally maintains all locally maintained variables.

Algorithm 13.2: rule `Loc-maintain`

> **pfor** $j \in \{-1, 1\}$ **do**
>> **forall** locally maintained fields F **do**
>>> **if** Age is stable for F **and** $Depth_{j,F} = \mathsf{max\text{-}depth}$
>>>
>>> **and** $Depth_{-j,F} < \mathsf{max\text{-}depth}$ **then**
>>>> $F \leftarrow F^j$

Age updating   The rule for updating age is similar to the "marching soldiers" rule for updating *Age* in Section 8. In Definition 13.5, we imposed some extra order on the age of all cells in the same extended colony: it must be non-increasing as they become more distant from the center cell of the originating colony. While healing some inconsistency, age updating is paused with the help of a field called *Frozen*. This variable will also point to the place of nearby inconsistency that caused the freezing, In pointing to the position of inconsistency, playing a role somewhat similar to the fields $Depth_j$.

Definition 13.9 There is a field

$$Frozen \in [-\Delta, \Delta] \cap \mathbb{Z}$$

with the property that when $Frozen \neq 0$ then *Age* will not be changed. A cell with $Frozen \neq 0$ is called *frozen*.   ⌟

Here is the basic updating rule for age:

Algorithm 13.3: sub-rule `March`, delay p_1

    if $Frozen = 0$ **and** $Age < U - 1$ **and**
(1) the increase of *Age* does not break the sibling relation with any
      neighbor **then**
        $Age \leftarrow Age + 1$

The main colony-organizing variables, address and age, can be changed only by very few rules.

Condition 13.10 (Address and Age)   a) Only `Finish` can change *Addr* of a live cell.

  b) Only `March`, `Heal-sync` and `Finish` can change *Age* of a live cell. Of these, `Heal-sync` can do it when the cell is frozen and the others when it is not.

                                                         ⌟

13.3   Edges

Definition 13.11 (Edges) Suppose that cell x has no siblings in direction j, that is it has $depth_j(x) = 1$. 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*. Recall expansion periods and cells in Definition 12.7. Essentially, growth edges are protected during

the expansion period or if the growth reached a colony end. Here is a list of the types of protected edges, for non-germ cells. For germ cells, the definition will be given in Section 14.4.

Member Colony end-cell towards j.

Expansion cell During the expansion period, while j is the direction of expansion. After the expansion period, if the cell is a growth cell and a colony end-cell towards j.

In a rule, the condition

$$Xposed_j.$$

means that the cell **x** applying the rule is an exposed edge in direction j. ⌟

An exposed edge is the sign of defect, or a call to eliminate an extension of a colony or a colony; it may be killed—fast by the purge rule, or slow by the decay rule.

Lemma 13.12 *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. The one case when this is not obvious is when the exposed edge is a left outer cell past its expansion period but its neighbor may not be. But the address-dependent nature of expansion periods as introduced in Definition 12.7 forces then the right sibling also to be past its expansion period. □

The following notion will be needed much later:

Definition 13.13 (Multi-domain) A *multi-domain* is one of the following kinds of set:

1) A domain.

2) The union of some adjacent domains meeting in protected colony end-cells.

⌟

From the above definitions it is clear that only a cut (as in Definition 12.8) can turn a domain into a multi-domain. The following lemma is an immediate consequence of the definition of protected edges. Its technical exceptions relate to the part of the program, detailed later, governing the growth of germs. A germ will start from a "leading" germ cell with address $Q/2$, and needs to grow to at least a size 3 without seeing live neighbors in order to stay alive.

Lemma 13.14 *If some maximal multi-domain has size $< Q$, then the following holds:*

a) *One of its edges is exposed, with the following exception: it consists of germ cells in their expansion period, contains a cell with address $Q/2$, and the size is either > 3, or 3; in the latter case the adjacent neighbor cells of the domain are not live.*

b) *If the size is $\leq Q - d$ with $d > 0$, then at least one the exposed edges cannot become protected by the death of an outside neighbor, or by shrinking, or by growing by fewer than d steps. The only exception is a germ of size 3 of the above kind: it may become protected if a neighbor dies.*

14 Killing and creation

In order to eliminate local functional obstacles, information in some cells will need to be erased and then replaced with one in harmony with one of its neighbors.

14.1 Killing

As said above, a cell will be "killed" by making it latent. It will advertise its death in advance:

Definition 14.1 We will have some one-bit fields

$$Dying_j, \; j \in \{-1, 0, 1\},$$

with default value 0. We will say that a cell x is *dying* if $Dying_0(x) = 1$. ⌟

In fact, $Dying_0 = 1$ announces that the cell will die soon. The fields $Dying_j$ for $j \in \{-1, 1\}$ try to keep track of whether a neighbor is dying, which will allow a cell to see even whether a second neighbor is dying. They are maintained by rule `Watch-dying` in Algorithm 14.1 which acts with the minimal delay.

Algorithm 14.1: rule `Watch-dying`

> **pfor** $j \in \{-1, 1\}$ **do**
> > **if** $Kind^j \neq Vac$ **then** $Dying_j \leftarrow_1 Dying_0^j$

A program will kill a cell via the rule `Die`(p) of Algorithm 14.2: the argument determines the delay. It actually takes $2p$ consecutive applications of the rule `Die`(p) to kill the cell: p applications to set $Dying_0 \leftarrow 1$ and then p more to kill it. The rule `Create` of Algorithm 14.5 will disqualify a dying cell from creating a live neighbor.

Algorithm 14.2: sub-rule $\texttt{Die}(p)$

> $Dying_0 \leftarrow_p 1$;
> **if** $Dying_0 = 1$ **then** $Kind \leftarrow_p Latent$

Algorithm 14.3 ($\texttt{Purge}$) kills exposed cells that are not to be healed: either because they belong to small maximal domains thus are isolated), or expansion arms, or to a doomed area near the end of the work period.

Algorithm 14.3: rule $\texttt{Purge}$

> **cfor** $j \in \{-1, 1\}$ **do**
> > **if** $Xposed_j$ **and** $(Depth_{-j} \leq \Delta$ **or** $Kind \neq Member$
> > **or** $(Doomed$ **and** $Age > U - 2Q))$ **then**
> > > $\texttt{Die}(p_1)$

14.2 Birth, creation, adaptation

Rule $\texttt{Birth}$ in Algorithm 14.4 is just enforcing the condition that newborn cells are latent. In simulation, birth will be implemented for big cells when germ cells succeed in creating a new colony.

Rule $\texttt{Create}$ in Algorithm 14.5 controls the values of the field $Creating_j$ in order to avoid creating overlapping cells. A new latent cell is born with $Creating_j = 0$, turning it on with a delay p_0. Part (1) turns off $Creating_j(x)$ if the neighbor $x + jB$ is already there. The last part of the rule is just a constraint on the kind of cell that can be created, namely only a latent one, thereby enforcing Condition 10.10. The cell **x** is not really there to "apply" this part: in the simulation it is observed by a creator neighbor $\vartheta_{-j}(\mathbf{x})$.

We do not have to worry about other strange rules for non-cells:

Condition 14.2 (Birth) $\texttt{Create}$ and $\texttt{Birth}$ are the only rules applicable to a vacant cell. ⌟

The birth condition implies that in all cases different from the ones listed in the rules $\texttt{Create}$ or $\texttt{Birth}$, the site is required by the transition function to remain vacant. Condition 10.16f) allows for the creation to be blocked by a cell whose body intersects the cell to be created.

Consider a cell x and its left nonadjacent neighbor y that may want to create a cell in $y + B$, overlapping the body of x. Whether x will be erased is

Algorithm 14.4: rule `Birth`

> **if** *Kind* = *Vac* **and** the two adjacent neighbors are vacant **then**
> *Kind* ← *Latent*

Algorithm 14.5: rule `Create`

> **cfor** $j \in \{-1, 1\}$ **do**
> (1)   **if** adj-nb(j) = 1 **then** *Creating*$_j$ ←$_1$ 0
> **else** *Creating*$_j$ ←$_{p_0}$ 1
> **if** *Kind* = *Vac* **and** *Creating*$_j^{-j}$ = 1 **then** *Kind* ←$_1$ *Latent*

decided not by whether y is stronger than x but by whether the new cell $y + B$ would be stronger than x. This distinction matters when a colony attempts to create an outer cell that would intrude into another colony. As the created cell would be weaker than the member cells of the other colony with whom it is competing for space, this will not happen. Let us introduce some auxiliary notation, useful for many rules.

Notation 14.3 For a relation R, let

$$a \overset{R}{<} b$$

mean "$a < b$ or ($a = b$ and R holds)".   ⌟

Here is the inequality to be used when deciding whether a non-germ cell x can be overwritten by a neighbor. It says that the neighbor must be non-dying and backed up by a non-dying sibling, the kind of x must be dominated by the kind to be created, and if a neighbor on the other side also wants to overwrite, the intended kind from that side must also be weaker. Ties are decided by preferring a creation towards the left. The competing growth of germs will be regulated later in Section 14.4.

Definition 14.4 Let $j \in \{-1, 1\}$. The field *Kind*$_j$ shows the kind of the cell intended to be created in the adjacent neighbor in direction j. Let

$$\textit{Non-germ-less}_j \Leftrightarrow \textit{Kind}^j_{-j} \overset{j=1}{>} \textit{Kind} \wedge \textit{Depth}^j_j > 1 \wedge \textit{Dying}^j_j = 0$$
$$\wedge (\textit{Kind}^{-j} = \textit{Vacant} \vee \textit{Kind}^j_{-j} \overset{j=1}{>} \textit{Kind}^{-j}_j). \tag{14.1}$$

⌟

For $j \in \{-1, 1\}$, rule 14.6, $\mathit{Adapt}(j)$, makes cell **x** a sibling of its neighbor in direction j. Sub-rule (1) is not really a rule that a cell can execute, since it is referring to the case when it is vacant. Rather, it shows a condition under which the neighbor cell in direction j is a potential creator, as in Definition 10.12. Sub-rule (2) kills a cell if it is in the way of the above creation, sub-rule (3) erases a latent cell fast.

Algorithm 14.6: sub-rule $\mathit{Adapt}(j)$

    if $\mathit{Dying}_0^j = 0$ **and** $\mathit{Creating}_{-j}^j = 1$ **then**

(1)   **if** $\mathit{Kind} = \mathit{Vacant}$ **and** adj-nb(j) **then** $\mathit{Kind} \leftarrow \mathit{Latent}$

(2)   **else if** $\mathit{Kind} \neq \mathit{Latent}$ **then** $\mathtt{Die}(p_1)$

(3)   **else if** $\neg$adj-nb(j) **then** $\mathit{Kind} \leftarrow_1 \mathit{Vac}$

      else make **x** a sibling of $\mathbf{x}^j$, with the same color

---

Definition 14.5 (Parent) If a latent cell x came to life by the rule $\mathit{Adapt}(j)$ then we will say that it has been *animated*. The neighbor in direction j will be called its *parent*. (The latent cell could have been created earlier by a neighbor, which we could call a "creating parent".) In case the same result could also have arisen using the neighbor on the other side then the parent cell is defined as the one closer to the center of its colony (or to the left if the animated cell is the center).   ⌟

The following lemma is immediate from the definition of the adaptation rule.

Lemma 14.6 *Suppose that*

  a) A cell x has just been adapted at time t to a non-germ neighbor $y = \vartheta_j(x)$ (this rule being a possible explanation for its becoming live);

  b) Rectangle $(x + [-2.1B, 3.1B)) \times (t + (-3T^{\bullet}, 0])$ is damage-free;

  c) There is no colony-boundary between y and its sibling required by the rule.

Then x and y stay siblings until after t.

Proof. The adaptation y to be 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 (as in Definition 12.8) will not break the sibling relation of these cells either.   □

The following lemma shows that the rule $\texttt{Create}$ indeed succeeds in creating a new cell. Here, cell $x - B$ will create a cell at site x. Creation from the right is analogous. Recall the notation τ_i from Definition 12.10.

Lemma 14.7 (Creation) *Assume the following, with $J = (t_0, t_0 + \tau_0 + 11T^\bullet]$:*

a) *Rectangle $[x - 4.1B, x + 3.1B) \times J$ is damage-free;*

b) *We have $\eta(x - B, t).\mathsf{Dying}_0 = \eta(x - B, t).\mathsf{Dying}_{-1} = 0$, and*

$$\eta(x - B, t).\mathsf{Kind}_1 \geq \eta(y, t).\mathsf{Kind} \vee \eta(y, t).\mathsf{Kind}_{-1}$$

for all live cells (y, t) in the rectangle $[x, x + 2B) \times J$.

Then $\eta(x, t)$ is non-vacant for some t in J.

Proof. Assume, on the contrary, that x is vacant during all of J, and we will arrive at a contradiction. The conditions imply that noise does not affect cell $x - B$, so it can obey its transition rule. Hence part (1) of the rule $\texttt{Create}$ sets $\eta(x - B, t).\mathsf{Creating}_1 = 1$ at some $t_1 \leq t_0 + 2T^\bullet$, and this stays so while x is vacant. Condition 10.16f) requires x to become a cell by time $t_1 + 2T^\bullet$ unless the body of some cell overlaps the body of x. Suppose therefore that cell y overlaps the body of x before time $t_2 = t_1 + 2T^\bullet \leq t_0 + 4T^\bullet$. Then $y > x$, since in case $y < x$, cell y would overlap cell $x - B$, which is assumed to be there during all this time interval. This y disappears by time $t_3 = t_2 + \tau_0 + T^\bullet = t_1 + \tau_0 + 5T^\bullet$. Indeed, part (2) of $\texttt{Adapt}$ makes y latent within τ_0, then part (3) erases y within $T^\bullet$.

If no similar obstacle cell y' arises before time $t_4 = t_3 + 2T^\bullet = t_1 + \tau_0 + 7T^\bullet$ then again x would be created, so assume one appears. This can only happen if $y' + B$ creates it (and hence is latent). After it does, rule $\texttt{Create}$ of Algorithm 14.5 turns off $\mathsf{Creating}_{-1}$, turning which on again takes time $\geq p_0 T_\bullet$. Now $\texttt{Adapt}$ will erase y' by time $t_5 = t_4 + 2T^\bullet = t_0 + \tau_0 + 9T^\bullet$. No other cell with $\mathsf{Creating}_{-1} = 1$ can arise by time $t_6 + 2T^\bullet = t_0 + \tau_0 + 11T^\bullet$; partly for the same reason and partly because if $y' + B$ is erased and a new latent cell occupies its place then its $\mathsf{Creating}_{-1}$ is turned on only by a delay of $p_0 T_\bullet$, as said in rule $\texttt{Create}$ of Algorithm 14.5. As a result no new obstacle y'' can appear soon after y' has been erased.

Condition 10.16d), allows only one other way for a cell to appear, namely "out of nothing"; but this requires the cell to have no neighbors with a body within distance $2B$ of its body, and as $x - B$ would be such a neighbor, this cannot happen.

$\square$

14.3 Growth

Growth cells as well as germ cells are born in the attempt to create a colony that encodes a latent big cell. Let us spell this out as a condition:

Condition 14.8 If a cell has kind *Growth* or *Germ* then its **Info** field's value is the symbol at **Addr** mod Q in the code of a latent big cell. (If, due to a fault, it does not have that value then this would be immediately corrected in the next step.) ⌟

Due to this condition, we do not have to set the **Info** field explicitly in the `Grow.passive` and `Germ-grow.passive` rules below. Here are the rules of growth for non-germ cells. For germ cells, see Section 14.4. Rules 14.9 (`Extend`) and 14.10 (`Grow`) both rely on the sub-rule 14.7, `Grow.active` to show the intent to adapt a neighbor. The adaptation will be performed by sub-rule 14.8, `Grow.passive` calling the rule `Adapt`. These rules use some fields introduced in Definition 14.4, and the following definition of end-cells:

Definition 14.9 (Endcells) Let

$$e_{-1} = 0, \ e_1 = Q - 1. \tag{14.2}$$

A cell x is a *colony endcell in direction j* if $\mathbf{Addr}(x) \equiv e_j \pmod{Q}$. ⌟

Rule `Extend` in Algorithm 14.9 serves to extend the channel in direction j. Rule `Grow` in Algorithm 14.10 depends on the fields $\mathbf{Growing}_j$ mentioned in Example 13.2. Rule `Loc-maintain` of Algorithm 13.2 keeps the locally maintained field $\mathbf{Growing}_j$ constant throughout the extended colony. We will see later that the computation rule sets $\mathbf{Growing}_j = 1$ in all cells of the colony iff the field $\mathbf{Creating}_j$ in the big cell represented by the colony has value 1. Otherwise, it will be 0 in all cells.

Algorithm 14.7: sub-rule `Grow.active`(j)

> **if** $depth_{-j} > 1$ **and** $((\mathit{Kind} = \mathit{Ext}_j$ **and** $\mathbf{Addr} \bmod Q \neq e_j)$ **or**
> $(\mathbf{Addr} = e_j$ **and** $\mathit{Kind} = \mathit{Member}))$ **then**
> $\quad \mathit{Kind}_j \leftarrow \mathit{Ext}_j$
> **else if** $\mathit{Kind} \neq \mathit{Germ}$ **then** $\mathit{Kind}_j \leftarrow \mathit{Latent}$

Algorithm 14.8: sub-rule $Grow.passive(j)$, delay p_3

    if $Non\text{-}germ\text{-}less_{-j}$ **and** $Kind_j^{-j} = Ext_j$ **then**
       $Adapt(-j)$

Algorithm 14.9: sub-rule $Extend$

    pfor $j \in \{-1, 1\}$ **do**
       if $Kind = Latent$ **and** $Age^{-j} \in (0, \text{compute-start}]$ **then**
          $Grow.passive(j)$
       else if $Age \in (0, \text{compute-start}]$ **and** $Kind \neq Germ$ **then**
      $Grow.active(j)$

Algorithm 14.10: sub-rule $Grow$

    pfor $j \in \{-1, 1\}$ **do**
       if $Age \in (\text{grow-start}, \text{grow-end}]$ **and** $Growing_j = 1$ **then**
          $Grow.active(j)$
       else if $Kind = Latent$ **and** $Age^{-j} \in (\text{grow-start}, \text{grow-end}]$
       and $Growing_j^{-j} = 1$ **then**
          $Grow.passive(j)$

14.4 Germ growth

A latent cell turns immediately into a germ cell with $Age = 0$, $Addr = Q/2$ (which we can assume to be an integer). A germ is a domain of germ cells; its goal is to grow into 5 colonies, and then turn the middle one into a big cell. It tries to grow left and right, to addresses

$$e'_{-1} = -2Q \text{ and } e'_1 = 3Q - 1 \tag{14.3}$$

respectively, with a notation analogous to (14.2). A growing germ cannot intrude into an extended colony, as the cells of the latter are stronger, and a growing colony can destroy germ cells in its way. Also, the germ growth rule arbitrates between germs that are in the way of each others' growth. See Algorithm 14.16.

    The germ work period in Algorithm 14.11 consists of a part $Germ\text{-}grow$ when it tries to grow, followed by a part when it will decay if it did not reach its growth goal. So the edge of a germ cell is *protected* if it is pointing away from the cell with address $Q/2$, and has either age $< \text{grow-end}$ or reached address e'_j

in one of the directions j. The work period ends with the procedure `Lift`, to be described in Section 19.4, that lifts the computing functionality to the new level of simulation by adding a new level of error correction to the payload. The total number U of age-increasing steps of a germ work period is the same as that of a colony work period.

For convenience of notation, in what follows properties and fields related to germs will be marked with "G-".

Algorithm 14.11: A germ work period

> `Germ-grow`;
> Shrink;
> `Lift`

The growth of the germ happens in *stages*. The field *Stage-start* will show the start age of the current stage, lasting until

$$\textit{Stage-end} = \textit{Stage-start} + R \cdot \textit{G-size}, \tag{14.4}$$

where the field *G-size* shows the size of the germ at that time, and the constant R was introduced in (11.4). When $\textit{Age} = \textit{Stage-end}$ then cells will set *Stage-start* $\leftarrow$ *Stage-end* and *Stage-end* according to (14.4). Each stage will be divided into two parts: a *computation* part and a *growth* part. The growth part starts at age $\textit{Stage-start} + 0.1R \cdot \textit{G-size}$, and lasts until the end of the stage. Let

$$\textit{G-growing}(n) \in \{0, 1\}$$

be 1 if n is in the growing part of one of the stages, and 0 otherwise. During the computation part, the germ's cells will be informed of the addresses of the of the germ's edges by the rule `Pass-germ-size` of Algorithm 14.12. At the end of this part, a single step

$$\textit{G-size} \leftarrow \textit{G-edge}_1 - \textit{G-edge}_{-1}$$

records the germ size in each cell of the germ. We don't want a germ to grow much on one side if its growth is blocked (for whatever reason) on the other side. To achieve this, at the beginning of the work period, the central cell computes

$$\textit{Grow-dir} \leftarrow \arg\min\{\textit{G-edge}_{-1}, \textit{G-edge}_1\},$$

which, in case $G\text{-}edge_{-1} = G\text{-}edge_1$ chooses 1. This value will also be propagated by `Pass-germ-size`, and growth will only be attempted in direction $Grow\text{-}dir$. Also, as seen in the rule `Germ-grow.active` of Algorithm 14.15, the growing side of the germ will be allowed to grow only to a distance $G\text{-}size$ from its center.

Algorithm 14.12: rule `Pass-germ-size`

> **if** $Kind = Germ$ **then**
>> **if not** $G\text{-}growing(Age)$ **then**
>>> **cfor** $j \in \{-1, 1\}$ **do**
>>>> **if** $Edge_j = 0$ **then** $G\text{-}edge_j \leftarrow_1 G\text{-}edge_j^j$
>>>>
>>>> **else** $G\text{-}edge_j \leftarrow Addr$
>>>>
>>>> **if** $\text{sign}(Addr - Q/2) = j$ **then** $Grow\text{-}dir \leftarrow Grow\text{-}dir^{-j}$

The growth of germs over other germs has some color restrictions, which we will express with the help of the relation $Fit'_j(c, d)$. In case of the variant 1) of Definition 13.4, only germs of the same color are not prohibited from overriding each other, so simply $Fit'_j(c, d) = Fit_j(c, d)$. In case of the variant 2), also germs with color 0 are not prohibited to override fitting germs. Formally,

$$Fit'_j(c, d) = 1 \iff Fit_j(c, d) \wedge (c = d \vee d = 0).$$

A germ can grow (subject to the above constraints) over another, if it is significantly larger. By "significantly larger", we mean larger by a factor

$$\rho = 5/4. \tag{14.5}$$

If they are close in size, that is they differ by a factor $\leq \rho$ then the random choice described below will arbitrate between them. However, let us see that even the smaller one, if the decision favors it, will grow to become significantly larger than the larger one. So suppose that germ G_1 has size n, germ G_2 has size ρn, and germ G_1 is allowed to grow. Its growth can be stopped by another germ G'_1 growing in the opposite direction. However, one of these will cover at least half of G_2. So the germ that grew will have size at least $n(\rho/2 + 1) > \rho^2 n$.

The randomized decision works as follows. Both germs look at a field

$$Dominant \in \{0, 1\},$$

which will be set randomly by the cell with address $Q/2$ at each stage start, and then propagated by the rule `Pass-dominant` of Algorithm 14.13. Between

germs of nearly the same size, the dominant one will prevail. After the computation part of the stage, the values of *G-size* and *Dominant* are supposed to be constant throughout the germ: this is part of the requirement for two neighbor cells to be siblings. Formally, the relation $Fit'_j(c, d)$ and

$$G\text{-}less_j(a, b)$$

will be used by the rule `Germ-grow.passive` of Algorithm 14.16. The meaning of $G\text{-}less_j(a, b)$ is that the germ cell with state b is in direction j from the germ cell with state a, and will be able to grow over it or over the space that a wants to grow over. Here are the rules to determine $G\text{-}less_j(a, b)$. The germ of b can definitely grow over the one of a if both are in their growing ages, and it is it is either significantly larger or has reached the end of its growth on the other side. A remaining case is when both germs reached their growth end on the other side: then the germ of b can grow over the one of a if it is dominant while a is not. The other remaining case is when neither germ reached its growth end on the other side, and their sizes are also comparable: then again the germ of b can grow over the one of a if it is dominant while a is not. Formally, here is what determines whether $G\text{-}less_j(a, b)$ holds:

1. We always need $\textit{Kind}(b) = Germ$, $G\text{-}growing(b.\textit{Age}) = 1$.

2. $a.\textit{Kind} < Germ$ is enough, otherwise $a.\textit{Kind} = Germ$ and $G\text{-}growing(a.\textit{Age}) = 1$ is necessary.

3. If $a.\textit{G-edge}_{-j} \neq e'_{-j}$ then $b.\textit{G-edge}_j = e'_j$ or $\rho a.\textit{G-size} < b.\textit{G-size}$ is enough.

4. $a.\textit{Dominant} = 0$ and $b.\textit{Dominant} = 1$ is enough if either $a.\textit{G-edge}_{-j} = e'_{-j}$ and $b.\textit{G-edge}_j = e'_j$ or $a.\textit{G-edge}_{-j} \neq e'_{-j}$ and $a.\textit{G-size} \leq \rho b.\textit{G-size}$.

Germ growth is governed by rule `Germ-grow` of Algorithm 14.14. The ages in which the edge of a germ can expand will be restricted only to part of the growth period, to make sure that it can finish the growth it started:

$$G\text{-}growing.active(\textit{Age}, \textit{Addr}) \iff G\text{-}growing(\textit{Age})$$
$$\wedge\, \textit{Age} < \textit{Stage-end} - 0.1R\textit{G-size} + 5\lambda\tau_2|\textit{Addr} - Q/2|.$$

Similarly, the ages in which the edge of a germ can be killed by the expansion of another germ will be restricted only to part of the growth period, to make sure that the whole germ can be killed before the end:

$$G\text{-}growing.passive(\textit{Age}, \textit{Addr}) \iff G\text{-}growing(\textit{Age})$$
$$\wedge\, \textit{Age} < \textit{Stage-end} - 5\lambda\tau_2|\textit{Addr} - Q/2|.$$

Algorithm 14.13: rule `Pass-dominant`

    if *Kind* $= Germ$ **then**

        if *Addr* $= Q/2$ **and** *Age* $=$ *Stage-start* **then** *Dominant* $=$ *Rand*

        else let $j = \text{sign}(Addr - Q/2)$

        Dominant $\leftarrow_1$ *Dominant*$^{-j}$

Algorithm 14.14: rule `Germ-grow`

    if *Kind* $= Germ$ **then**

        let $j \leftarrow \text{sign}(Addr - Q/2 - 0.1)$

        `Germ-grow.active`(j)

    if *Kind* $\leq Germ$ **then**

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

            if *Kind*$^j_{-j} = Germ$ **then** `Germ-grow.passive`(j)

Algorithm 14.15: rule `Germ-grow.active`(j)

    if *G-growing.active*$(Age, Addr)$ **and** *Age* $<$ grow-end

    and $e'_{-1} < Addr < e'_1$ **and** $j =$ *Grow-dir* **and** $|Addr - Q/2| <$ *G-size*

    then *Kind*$_j \leftarrow Germ$

    else *Kind*$_j \leftarrow Latent$

Algorithm 14.16: rule `Germ-grow.passive`(j)

    if *Kind* $= Latent$ **or** (*Kind* $= Germ$

    and *G-growing.passive*$(Age, Addr)$ **and** $Fit'_j(Color, Color^j)$

    and *G-less*$_j(\mathbf{x}, \mathbf{x}^j)$ **and** *G-less*$_j(\mathbf{x}^{-j}, \mathbf{x}^j)$) **then**

        `Adapt`(j)

14.5 Healing rules

Healing involves several rules. Recall frozen cells in Definition 13.9, and that a frozen cell's age will not be advanced in the regular way (seen in the rule 13.3, `March`). The default value of *Frozen* is 0; nonzero values of *Frozen* will point to some "defects". This field is intended to show that in the direction indicated by its sign, there is a time consistency at a distance indicated by its absolute value $|Frozen| \leq \Delta$ (the latter constant is defined in (13.4)). So if *Frozen* $= 1$ then the cell has on the right a time-inconsistent neighbor that is not a protected edge; if it is 2 then its right neighbor has this property. The value of *Frozen* will be propagated by part (2) of `Freeze` of Algorithm 14.17. Suppose that

$-\Delta < \textit{Frozen}(x - B) < 0$ and x is not exposed on the right: then we will want to propagate the signal to the right by setting $\textit{Frozen}(x) \leftarrow \textit{Frozen}(x - B) - 1$.x If at the same time $x + B$ is a sibling with $0 < \textit{Frozen}(x + B) < \Delta$, then $\textit{Frozen}(x) \leftarrow \textit{Frozen}(x + B) + 1$ would also be suggested. In such cases we will break the tie and propagate to the right. The freezing process is typically started by part (1). If our cell is exposed to the right then we set $\textit{Frozen}(x) = 1$.

The last **else** of the rule *gradually* unfreezes the area after the inconsistency was eliminated. As expected, the field *Frozen* is private to the rule `Freeze`:

Condition 14.10 (Freeze) Only the rule `Freeze` can change the field *Frozen* (without killing the cell). ⌋

The typical result of repeated application of the `Freeze` rule is that values of the field *Frozen* in consecutive cells in the place where damage occurred will be like

$$0, \ldots, 0, 5, 4, 3, 2, 1, *, *, -1, -2, -3, -4, -5, 0, \ldots, 0, \qquad (14.6)$$

(using $\Delta = 5$). Here the $*$'s show latent cells. All (weakly) exposed edges are confined between the 1 and -1 in this picture. One of the sides of this picture may be cut short by a protected edge.

Algorithm 14.17: rule `Freeze`

> **cond**
>> **cfor** $j \in \{-1, 1\}$ **do**
>
> (1)   **if** $\textit{Xposed}_j$ **and** the neighbor in direction j is not
>>> time-consistent   **then**
>>>> $\textit{Frozen} \leftarrow_1 j$
>>
>> **cfor** $j \in \{-1, 1\}$ **do**
>
> (2)   **if** $1 \leq |\textit{Frozen}^j| < \Delta$
>>> **and** $(j = -1$ **or not** $1 \leq |\textit{Frozen}^{-j}| < \Delta)$ **then**
>>>> $\textit{Frozen} \leftarrow_1 \textit{Frozen}^j + j$
>>
>> **else** $\textit{Frozen} \leftarrow_{p_1} 0$

Recall the notion of directly affected cells in Definition 10.15. The number of consecutive adjacent cells directly affected by the damage rectangle is at most Δ, defined in (13.4). For the moment, let us call the smallest interval encompassing these sites the *corruption*. The healing rule will be able to repair a corrupted interval of size at most Δ arising in some domain. A typical case is when the corruption occurs inside a domain which itself is part of a colony. In thinking of the healing rule, let us assume that other rules (namely `Freeze`, `Purge` and `Create`) have achieved that the corruption is filled with cells aligned with its boundaries. If our model was in discrete-time

then the correct values of *Addr*, *Age* and the locally maintained fields could just be inferred from either edge of the corruption and copied into its cells. But since the model is in continous time, the task is more complex. First, the `Freeze` rule will freeze the *Age* values in some of the cells surrounding the gap. (Otherwise, if the *Age* in one of these cells speeds ahead, it may become impossible to bridge the gap with continously varying *Age* values.) Then, it takes some coordination to recreate a continous sequence of *Age* values.

Remark 14.11 *It would be convenient here if our robust medium had range $r > 4$, so cells on both sides of the corruption can see each other. But introducing "action at a distance" brings its own headaches.* ⌟

The healing rule itself will carry out only a single step of the reconstruction, but its repeated applications will close the gap. As a further complexity, the corrupted cells can be alive, and therefore it might not even be clear which cell to correct. Suppose, for example, that values of *Age* in adjacent cells are $0, 0, 2, 2$. Both $0, 1, 2, 2$ and $0, 0, 1, 2$ are possible corrections: the healing rule will choose one.

Rule `Heal` in algorithm 14.18 consists of several rules applied simultaneously.

Algorithm 14.18: rule `Heal`

   `Heal-revive ∥ Heal-sync`

Rule 14.19, `Heal-revive`, will revive a latent cell x, making it a weak sibling to one of its neighbors. Rule `Heal-sync` in Algorithm 14.23 will try to adjust the *Age* variables. In rule `Heal-revive`, assume that x is latent, and $x - B$ is alive. Then the rule may try to apply the rule `Adapt(−1)` to x, adapting it to its left neighbor. If there is a similar demand on the right-hand side, then if one of the adaptations creates a stronger cell, that one is chosen, otherwise the left side is preferred. The field *Healing* will count the cells revived during healing (by numbering them), limiting this way the size of revived segments to heal-span. It actually may need to revive a segment of size 2Δ with (Δ defined in (13.4)): if cells with *Addr* $\in \{\Delta, \dots, 2\Delta - 1\}$ are changed by damage then the purge rule may kill these cells as well as the now isolated segment with *Addr* $\in \{0, \dots, \Delta - 1\}$.

Definition 14.12 (Healing field) We have a field called

$$Healing \in \{0, 1, \dots, \text{heal-span}\}.$$

Its default value is 0. A cell with *Healing* > 0 will be called a *healing cell*. The field

$$Decaying \in \{0, 1\}$$

will limit the healing process in time. ⌋

The rule `Heal-revive` is the combination of several simpler rules. The rule `Heal-passive` adapts a cell to an exposed member neighbor (making it thus also a member), unless the neighbor is decaying or is doomed and old. Its part (1) requires the healing to proceed always away from a colony center (this restriction is only for proof convenience). The end-healing will allow to kill intruder cells at the end of a colony in order to heal the end; these intruder cells would not be necessarily isolated, so might not be eliminated by the `Purge` rule.

Condition 14.13 The rule `Heal-revive` is the only one changing the value of the field *Healing*. ⌋

Algorithm 14.19: sub-rule `Heal-revive`

> **if** *Kind* $= Latent$ **then** `Heal.passive`
> **else**
>> `End-heal`
>> `Control-decay`

Algorithm 14.20: sub-rule `Heal.passive`, delay p_1

> **cfor** $j \in \{-1, 1\}$ **do**
>> **if** *Healing*^{j} $<$ heal-span **and** $Xposed^j_{-j}$
>> **and** *Decaying*$^j = 0$ **and** *Kind*$^j = Member$
>> **and not** (*Doomed*^{j} **and** *Age*$^j > U - 2Q$)
(1) >> **and** direction j is towards the colony center of $\mathbf{x}^j$
>> **and** *Non-germ-less*$_j$ **then**
>>> `Adapt`(j)
>>> *Healing* $\leftarrow$ *Healing*$^j + 1$

Here is the "story" interpreting these rules (and helping to follow their analysis in later proofs). Suppose healing proceeds, say, towards the left. An exposed cell sets *Decaying* $= 1$ *slowly* in part (1) of the rule `Control-decay`, Algorithm 14.22, so while *Decaying* is not set it still allows

Algorithm 14.21: sub-rule `End-heal`

> **cfor** $j \in \{-1, 1\}$ **do**
> > **if** $Xposed_j$ **and** $0 < |e_j - (Addr \bmod U))| \le$ heal-span
> > **and** $Kind = Member$ **then**
> > > $Kind_j \leftarrow Member$

the rule `Heal.passive` in Algorithm 14.20 to proceed with healing. If a healing arm created by it reaches maximum length and does not close a gap then its exposed end eventually sets $Decaying = 1$, and the rule `Decay` in Algorithm 15.1 will kill it. In rule `Control-decay` of Algorithm 14.22, as soon as a cell gets a neighbor with a larger $Healing$ value, it sets $Decaying = 1$ *fast* in part (2). Thus if the neighbor with higher $Healing$ value gets killed due to unsuccessful healing, it will not be revived again by rule `Heal.passive` of Algorithm 14.20. This way eventually the whole unsuccessful healing arm will be killed. Its first neighbor with $Healing = 0$ will also be killed; so an unsuccessful healing eventually widens the gap it tried to heal. Rule `Control-decay` in Algorithm 14.22 also sets the fields $Healing$ and $Decaying$ to their default value 0 when their role has ended.

Algorithm 14.22: rule `Control-decay`

(1) **if** $\exists j \; Xposed_j$ **then** $Decaying \leftarrow_{p_2} 1$

(2) **else if** $\exists j \; Healing^j > Healing$ **then** $Decaying \leftarrow_1 1$
> **else**
> > $Healing \leftarrow_1 0$
> > $Decaying \leftarrow_1 0$

Rule `Heal-sync` in Algorithm 14.23 will readjust the age of some frozen cells, to eliminate a discontinuity. To do this, it will try to bring Age closer to the one of the neighbor to which the $Frozen$ field points. The conditions $Frozen/j > 0$ and $Frozen^{-j} \cdot Frozen > 0$ say that both $Frozen(x)$ and $Frozen(x - jB)$ point to the same direction j. Condition $Depth_{-j} > 1$ requires the neighbor in direction $(-j)$ to be a sibling. The condition $\Delta(j) \cdot \Delta(-j) \ge 0$ does not allow an age change that would break this sibling relation. Also, because of $Frozen^{-j} \cdot Frozen > 0$, if changes happen simultaneously in both x and $x - jB$ then they happen in the same direction, still not breaking the sibling relation.

Condition 14.14 (Age decrease) The only rule in which the Age field may decrease is `Heal-sync`. ⌟

Algorithm 14.23: sub-rule `Heal-sync`

> **pfor** $j \in \{-1, 1\}$ **do let** $\Delta(j) \leftarrow Age^j - Age$ amod U
> **cfor** $j \in \{-1, 1\}$ **do**
> > **if** $Frozen/j > 0$ **and** $Frozen^{-j} \cdot Frozen > 0$ **and** $Depth_{-j} > 1$
> > **and** $\Delta(j) \neq 0$ **and** $\Delta(j) \cdot \Delta(-j) \geq 0$ **then**
> > > $Age \leftarrow Age + \text{sign}(\Delta(j)) \bmod U$

Example 14.15 The typical application of `Heal-sync` is, say, when in a domain of cells with age values $1, 1, 1, 2, 3, 4, 5, 5$, the damage changes the values of three to

$$1, 1, 1, 0, 7, 6, 5, 5.$$

Now the cells with age values 0 and 7 become exposed towards each other, and the *Frozen* variable develops the values $4, 3, 2, 1, -1, -2, -3, -4$. After this, `Heal-sync` will gradually bring the ages closer to each other. Here is a possible synchronization history (time going downwards):

$$
\begin{array}{cccccccc}
1 & 1 & 1 & 0 & 7 & 6 & 5 & 5 \\
1 & 1 & 0 & 0 & 6 & 6 & 5 & 5 \\
1 & 1 & 0 & 1 & 6 & 6 & 5 & 5 \\
1 & 1 & 0 & 1 & 5 & 6 & 5 & 5 \\
1 & 1 & 0 & 1 & 5 & 5 & 5 & 5 \\
1 & 1 & 0 & 1 & 4 & 5 & 5 & 5 \\
1 & 1 & 1 & 1 & 4 & 4 & 5 & 5 \\
1 & 1 & 1 & 2 & 3 & 4 & 5 & 5 \\
\end{array}
$$

As in the second line here, the synchronization may not start in the best direction, but since the adaptation is towards the fault in the middle, eventually the fault will disappear.

It is not hard to see that in general, if the domain to synchronize has width Δ then the synchronization will be finished within 2Δ steps. ⌙

Let us summarize the possible uses of animation and killing.

Condition 14.16 (Animation, killing)   a) Only the rule `Die`(p) can kill a cell. It is invoked always with $p \geq p_1$.

  b) Only the following rules can invoke `Die`: `Adapt`, `Purge`, `Decay`.

  c) Only the rule `Adapt` can erase a cell (make it vacant).

d) A protected edge can be killed only if a neighbor sets $Kind_j \neq Latent$ in its application of a growth rule or the end healing rule.

e) Only the rule **Create** can change $Creating_j$.

f) Only the rule **Adapt** can create a non-latent cell, invoked only by **Heal**.**passive**, **Grow**.**passive** and **Germ-grow**.**passive** (with the corresponding delay). It is also the only rule by which a potential creator can arise in the sense of Definition 10.12.

⌟

An exposed edge is almost always a sign of past damage, the exception being when at the end of a stage, a large group of cells needs to be killed off:

Lemma 14.17 (Exposing) *A rule exposes an edge only in the following cases:*

1) *Member cell whose neighbor is doomed, and dies at $Age = 0$, that is at a cut, as in Definition 12.8.*

2) *Channel cell, $Age =$ compute-start.*

3) *Germ cell or growth, non-end-cell. $Age =$ grow-end, or when a germ cell is killed by **Adapt** invoked in **Germ-grow**.**passive**.*

Proof. Direct consequence of the definition of siblings and exposed edges and Conditions 13.10 (Address and Age), 14.16 (Killing). □

14.6 Continuity

Our terminology turns out to be "incestuous": a child cell can only be created if it also becomes a sibling. Recall the interval $I_1(x)$ in Definition 10.14.

Lemma 14.18 (Parent) *Suppose that $I_1(x)$ is damage-free during $(u - T^\bullet, u]$ (that is x is not affected via neighbors during this time, as in Definition 10.15), and the non-germ cell x becomes animated at time u. Then there is a $j \in \{-1, 1\}$ and $t' \in (u - 3T^\bullet, u - T_\bullet/2]$ such that the following holds:*

a) *x gets animated by parent $x + jB$, whose state at that time makes it a sibling of (x, u).*

b) *If $I_0(x) \cup I_0(x + jB)$ remains damage-free during $(u, u + T^\bullet]$ then x and $x + jB$ remain siblings during that time.*

Proof. We have an application of **Adapt** of Algorithm 14.6 from some $y = x + jB$. So at the observation time t' corresponding to the switch (x, u), we have the situation described by a). To prove b): according to Condition 14.16, the applied rule was either **Heal** or **Grow**, creating a sibling with Age equal to that of the parent. Dying can only happen by decay or purge, which takes

time at least $p_0 T_\bullet \geq 2T^\bullet$. The age of the child is made equal to the age of the mother. In case of growth, the mother has time for at most change of age (with delay p_1), and this will not break the sibling relation within time $2T^\bullet$. The healing case is trickier since the rule **Heal-sync** in Algorithm 14.23 can also decrease *Age*. But given that x was just animated then if, say, it is created from the left, then the *Frozen* variables must point to the right. Therefore the rule will not cause age change in the left neighbor as the ages are made equal. $\qquad\square$

Normally, siblings remain siblings:

Lemma 14.19 (Glue) *Suppose that the adjacent cells $x, x + B$ are siblings just before the time t_0, at which x breaks the sibling relation. Suppose also that both x and $x + B$ are unaffected by damage even via neighbors during $(t_0 - T^\bullet, t_0]$. Then we have the following possibilities:*

1) A cut as in Definition 12.8.

2) Cell $x - B$ was not a sibling of x at the last observation time of x, and the switch kills x by **Decay**, **Purge** *or* **Adapt**.

The statement also holds if we exchange left for right and $x + B$ for $x - B$.

Proof. 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 13.10 (Address and Age), 14.16, and the healing rule. We will show that if neither of the possibilities listed in the statement of the lemma holds then the cells remain siblings.

1. Suppose that x or $x+B$ were animated at some time in $(t_0 - T^\bullet, t_0]$; without loss of generality, suppose that the cell was $x + B$.

   Then $x + B$ will be without an age change or dying for at least $p_0 T_\bullet$ time units which is longer than the whole period under consideration, as (12.6) implies $p_0 \geq 5\lambda$. 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 **Adapt** implies that $x + B$ is not a germ unless x is a parent. If x is a parent of $x + B$, then part b) of Lemma 14.18 (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 13.10 (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 was exposed to the right (since $x+B$ was latent), and therefore the rule 14.17 (*Freeze*) froze it, keeping x and $x + B$ siblings.

2. Suppose now that both cells have been live during $(t_0 - T^\bullet, t_0]$.

If x changed 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. Suppose that $x + B$ became a sibling of x during this time. Then it was not a sibling before, and the *Freeze* rule must have frozen x, so x would not change its age so soon. 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. This is guaranteed even within the rule *Heal-sync*, which may change age in both directions.

$\square$

15   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. This serves as a preparation to the Attribution Lemma of the next section.

15.1   Paths

We will track the continuity of live areas in space-time via the notion of a path.

Definition 15.1 (Links) For times $t < u$, assume cell x is live at t and its body is damage-free during $[t, u]$, and there are no switching times in the open interval (t, u). Then we say that point (x, t) is connected by a *vertical link* to point (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 at time t or $(t-)$ such that $[x - B, x + 2B)$ is damage-free during $(t - T^\bullet, t]$, then the points (x, t), $(x + B, t)$ are said to be connected by a *horizontal link* . If point (y, t') is, according to case a) of Lemma 14.18 (Parent) a parent of point (x, u), we will say that (y, t') is connected by a *parental* (maternal or paternal) link to point (x, u). 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.) ⌟

By Lemma 14.18 (Parent), the parental link can be replaced by a vertical connection and a horizontal connection. The horizontal connection is to a sibling that is a successor to the parent, and the vertical ones continue back in time towards the parent.

Definition 15.2 (Path) 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 path with only steep links is *steep*, or *slow*. A *backward path* is the reversed reading of a forward path, backwards in time. The adjective "forward" or "backward" will be omitted, when it is obvious from the context. For a path $P = (x_0, t_0), \ldots, (x_n, t_n)$ and $t \in [t_0, t_n]$, let

$$P(t)$$

be x_i for the smallest i with $t \in [t_i, t_{i+1}]$. ⌟

A point (x_i, t_i) on a path can actually be dead, if it has just died: indeed, it can be connected for example to a point (x_{i+1}, t_{i+1}) by a horizontal link with $t_i = t_{i+1}$ such that x_i, x_{i+1} are siblings at time (t_i-). The following lemma says that if two paths cross then they have a common point.

Lemma 15.3 (Crossing) *Let* $(x_1, s_1), \ldots, (x_m, s_m)$ *and* $(y_1, t_1), \ldots, (y_n, t_n)$ *be 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. Let us call the point (x_i, s_i) whose existence is asserted, the *crossing point*. Let us replace parental links with horizontal and vertical connections as described in the remark after Definition 15.1. Now the two paths cannot jump over each other. Indeed, at the time of the crossing, the cells involved have to be at a distance B. At this crossing, one of the links must be horizontal; suppose it is (x_i, x_{i+1}). Then either (x_{i+1}, s_i) is on a vertical link between some t_j and t_{j+1} or there is also some horizontal a link $(y_j, y_{j+1}) = (x_{i+1}, x_i)$ at time $s_i = t_j$. In both cases we can choose the crossing point $x_{i+1} = y_j$ at time s_i. □

According to the Parent Lemma, a steep path can be continued backwards in time until it hits some island (as in Definition 10.3). Moreover, occasionally we have a choice between two parents to continue to. The lemma below says that any path started backwards not too soon after damage (the time defined as purge-t in (15.2)) can be diverted and continued back past any island.

Definition 15.4 (Traceability) A cell (x, t) will be called *traceable* if $I_0(x)$ is damage-free during $(t - 4\,\mathsf{purge\text{-}t}, t]$. ⌟

Lemma 15.5 (Skirting) *Let $[a_0, a_1) \times (u_0, u_1]$ denote the (only) island in the area under consideration. Consider a traceable live point (x_0, t_0). There is a path going backwards from (x_0, t_0) and ending either in time u_0 or in a birth (in the sense of Rule Birth of Algorithm 14.4). It has at most $\Delta + 1$ non-steep links (with Δ as in (13.4)) which, with one possible exception, form a series of up to Δ consecutive horizontal links, and can only cross a colony boundary in the inward direction, and only if $Age <$ grow-end $+ \Delta$ on all cells involved.*

Proof. Let us start constructing a steep path $c_0, \dots, c_n$ with

$$c_i = (x_i, t_i)$$

backwards in time from (x_0, t_0), consisting of either vertical links, or parental links if the conditions of Lemma 14.18 (Parent) are applicable. If we get to time u_0 then we are done. Otherwise let us stop just before going below $u_1 + T^\bullet$, with c_k being the last element. Then the body of x_k intersects $[a_0 - 1.1B, a_1 + 1.1B)$: indeed, otherwise we could continue the path either by a vertical or by a parental link. The vertical link would be shorter than $T^\bullet$, and the parental link would lead to a damage-free cell, so either of them would be allowed.

1. There is an $i \le k$ such that (x_i, t_i) can be connected by horizontal links that do not cross a colony boundary in the outward direction, to a cell y unaffected by the damage even via neighbors at any time (in the rectangle under consideration).

   Proof. Let us go back on the path for $k, k-1, \dots$ until the first i (counting from k) such that either $t_i > u_1 + (\Delta + 1)\tau_0$, or c_i is a parent of c_{i-1}; let us call it i_1.

   Suppose first $t_{i_1} > u_1 + (\Delta + 1)\tau_0$. Then there is some $i_1 < i \le k$ such that (x_i, t_i) can be connected by horizontal links to a directly unaffected cell y. Indeed, if there is not, then $x_i = x_k$, $i = k, k-1, \dots, i_1$, since we have not encountered a parent before. The part of the domain containing x_k that consists of cells directly affected by damage has size $\le \Delta$. If we cannot pass away from it via siblings within the same colony or towards the originating colony then at least one end will be exposed and the depth will also be bounded by Δ. This allows the *Purge* rule to gradually eliminate this part, including x_k, leading to a contradiction. The time this takes is at most $\Delta T^\bullet$ to propagate the depth information, followed by $\Delta \tau_0$ to apply the steps of *Purge* until x_k is reached, for a total time of $\Delta \tau_0 + \Delta T^\bullet$. This shows $t_0 \le u_1 + (\Delta + 1)\tau_0$ since $\Delta < p_0$. By the definition of purge-t this contradicts the assumption of the traceability of (x_0, t_0). If the only way

to get by horizontal links to a directly unaffected cell is to cross a colony boundary in the inward direction with $Age \geq \mathsf{grow\text{-}end} + \Delta$ then there is an exposed cell in the opposite direction, at distance $\leq \Delta B$, so `Purge` will act again in the same way.

Suppose now that c_{i_1} is a parent of c_{i_1-1}. Let us then go back on the path for $i = i_1, i_1 - 1, \dots$ until the first i (counting from i_1) such that $t_i > t_{i_1-1} + (\Delta + 1)\tau_0$. There is such an i, and we have $x_i = x_{i_1-1}$ for all i, since the newborn x_{i_1-1} cannot be a parent sooner. Indeed, Condition 14.16 says that animation happens only via healing or growth, and even the faster healing step has a delay of p_1. By the same reasoning as above, there is some $i_1 < i \leq k$ such that (x_i, t_i) can be connected by a sequence of at most Δ horizontal links not crossing a colony boundary in the outward direction, to a directly unaffected cell y.

2. There is a steep path backwards in time from the point (y, t_i) computed above, going only either on vertical or parental links, and ending below time u_0.

   Proof. Let us construct such a path. If it contains at most one parental link, then the body of cells it reaches still is damage-free, hence the path can be continued. On the other hand, it cannot contain two parental links, since a newborn x_{i_1-1} cannot be a parent soon.

   $\square$

Definition 15.6 (Trace-back path) A backward path is called a *trace-back path* if every link in it is chosen to be a vertical or a parental link if this is possible, with horizontal links only as needed in Lemma 15.5 (Skirting).   ⌟

Definition 15.7 (Age progress) Let $P = (c_0, \dots, c_n)$ be a forward path. The *age progress* of P is defined as

$$age\text{-}progress(P) = \sum_{i=0}^{n-1} (Age(c_{i+1}) - Age(c_i) \operatorname{amod} U).$$

⌟

The following lemma upper-bounds the number of colonies a trace-back path can cross and also its age progress.

Lemma 15.8 *Suppose that a trace-back path P with n links has time projection $d < T_\bullet^*/4$. Then the following holds.*

a) *P passes through at most 3 colonies, for a total space projection of $< 2.1QB$.*

b) $d \geq (n - \Delta - 1)T_\bullet/2$. *If the path has no parental links then the factor* $1/2$ *can be omitted.*

c) *The age progress of* P *is at most* $n/p_1 + \Delta + 2$.

Proof. Let us show that P can pass through at most 3 colonies. It moves horizontally only along parental and horizontal links. The latter only occur in connection with the application of Lemma 15.5 (Skirting), with at most Δ horizontal links per island. Given the time bound on the whole path, as long as it does not cross more than three colonies, at most one island will appear, hence we do not have to count with more than one series of $\leq \Delta$ horizontal links.

Parental links can only occur in connection with healing and growth. Healing moves the forward path towards the edge of a colony as shown in rule *Heal.passive* of Algorithm 14.20, only growth can move it towards the center. In order to move towards the center of another colony, another growth cycle must start (except in case of germ growth, which can span three colonies). But definition (12.3) implies that within the time bound $T_\bullet^*/4$ the path cannot reach from the end of one growth to the beginning time of another.

Let us prove the time projection lower bound. There are at most Δ horizontal links and 1 non-steep link, arising from the island. The other at least $n - \Delta - 1$ links are steep. A steep link is either vertical, in which case it has size $\geq T_\bullet$, or parental, in which case it has size $\geq T_\bullet/2$.

Let us estimate now the age progress. By the definition of the animations in healing and growth, age does not increase along parental links. Along horizontal links the age can increase by 1, and it can also increase immediately after the last horizontal link. It can also increase by 1 at the first link (in time). But during the remaining vertical links, age increase occurs no sooner than every p_1 steps, hence the total age increase is at most $\Delta + 2 + n/p_1$. □

15.2 Running gaps

The rule *Decay* attempts to widen any gap that was not closed in reasonable time. It erases all the cells created by *Heal-revive*, as well as one more cell, since they were marked by *Decaying*. The delay p_2 of decay gives a chance for the healing process to complete. The difference between killing by *Purge* and killing by *Decay* is that *Purge* kills fast, but its reach, in case of non-doomed member cells, is only local: healing is intended after it. On the other hand, *Decay* slowly eliminates the remainders of a colony if healing fails to close a gap within a certain time limit.

Algorithm 15.1: rule `Decay`

    if *Decaying* $= 1$ **and** $\exists\, j \in \{-1, 1\}$ *Xposed$_j$* **then** `Die`(p_2)

Gaps and gap paths are a tool to show that the decay rule indeed destroys incomplete colonies. Their definition is complicated by the consideration of germs, needed for self-organization. Here are some constants; for readability, we omit the notation $\lfloor \cdot \rfloor$ for integer part. Recall the definition $\tau_i = (p_i + 1)T^\bullet$ in (12.4), Δ in (13.4) and heal-span in (12.5). Let

$$\text{island-size} = 2.5,$$

$$\text{gap-lb} = 4\,\text{heal-span} + 2\,\text{island-size} + 21, \tag{15.1}$$

$$\text{purge-t} = (\Delta + 1)\tau_0, \tag{15.2}$$

$$\text{split-n} = (\text{heal-span} + 1)(2\,\text{gap-lb} + 1), \tag{15.3}$$

$$\text{split-t} = \text{split-n}\,\tau_2. \tag{15.4}$$

Definition 15.9 (Gaps) Consider an open interval $G = (l, r)$ at some time t, where $0 < r - l$ is divisible by B. We say that G is a *right gap* with *size* $|G| = r - l - B$ if every traceable cell in G can be space-consistent with r only if it is a germ cell. The *right age* of the gap G is is the upper bound of the age of these germ cells (it is 0 if the gap contains no germ cells). If G is contained in the colony of r then it is called an *interior gap*. A *right bad gap* is a gap of right-age

$$\lambda\,\text{split-n}\,(\text{heal-span}\,/2 + 1)$$

containing an interior gap of size $\geq \text{gap-lb}\,B$. If the gap itself is not interior then its right edge is required to be an exposed right outer cell. Left gaps are defined similarly. ⌟

We are interested in how a gap changes in time.

Definition 15.10 (Gap path) 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 $G(t_1) \cap G(t_2) \neq \emptyset$ for all t_1, t_2 with $|t_2 - t_1| \leq 3T^\bullet$. 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. ⌟

Let us make the following simple observation.

Lemma 15.11 *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 $G(t)$.*

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.

Lemma 15.12 (Running Gap) *Recall the definition of* gap-lb *in* (15.1). *Let* $P_1 = (x_0, v_0), \ldots, (x_n, v_n)$ *be a trace-back path (listed forwards), let* L, k *be positive integers with*

$$L < Q^2,$$
$$k < U - 3Lp_2\lambda.$$

Assume the following:

a) $y_0 < x_0$ *are in the same domain at time* v_0.

b) *The interval* $[y_0 - 2.1B, x_0 + 3.1B)$ *has been noise-free during* $(v_0 - T^\bullet, v_0]$.

c) y_0 *is an exposed left edge at time* v_0 *(see Section 13.3), and has been the right end of a bad gap of right-age* k *during* $(v_0 - T^\bullet, v_0]$. *Its age is not within* Δ *steps of any of the age values listed in Lemma 14.17 (Exposing), with* Δ *defined in* (13.4).

d) *If* y_0 *is a left outer cell then* P_1 *is in the same colony. In this case, let* Z *be the starting cell of the originating colony. Otherwise,* $Z = \infty$.

e) $v_n - v_0 \leq L\tau_2$.

Then during $(v_0, v_n]$, *a right gap path* $G(t) = (l(t), r(t))$ *can be defined, with* $r(v_0) = y_0$,

$$r(v_n) \geq Z \wedge \left(y_0 + B\left(\frac{|v_n - v_0 - \mathsf{purge\text{-}t}|^+}{3\tau_2} - 2\,\mathsf{heal\text{-}span} - \mathsf{island\text{-}size} - 11 \right) \right),$$
$$(15.5)$$

$|G(t)| > 0$, *with right-age* $< k + 3L\lambda p_2$.

The following corollary says (in contrapositive) that if path P_1 is long, then there is no large young gap next to its beginning (say, on the left).

$$\mathsf{destr\text{-}t} = 10\tau_2, \tag{15.6}$$
$$\mathsf{destr\text{-}s} = \mathsf{destr\text{-}t}\,/p_1 T_\bullet. \tag{15.7}$$

We will also use the following lower bound on Q:

$$Q > 3(2\,\mathsf{heal\text{-}span} + \mathsf{island\text{-}size} + 11) + \mathsf{purge\text{-}t}\,/\tau_2. \tag{15.8}$$

(x_n, y_n)

$l(t)$ $r(t)$ $P_1(t)$ time

(y_0, v_0) (x_0, v_0)
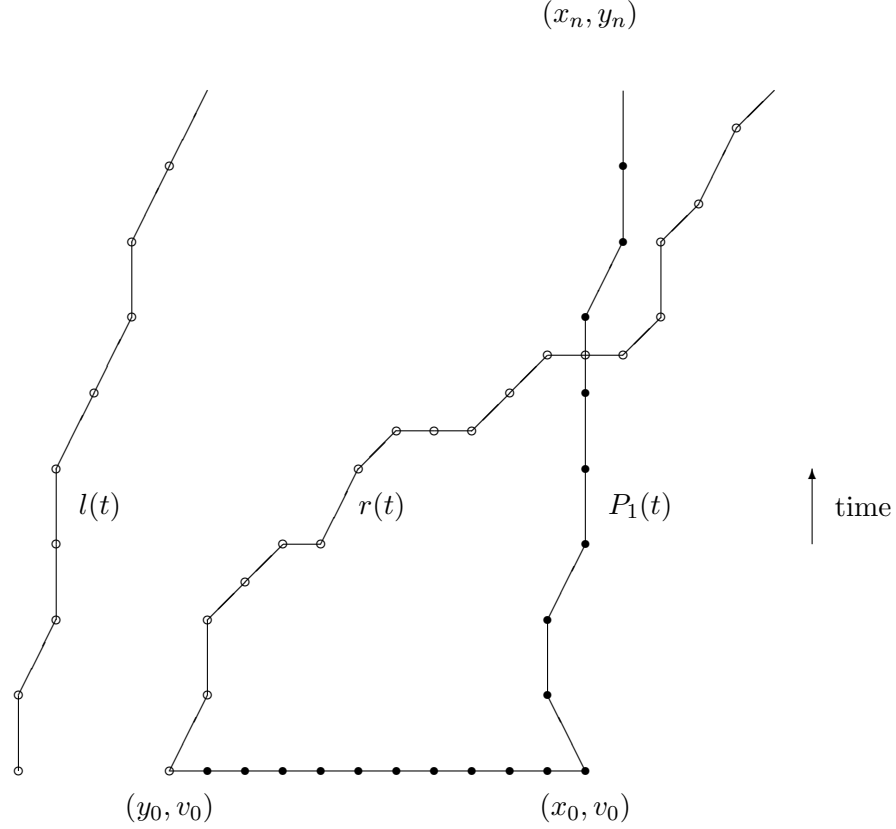
Figure 10: Running Gap Lemma

Corollary 15.13 (Running Gap) *Assume the conditions of the Running Gap Lemma, further that (x_n, v_n) is not a germ cell younger than $k + 6L\lambda p_2$. Then $v_n - v_0 < \mathsf{destr\text{-}t}\, Q$.*

Note that the statement is meaningless unless $L > 10Q$, since $v_n - v_0 < L\tau_2$ is a condition of the Running Gap lemma.

Proof. It is easy to see that the length of the path is at most $3L\lambda p_2$ (the proof is just like the proof of the lower bound on the length of path P_2 in the proof of the Running Gap Lemma below). Path P_1 starts to the right of the gap path $G(t)$. Since age varies by at most 1 per link along a path and since (x_n, v_n) is not a germ cell younger than $k + 6L\lambda p_2$, no cell on P_1 is a germ

cell younger than $k + 3L\lambda p_2$, while according to Lemma 15.12, all cells in $G(t)$ space-consistent with $r(t)$ are germ cells with such age. Therefore P_1 never crosses the gap path. Inequality (15.5) gives a lower bound on how fast $r(t)$ moves right. According to Lemma 15.8, P_1 passes at most 3 colonies. Edge $r(t)$ therefore can move right by at most $3Q$, giving

$$3Q \geq (r(v_n) - y_0)/B \geq \frac{v_n - v_0 - \mathsf{purge\text{-}t}}{3\tau_2} - 2\,\mathsf{heal\text{-}span} - \mathsf{island\text{-}size} - 11,$$

$$v_n - v_0 \leq 9\tau_2 Q + 3\tau_2(2\,\mathsf{heal\text{-}span} + \mathsf{island\text{-}size} + 11) + \mathsf{purge\text{-}t} < 10\tau_2 Q,$$

where we used (15.8) and (15.6). $\qquad\qquad\square$

Proof of Lemma 15.12. 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 $\leq k$.

1. Let $t_1 > v_0$. Assume that for all $t \leq t_1$, 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 $k + 6Lp_2\lambda$, and is traceable. Then $r(t_1)$ is an exposed edge, space-consistent with $r(v_0)$.

   Proof.

   1.1. Let us build a path.

   We start by building a backward trace-back path of length m

   $$P_2 = ((z_0, w_0) = (r(t_1), t_1), \ldots, (z_m, w_m))$$

   ending at time v_0. Lemma 15.8 and the bound e) gives

   $$m \leq \Delta + 1 + 2(v_n - v_0)/T_\bullet \leq \Delta + 1 + 2L\tau_2/T_\bullet < 3L\lambda p_2.$$

   The path does not end in a birth; indeed, otherwise $(r(t_1), t_1)$ would be a germ cell younger than $3Lp_1\lambda$ contrary to the assumption. It does not enter the gap; otherwise $(r(t_1), t_1)$ would be a germ cell younger than $k + 3L\lambda p_2 + 3L\lambda p_2$, which was excluded. Therefore defining $x = P_2(v_0)$ gives $x \geq y_0$. Without loss of generality, we can suppose $x \leq x_0$. Otherwise, Lemma 15.3 (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 H from (x, v_0) to (y_0, v_0) into a single chain of links. It has at most $3Q$ horizontal links on H, and then at most $\Delta + 1 + 2(v_n - v_0)/T_\bullet$ links on P_2, giving fewer than

   $$3Q + 2(v_n - v_0)/T_\bullet + \Delta + 1$$

links. We have $r(t) \geq B(y_0 - 2\,\text{heal-span} - \text{island-size} - 11)$. Indeed, the only way for $r(t)$ to move left is via an island (can happen at most once) that is then connected to the right end of the gap by healing.

1.2. A protected left edge cannot occur on $H + P_2$.

Proof. Suppose first that (y_0, v_0) is a member or right outer cell, or a germ cell with *Addr* > 0. Then the large bad gap on its left contains, by the definition of bad gaps, a gap in the same colony or in the originating colony. The path P_2 could cross it only by repeated parental links. But the size of the gap is too large for using rule `Heal-revive`, since the field *Healing* limits the number of applications. Instead, P_2 (when traveled forward) would first have to walk right via parental links to participate in the creation of a colony and then walk back via parental links through the expansion process from that colony—for which there is not enough time due to (12.3).

Suppose now that (y_0, v_0) is a left outer cell. By requirement d), then the path P_1 stays in the colony of y_0, and hence so does the whole gap path. Since the cell is exposed, its age is already past the expansion stages. Walking right, the age of cells on the path H (which remains in the same colony as well) does not decrease except when it crosses into another work period (this is taken care of by our Definition 15.7 of age progress that uses amod U). Indeed, by Definition 13.5 (Time consistency), the ages of siblings must be non-increasing as one is moving away from the center of the originating colony. It is also non-decreasing on the trace-back part P_2, except possibly on the $\leq \Delta$ horizontal links allowed by that Lemma 15.5 (Skirting). Therefore the age of $(r(t_1), t_1)$ cannot be smaller by more than Δ than that of (y_0, v_0). If the age is not smaller at all then $(r(t_1), t_1)$ is also exposed, since the number of steps on the path is not sufficient to get into an expansion stage of the next work period. If decreasing the age by Δ makes it protected, then one of the cases in Lemma 14.17 (Exposing) must have occurred within the last Δ steps before the age of (y_0, v_0): but this has been excluded by the condition c) of the lemma we are proving.

Suppose that (y_0, v_0) is a germ cell with *Addr* < 0. Since it is exposed to the left its age is $\geq \text{grow-end}$, and its address is $> -Q$. In this case, the same argument works as for left outer cells above.

A large gap cannot be closed by the healing rule, but it can be narrowed somewhat. Suppose that for a gap $G = (l, r)$ the cells $r, r + B, \ldots, r + kB$ are siblings with *Decaying*$(r + iB) = 1$, *Healing*$(r + iB) = k - i$. Then we set $r' = r + kB$. If these conditions are not satisfied then set $r' = r$. We define

similarly, $l' = l - kB$ for an appropriate k. The values l', r' will be called the *adjusted edges* of the gap, and we will write $G' = (l', r')$. the *adjusted size* is $|G'| = r' - l' - B$.

2. Assume that the gap $G(t)$ with the desired properties was defined up to time f_0, with adjusted size $|G'(f_0)| \geq 2\,\mathsf{heal\text{-}span}\,B + 2$ and right age $\leq k$. Assume also that in the area where we define the path further, all cells are unaffected even via neighbors during $(f_0 - \mathsf{purge\text{-}t}, f_1]$.

Then the gap path $G(t)$ can be defined further in $(f_0, f_1]$ in such a way that as long as $r(t)$ did not reach Z:

$$(r(f_1) - r(f_0))/B \geq \frac{f_1 - f_0}{3\tau_2} - \mathsf{heal\text{-}span} - 1, \qquad (15.9)$$

$$(|G(f_1)| - |G(f_0)|)/B \geq \frac{f_1 - f_0}{3\tau_3} - 2\,\mathsf{heal\text{-}span} - 2. \qquad (15.10)$$

If $r(t)$ reaches Z it never decreases again.

Proof. 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 $r(t_1) - B$ 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 creating a sibling of $l(t_1)$, then $l(t_2) = l(t_1) + B$.

2) t_2 is a switch of $r(t_1) - B$. If the switch is an animation creating a sibling of $r(t_1)$, then $r(t_2) = r(t_1) - B$.

3) 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 $k + 2(t_2 - f_0)/T_\bullet + 1$.

4) In all other cases, we leave $G(t)$ unchanged.

If $r(t)$ reaches Z, it will never decrease again. Indeed, this happens only if left outer cells were exposed to the left. Without loss of generality, assume that these outer cells were growth cells. In this case, $\mathit{Age}(Z) > \mathsf{grow\text{-}end}$. As Z is protected to the left, the healing rule does not create a left neighbor of Z. The only way that $r(t) = Z$ could move left is by the growth rule, but it does not work for $\mathit{Age}(Z) > \mathsf{grow\text{-}end}$.

Let us show

$$(r'(f_1) - r'(f_0))/B \geq \left\lfloor \frac{f_1 - f_0}{3\tau_2} \right\rfloor, \qquad (15.11)$$

which will prove (15.9). Here we assume that Z has not been reached. Since the conditions of part 1 are satisfied, the cell $y = r(t)$ is an exposed left

edge for all $t \in (f_0, f_1]$. Hence the longest it can take to move $r'(t)$ right is to have the rule `Heal-revive` move $r(t)$ left by up to heal-span cells; then all of them die by decaying. The `Heal-revive` steps and the killing of the healing cells take time at most τ_0 each, for a total of $2\,\mathsf{heal\text{-}span}\,\tau_0 = \tau_2$, setting the field *Decaying* takes time at most τ_2, and the final killing again time τ_0.

Now let us show (15.10). The inequality

$$(l'(f_1) - l'(f_0))/B \leq \left\lceil \frac{f_1 - f_0}{p_3 T_\bullet} \right\rceil \tag{15.12}$$

follows from the fact that $l'(t)$ can only increase when it is equal to $l(t)$ and increases via the the growth rule. But the next application of `Grow` is away by a waiting period of length $\geq p_3 T_\bullet$.

Let us combine (15.12) and (15.11).

$$(|G'(f_1)| - |G'(f_0)|)/B \geq (f_1 - f_0)\left(\frac{1}{3p_2 T^\bullet} - \frac{1}{p_3 T_\bullet} \right) - 2.$$

Now, using the definition of p_3 in (12.9),

$$\frac{1}{3p_2 T^\bullet} - \frac{1}{p_3 T_\bullet} = \frac{p_3 T_\bullet - 3p_2 T^\bullet}{3p_2 p_3 T_\bullet T^\bullet} \geq \frac{p_3 - 3p_2 \lambda}{3p_2 p_3 T^\bullet} \geq \frac{1}{3p_3 T_\bullet},$$

hence the estimate (15.10).

3. Let $[a_0, a_1) \times (u_0, u_1]$ be an island with $u_0 \in (v_0, v_n]$. Then the gap path $G(t)$ with the desired properties can be defined for $t \in (u_0, u_1 + \mathsf{purge\text{-}t}]$. We will have

$$r(u_1 + \mathsf{purge\text{-}t}) \geq r(u_0) - (\mathsf{island\text{-}size} + 9)B,$$
$$|G(u_1 + \mathsf{purge\text{-}t})| \geq |G(u_0)| - (2\,\mathsf{island\text{-}size} + 19)B.$$

Proof. Let us define $l(t)$ for $t \in (u_0, u_1 + \mathsf{purge\text{-}t}]$. If $a_0 \leq l(u_0) + 5.1B$, then we set $l(t)$ to be the first site x aligned with $l(u_0)$ to the right of a_1 and $l(u_0) + B$ that is unaffected even via neighbors. This definition is justified, since growth or healing do not have time to add any cell further to the right of $l(u_0) + B$. Now $l(t) \leq l(u_0) + (5.1 + \mathsf{island\text{-}size} + 3.1)B < l(u_0) + \mathsf{island\text{-}size} + 9B$. If $a_0 > l(u_0) + 5.1B$, let $l(t) = l(u_0) + B$ since, as noted above, at most one cell can be added to $l(u_0)$ by growth or healing during $[u_0, u_1 + \mathsf{purge\text{-}t})$. The definition of $r(t)$ is done symmetrically similarly (but only as a lower bound).

Thus, the island may decrease the gap by at most island-size $+ 9B$ on one side; with another possible decrease of size B on the other side, this gives the size estimate of statement 3.

In the space-time rectangle considered, at most one island occurs: indeed, $v_n - v_0 \leq T_\bullet^*/2$ follows from e) and (12.3). Use the construction of part 2 up to the time u_0 of the start of the island, then construction of part 3 from u_0 to u_1+purge-t (or v_n, whichever comes first), then possibly again the construction of part 2 from $u_1 +$ purge-t to v_n, we defined $G(t)$ for all $t \in [v_0, v_n]$.

4. For all $t \in [v_0, v_n]$, the gap $G(t)$ has right-age $\leq k + 2(t - v_0)/T_\bullet + 1$. In particular, according to the assumed bound on $v_n - v_0$, at time v_n, it still has age at most

$$k + 2(v_n - v_0)/T_\bullet + 1 \leq k + 2L\tau_2/T_\bullet + 1 < k + 3Lp_2\lambda,$$

as stated in the lemma.

Proof. Let z_1 be a live cell in $G(t)$ space-consistent with $r(t)$, 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 there is no other possibility. Indeed, otherwise, z_1 would have been created by animation which, by Condition 14.16 can only be by healing or growth. Since $r(t)$ is exposed, growth can only happen from the left. We end up with one of the cases of the definition of $G(t)$ above, and thus with $z_1 \notin G(w_1)$, contrary to the assumption.

By the parent construction, $w_2 \leq w_1 - T_\bullet$, and $\mathsf{Age}(z_1, w_1) \leq \mathsf{Age}(z_2, w_2) + 1$. The value w_i decreases at least $T_\bullet/2$ in every step, while the age of (z_i, w_i) can decrease by at most 1. Since it can only end in a birth or a germ cell in $G(t)$, this and e) proves the age bound.

5. Let us lower-bound the growth of $G(t)$ for $t \in [v_0, v_n]$.

Since we applied the estimate of part 2 twice and the estimate of part 3 once, we get

$$(r(v_n) - r(v_0))/B \geq \frac{|v_n - v_0 - \mathsf{purge\text{-}t}|^+}{3\tau_2} - 2\,\mathsf{heal\text{-}span} - \mathsf{island\text{-}size} - 11,$$

and

$$(|G(v_n)| - |G(v_0)|)/B$$
$$\geq \frac{|v_n - v_0 - \mathsf{purge\text{-}t}\,|^+}{3\tau_3} - 4\,\mathsf{heal\text{-}span} - 2\,\mathsf{island\text{-}size} - 21.$$

Hence using (15.1), $|G(t)|$ never decreases below $|G(v_0)| - \mathsf{gap\text{-}lb}\,B > 0$.

$\square$

15.3 Non-damage gaps are large

We now show that if healing does not succeed, then a large gap develops. 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. Recall the definition of heal-span in (12.5) and split-n, split-t in (15.3-15.4).

Lemma 15.14 (Healing Cause) *Let $c_0 = (x_0, t_0)$ be a cell with Healing$(c_0) > 0$,*

$$D = x_0 + [-(\mathsf{heal\text{-}span} + 0.1)B, (\mathsf{heal\text{-}span} + 1.1)B),$$
$$I = (t_0 - 2\,\mathsf{heal\text{-}span}\,T^\bullet, t_0].$$

Assume that $D \times I$ is damage-free. Then there is a backward path with at most heal-span $- 1$ *horizontal links, at most 1 parental link, and at most* $2\,$heal-span *links altogether, connecting c_0 to an exposed cell.*

A similar statement holds if Frozen$(c_0) \neq 0$, except that here the length of the path is bounded by 2Δ, with Δ defined in (13.4).

Proof. Let us build a path $P = (c_0, c_1, \ldots, c_n)$ with $c_i = (x_i, t_i)$ backwards with the property that each of its cells c_i with $i < n$ has *Healing* > 0, and c_n is exposed. According to Condition 14.13, `Heal-revive` is the only rule changing the value of field *Healing*. It sets the value to 0 immediately unless x_i is exposed or has a neighbor c_{i+1} with a higher *Healing* value. Therefore if *Healing*$(c_i) > 0$ and it was not set to 0 in the last step, there are the following possibilities:

- There is an exposed parent.

- The cell itself was exposed at the observation time.

- There was a neighbor y with a higher *Healing* value at the last observation time $t' < t_i$ of x_i.

In the first two cases, the path is finished with a last link. In the last case, we add two links: a vertical link to $c_{i+1} = (x_i, t')$ followed by a horizontal link to $c_{i+2} = (y, t')$. Since this leads to an increase of *Healing* value, the path can contain at most heal-span pairs of such links.

The statement on the field *Frozen* is proved the same way, using Condition 14.10 and rule `Freeze`. $\square$

Definition 15.15 (Boundary path) Let us call the sibling relation between two cells *strong*, if either they have been siblings for at least $2T^{\bullet}$ time units or one cell is a parent of the other.

A cell is called for example a *weak left exposed edge* if it is either a left exposed edge or would be one in case it had no left sibling, and has no strong left sibling. A sequence $R_1 = (y_0, v_0), \ldots, (y_m, v_m)$ of cell-time pairs with $v_0 < v_1 < \cdots < v_m$ will be called a *left boundary path* if y_i is a weak left exposed edge during (v_i, v_{i+1}) and one of the following is true:

1) Cell $y_{i-1} = y_i - B$ dies at time t_i and is a strong sibling of y_i at time t_i-.

2) Cell $y_i = y_{i-1} - B$ is created at time t_i by parent y_{i-1}.

$\lrcorner$

Lemma 15.16 (Bad Gap Opening) *For $m = $ split-n let $R_1 = (y_0, v_0), \ldots, (y_m, v_m)$ be a left boundary path that does not leave the colony of (y_m, v_m) on the left (it may leave on the right). Assume that the the segment*

$$[\textstyle\bigwedge_i y_i - 1.1B, \bigvee_i y_i + 1.1B) \times (v_0 - T^{\bullet}, v_m]$$

is noise-free. Then y_i has at most $p_2 + p_0$ full dwell periods during $(v_i, v_{i+1}]$, and there is a gap on the left of $R_1(v_m-)$ that contains a right bad gap. The same statement holds if we interchange left and right.

Proof. Let us show that (y_i, t) is a left exposed edge during $(v_i + 2T^{\bullet}, v_{i+1} - 2T^{\bullet})$. 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 14.19 (Glue) (and the exclusion of cut, since the it would not expose the edge) 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.

Now, as soon as y_i is a left exposed edge, unless the healing rule creates a left sibling, it gets *Decaying* $\leftarrow 1$ in p_2 dwell periods, and then the decay rule

kills it in further p_0 dwell periods. Hence within $p_2 + p_0$ dwell periods, the boundary paths moves left or right. Now a reasoning similar to part (15.11) of the proof of Lemma 15.12 (Running Gap) shows that the left movement is associated with a healing step of rule *Heal-revive*, and once the right movement starts it will not stop before killing at least one cell with *Healing* = 0. Thus within each $2\,\mathsf{heal\text{-}span} + 1$ moves, the edge moves right. In $\mathsf{gap\text{-}lb} + 1$ repetitions of this, the weak left exposed edge moves right by this many steps. During this time, by (12.8), at most one growth step can occur on the left, so a gap of width $\mathsf{gap\text{-}lb}\,B$ will be created which is internal by definition, since we assumed that the boundary path does not leave the colony of (y_m, v_m) on the left.

To show that it is a bad gap, we bound the age of its germ cells. The total number of (full or partial) dwell periods along the boundary path is at most $m(p_2 + p_0 + 1)$. During this time, the age of any germ cell created in place of the killed cells can grow by at most $\lambda m(p_2 + p_0 + 1)/p_1$. By the definition of p_i in (12.6-12.8), we have

$$(p_2 + p_0 + 1)/p_1 \leq \frac{2\,\mathsf{heal\text{-}span}\,\lambda p_0 + p_0 + 1}{4\lambda p_0} \leq \mathsf{heal\text{-}span}\,/2 + 1,$$

so the total germ age is at most $\lambda\,\mathsf{split\text{-}n}\,(\mathsf{heal\text{-}span}\,/2 + 1)$, making the gap by Definition 15.9 a bad gap. $\qquad\square$

Lemma 15.17 (Bad Gap Inference) *Let $c_0 = (x_0, t_0)$ be a left exposed edge,*

$$D = [x_0 - (\mathsf{split\text{-}n} + 1.1)B, x_0 + (\mathsf{heal\text{-}span} + 1.1)B)\,,$$
$$I = (t_0 - \mathsf{split\text{-}t}\,, t_0].$$

Assume $D \times I$ is damage-free. Then one of the following holds:

1) *There is a bad gap on the left of c_0;*

2) *There is a boundary path of length $< \mathsf{split\text{-}n}$ leading backwards in time from c_0 to a cell undergoing one of the changes listed in Lemma 14.17 (Exposing).*

The same statement holds if we replace left with right.

Proof. We will distinguish a special case when c_0 is involved in the leftward rule *End-heal* (Algorithm 14.21) case of healing, calling this the (leftward) *end-heal* case.

Let us construct a backward path (x_i, t_i), $i = 0, 1, \ldots, n$ of elements of the colony of c_0, 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}). Suppose that (x_i, t_i) has already been constructed.

i) 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 $(x_{i+1}, t_{i+1}) = (x_i, t')$.

ii) 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 $(x_{i+1}, t_{i+1}) = (x_i, t')$.

iii) If $t' = t_i$ then at time t_i, cell x_i became a weak left exposed edge without switching, which can happen in the following ways::

   x1) Losing protected status via one of the cases of Lemma 14.17 (Exposing), that is case 2) of the present lemma. This cannot happen in the end-heal case.

   x2) By animation via a right parent, which by Condition 14.16, can only happen via the healing rule. Let $(x_{i+1}, t_{i+1}) = (x_i + B, t_i)$. Such an i will be called a *right jump* (backwards in time).

   x3) Losing a strong left sibling by one of the ways listed in Lemma 14.19 (Glue). It is easy to check that of these, only the death of a strong left sibling produces an exposed edge in x_i. Each of the rules `Heal`, `Decay`, `Purge` and `Adapt` 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 $2T^\bullet$ time units. Let $(x_{i+1}, t_{i+1}) = (x_i - B, t_i)$. Such an i will be called a *left jump* (backwards in time).

   x4) The strong left sibling was killed by the right end-healing of member cells of another colony on the left (via `Adapt`). This clearly cannot happen in the end-heal case of the present proof. We will see, moreover, that it does not happen at all.

The construction can stop in the following ways:

s1) The number of jumps on the path reaches split-n.

s2) Cell (x_n, t_n) would be a weak left exposed edge belonging to the left neighbor colony. This cannot happen in the end-heal case.

By the construction, each jump is surrounded by vertical links. For each left jump i, cells x_i and x_{i+1} are strong siblings at time t_i-. Let (x_{i_k}, t_{i_k}) for $i_1 < \cdots < i_{m-1}$ be the points on the backward path with the property that to $(i_k, i_k + 1)$ belongs a horizontal link. Let us number these points forwards in time: $(y_0, v_0) = (x_n, t_n)$, $(y_1, v_1) = (x_{i_{m-1}}, t_{i_{m-1}})$, $(y_2, v_2) = (x_{i_{m-2}}, t_{i_{m-2}})$, ..., $(y_{m-1}, v_{m-1}) = (x_{i_1}, t_{i_1})$, $(y_m, v_m) = (x_0, t_0)$, creating a left boundary path as in Definition 15.15. If $m \geq$ split-n while we do not have case s2) then Lemma 15.16 (Bad Gap Opening) implies a bad gap on the left of c_0.

In case s2), the death of (x_n, t_n) creates the weak left exposed cell (x_{n-1}, t_{n-1}), a left colony endcell. By Definition 13.11, this is possible only by the shrinking process, as described after Algorithm 12.1. (It is clear from there that (x_{n-1}, t_{n-1}) is not a member cell.) Continue the construction of the left boundary path. Now it can be stopped in fewer than split-n steps only by one of the changes listed in Lemma 14.17, since split-n $< Q$; now Lemma 15.16 is applicable.

Let us now show that case x4) does not occur at all. Assume it does and let y be the right-exposed cell trying to do the end-healing. Then we can apply the present lemma to y. Since it is the (rightward) end-heal case, a bad gap is implied on the right of y, so y is far on the left of x_i and cannot possibly kill its left sibling.

$\square$

16 Attribution, progress

In this section Lemma 16.3 (Attribution) shows that a non-germ cell implies a full colony nearby in the near past related to it in one of several identifiable ways. The main tool of the proof is a trace-back path. The following lemma treats various possibilities for the path. Recall the definition of destr-t in (15.6) and the definition $n_{\text{l-m}}$ for the number of update ages of locally maintained fields in (13.1) and denote

$$\text{attrib-t} = \text{destr-t} + 9\tau_1. \tag{16.1}$$

For the lemmas below, for a time t_0 let

$$\begin{aligned} J &= t_0 + (-\text{attrib-t}\, Q, 0], \\ K &= t_0 + (-\text{attrib-t}\, Q, -\text{destr-t}\, Q]. \end{aligned} \tag{16.2}$$

Lemma 16.1 (Cover) *Let $c_0 = (x_0, t_0)$ be a live cell belonging to colony $\mathcal{C}$ with starting cell z,*

$$I = [z - (Q + 1.1)B, z + (2Q + 1.1)B).$$

For J as in (16.2) let

$$n_J = 2|J|/p_1 T_\bullet + 3\Delta$$

and assume that Damage* does not intersect $I \times J$. Consider a trace-back path P_1 in $I \times J$ leading backwards from c_0 to a cell $c_1 = (x_1, t_1)$. The following holds.

a) *If c_0 is not an outer or germ cell younger than* grow-end $+ n_J$ *then the path P_1 does not leave its own colony.*

b) *Assume $t_1 \leq t_0 -$ destr-t Q, the rectangle*

$$I \times (t_1 - \text{split-t}, t_1 + \text{split-t}]$$

is damage-free, c_1 is not a doomed member cell with $|Age(c_1) > U - 2Q -$ split-n. Then the maximal domain containing the cell c_1 contains the colony of c_0 as well as its originating colony.

We call this the Cover Lemma since part b) implies that the originating colony is covered by a domain.

Proof. 1. The age increase along the path is at most n_J. Indeed, for the path length n we have $n \leq 2|J|/T_\bullet + 2\Delta + 2$, from part b) of Lemma 15.8 (where Δ was defined in (13.4)). The age increase along the path can be upper-bounded using Lemma 15.8 as

$$n/p_1 + 2\Delta + 3 \leq 2|J|/p_1 T_\bullet + 2\Delta(1 + 1/p_1) + 3 + 2/p_1.$$

The sum all terms but the first can be bounded by 3Δ.

2. Let us prove a). According to Lemma 15.5 (Skirting), the horizontal links on the path can only cross a colony boundary in the inward direction, and only if $Age <$ grow-end $+ \Delta$ on all cells involved. A parental link could be created only by the healing rule or one of the expansion rules. The healing rule does not act across colony boundaries, so the later end of the link along which the path is leaving would need to be an expansion or germ cell. The expansion rules do not work for cells past age grow-end, so in each case, the cell at the crossing would need to have an age $\leq$ grow-end. By part 1 above, the age could not grow beyond grow-end$+2|J|/p_1 T_\bullet+3\Delta$, so it does not reach the lower bound required by a).

3. Let us prove b). Assume that the maximal domain contains an exposed edge e. Lemma 15.17 (Bad Gap Inference) implies that either there is a bad gap next to e or we have case 2) of that lemma.

In case of a bad gap, we can apply Corollary 15.13 (Running Gap) to the assumption $t_1 \leq t_0 -$ destr-t Q, where the role of the path P_1 there is played by path P_1 here. It implies that the bad gap cannot be inside the originating colony of c_0, nor can it be in the same colony (as the gap path cannot cross P_1).

In case 2) of Lemma 15.17 there is a path of length $<$ split-n leading back to an edge that becomes exposed by one of the cases listed in Lemma 14.17

(Exposing). If it is one the cases different from 1) then it is outside the originating colony and pointing away from it. Case 1) does not occur; indeed, it could only occur to member cells c_1, but the condition $|Age(c_1) \text{ amod } U| > 2Q + \mathsf{split\text{-}n}$ implies $|Age(e) \text{ amod } U| > \mathsf{split\text{-}n}$.

$\square$

Here is the sense in which we will be attributing a cell $c_0 = (x_0, t_0)$ to some colony. Recall the definition of J, K in (16.2).

Definition 16.2 (Attribution) We say that c_0 is *attributed* to colony $\mathcal{C}$ if there is a path P_1 going back to time $t_0 - \mathsf{attrib\text{-}t}\, Q$, and a union E_0 of intervals of total length $< |K|/2$ such that $P_1(t) \in \mathcal{C}$ for t in K, and $\mathcal{C}$ is covered by a domain for all times in $K \setminus E_0$.

⌟

Suppose that another cell (x_1, t_0) is attributed to some colony $\mathcal{C}_1$; Then even the union of their exception sets $E_0 \cup E_1$ has a total length $< |K|$, and therefore there is some time in K where both colonies C_0 and C_1 are covered by a domain. It follows that they are either disjoint or identical. The following lemma again uses the intervals J, K in (16.2).

Lemma 16.3 (Attribution) *Assume that the live cell $c_0 = (x_0, t_0)$ is traceable, and is not a germ cell younger than $\mathsf{grow\text{-}end} + 2|J|/p_1 T_\bullet$. Then we have:*

 a) *c_0 can be attributed to its originating colony.*

 b) *If it is a member cell, or an outer or germ cell older than $\mathsf{grow\text{-}end} + |J|/p_1 T_\bullet$ then it can also be attributed to its own colony.*

Proof. Build a trace-back path P_1 from c_0 backwards in time. Due to the absence of $Damage^*$, at most one island occurs in the area of P_1. Lemma 16.1 (Cover) implies that unless c_0 is an outer or germ cell younger than $\mathsf{grow\text{-}end} + 2|J|/p_1 T_\bullet$, we can continue the path P_1 all the way to time $t_0 - \mathsf{attrib\text{-}t}\, Q$ and stay in the colony of c_0. Let E be the set of times t of K that are either within distance $\mathsf{split\text{-}t}$ of the damage or such that $|Age(P_1(t)) \text{ amod } U| \leq 2Q + \mathsf{split\text{-}n}$. Then $|E| \leq 2\,\mathsf{split\text{-}t} + (4Q + \mathsf{split\text{-}n})\tau_1 < |K|/2$. Lemma 16.1 implies that for all times t in $K \setminus E$ the cell $P_1(t)$ is contained by a domain covering its originating colony.

In case b), the domain must contain the colony of c_0, because being past $\mathsf{grow\text{-}end}$ otherwise it would have an exposed edge. So c_0 can be attributed to its own colony. $\square$

We will estimate the amount of progress in a given time made by a colony in the absence of higher-order damage.

Lemma 16.4 (Colony Trace-back) *Consider a time interval $(t_0, t_1]$. Suppose the following, with $I = z_0 + [-1.1QB, 2.1QB)$:*

1. *The rectangle $I \times (t_0 - \mathsf{split\text{-}t}, t_0]$ is damage-free, and Damage* does not intersect $I \times (t_0 - \mathsf{split\text{-}t}, t_1]$.*

2. *At time t_1, colony $\mathcal{C}$, with starting cell z_0, is covered by a maximal domain with member cells in $\mathcal{C}$, and no exposed edges.*

Assume that some trace-back path P starts within $0.2QB$ of the center at time t_1, goes back to time t_0, and does not come within $2Q$ steps of a work period boundary time. Then:

a) *Path P moves at most to a distance $2\lambda\,\mathsf{heal\text{-}span}\,\mathsf{split\text{-}n}\,B$ from where it started. If there is no damage in $I \times (t_0, t_1]$ then P does not move horizontally at all.*

b) *The colony $\mathcal{C}$ is covered by a maximal domain without exposed edges also at time t_0.*

c) *The age progress from t_0 to t_1 along path P is at most $\frac{t_1 - t_0}{p_1 T_\bullet} + \Delta + 3$, and at least $\frac{t_1 - t_0}{4 p_1 T^\bullet} - 6Q$.*

The following corollary is just the inversion of part c) of the above lemma.

Corollary 16.5 (Progress) *Under the assumptions of Lemma 16.4, suppose that the age progress along the path is n. Then*

$$(n - \Delta - 3) p_1 T_\bullet \leq t_1 - t_0 \leq 4(n + 6Q) p_1 T^\bullet.$$

Proof of Lemma 16.4. Let $P = (c_1, c_2, \dots)$ be the trace-back path mentioned in the lemma, listed backwards in time, with $c_i = (x_i, u_i)$. In the given time interval, at most one island intersects colony $\mathcal{C}$. Let K be the interval of times within $\mathsf{split\text{-}t}$ of this island, then $|K| \leq 2\,\mathsf{split\text{-}t} + \mathsf{island\text{-}size}\,T^\bullet$.

1. For times outside K, the maximal domain containing the cell $P(t)$ has only protected edges.

   Proof. If there was an exposed edge, then Lemma 15.17 (Bad Gap Inference), combined with Lemma 15.12 (Running Gap) would imply that the colony $\mathcal{C}$ would not be covered by a domain at time t_1.

2. Claims a) and b) are true.

   Proof. The island may cause at most Δ horizontal links (with Δ defined in (13.4)), all other links are vertical or parental. As shown in part 1, the path can have parental links only within distance $\mathsf{split\text{-}t}$ in time of the island. Since they use $\mathit{Adapt}(\cdot)$ with a delay of at least p_1, parental links are separated from each other by at least p_1 vertical links. If follows then, using

the definition of split-t and split-n in (15.3-15.4), that the path can move at most $\Delta + |K|/p_1 T_\bullet$ steps away, where

$$|K|/p_1 T_\bullet = (2\,\text{split-t} + \text{island-size})\tau_2\,\text{split-n}\,/p_1 T_\bullet \leq 3\lambda\,\text{heal-span}\,\text{split-n}\,/2$$

(and will therefore stay in the colony). Assume now that, contrary to claim a), there is a first cell c_i with a parental link to c_{i-1}, farther than split-t in time from the island. Then x_i is at distance at most $\lambda\,\text{heal-span}\,\text{split-n}\,B$ from x_1, and as by assumption does not come within $2Q$ steps of a work period boundary time, is an exposed member cell, or growth cell with age $\geq$ grow-end, inside the the colony. Lemma 15.17 (Bad Gap Inference) implies then a bad gap which would not close until time t_1, contrary to the assumptions. Reference to the same lemma proves also the statement b).

3. Claim c) is true.

   Proof. The upper bound follows because according to Lemma 15.5 (Skirting) there are at most $\Delta + 1$ horizontal links in the path, while along the vertical part of the path, age increases at most by 1 every p_1 steps. We proceed to the lower bound.

   3.1. We will build a new path, $P' = (c'_0, \ldots, c'_m)$ with $c'_i = (x'_i, t'_i)$.

   Start with $c'_0 = c_0$. Within the set K we continue P' as a trace-back path; consider the cases $t'_i \notin K$. We have $\textit{Healing}(x'_i) = 0$ and $\textit{Frozen}(x'_i) = 0$ for times near t'_i, since otherwise Lemma 15.14 (Healing Cause) would imply the nearby presence of exposed edges. In the absence of freezing, the age cannot decrease in time. If $\textit{Age}(x'_i)$ did not increase in time t'_i, then this can only have the following reasons:

   1) Cell x'_i is in the process of decreasing its $\textit{Wait}.\texttt{March}$ variable from p_1.
   2) Cell c'_i has just been born.
   3) There is a neighbor y at the last observation time t' of x'_i that does not let its age advance, by condition (1) of the $\texttt{March}$ rule 13.3. Thus it is either lower in age (call this a *delay link of the first kind*) or is equal in age but then it is closer to the center of the originating colony (call it a *delay link of the second kind*).

   In cases 1) and 2), let c'_{i+1} be found by regular trace-back: thus, the previous switching time of x'_i if there is one, and the parent, if x'_i has just been born. In case 3), let $c'_{i+1} = (x'_i, t')$, $c_{i+2} = (y, t')$.

   We will now lower-bound age decrease and upper-bound age decrease backwards on path P'.

   3.2. Let us upper-bound age increase inside the set K.

Inside the time set K, the only horizontal links on the path P' come from Lemma 15.5 (Skirting), adding at most Δ to the total age increase backward. There is no increase on parental links, and at most one backward increase on each vertical link (as `Heal-sync` may decrease age). This is a total increase of at most $|K|/T_\bullet + \Delta$.

3.3. Let us lower-bound age decrease outside the set K.

Going to the parent via alternative 2) above is always going towards the originating colony: since we are in a maximal domain without exposed edges, the only possible kind of cell creation is growth. The cells created by growth have age equal to the creator cell. During a colony work period, there are at most $4Q$ growth steps (growing a channel and then a growth arm to the left, and also to the right), this bounds the number of parental links. Let n_1 be the number of delay links of the first kind and n_2 that of delay links of the second kind, $n = n_1 + n_2$. We have $n_2 \le 2Q + n_1$, as one can move towards the center of the originating colony only after being moved away from it. The total age decrease on these non-vertical links is hence at least $n_1 \ge n/2 - Q$. The total number of non-vertical links is $m \le 4Q + n$, so

$$n_1 \ge (m - 4Q)/2 - Q = m/2 - 3Q. \tag{16.3}$$

In a vertical segment of the path of time projection l, age decreases at least by 1 in every p_1 vertical steps. So it decreases at least $0 \vee (\frac{l}{p_1 T_\bullet} - 1)$. Let us call a vertical segment *short* if this number is < 1, that is whose length is $\le 2p_1 T^\bullet$, else *long*. On a long segment of length l, the decrease is at least $\frac{l}{p_1 T^\bullet} - 1 \ge \frac{l}{2p_1 T^\bullet}$. Let L_1 be the union of long vertical segments, and L_2 that of the short ones. The time decrease on L_1 is at least $\ge |L_1|/2p_1 T^\bullet$. The number of non-vertical links is at least $m \ge |L_2|/2p_1 T^\bullet - 1$, and by (16.3) the age decrease on these links is at least

$$m/2 - 3Q \ge |L_2|/4p_1 T^\bullet - 3Q - 1.$$

So the total decrease of at least

$$|L_1|/2p_1 T^\bullet + |L_2|/4p_1 T^\bullet - 3Q - 1 \ge \frac{t_1 - t_0 - |K|}{4p_1 T^\bullet} - 3Q - 1.$$

3.4. Let us add up all the decreases inside and outside of K.

We get

$$\frac{t_1 - t_0 - |K|}{4p_1 T^\bullet} - 3Q - 1 - |K|/T_\bullet - \Delta = \frac{t_1 - t_0 - |K|}{4p_1 T^\bullet} - V,$$
$$V \le 3Q + |K|/T_\bullet + \Delta + 1 \le 4Q,$$

where the last step follows from our assumptions about Q. At time t_0, there is no exposed edge between $P(t_0)$ and $P'(t_0)$, since this would imply a bad gap between these cells. This could not close, and at least one of the paths would need to cross it in order to meet, which is not possible. Therefore these two nodes are connected by a chain of siblings and so their age differs by at most $2Q$. Since $P(t_1) = P'(t_1)$, this shows that the age progress along P differs from that of P' by at most $2Q$.

$\square$

17   Healing

This section shows how the effect of an island will be healed. By its definition, the healing process only tries to bridge gaps inside a colony of member cells. If a connecting arm gets damaged, it can just regrow from the healthy part. If growth is thwarted by damage, and is still possible in the next work period, then it will happen then. Similarly, a germ killed by damage can regrow later.

The effect of an island can be delayed, in one case.

Definition 17.1 (Successor) 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 trace-back path of cells from some (x, t) with x in E.   ⌟

Recall the definition of siblings in Definition 13.6, and the definition of (multi-) domains in Definition 13.13. Let

$$\mathsf{heal\text{-}t} = \mathsf{purge\text{-}t} + 4\Delta\tau_1, \tag{17.1}$$

where $\mathsf{purge\text{-}t}$ was defined in (15.2).

Lemma 17.2 (Healing) *Let* $[a_0, a_1) \times (u_0, u_1]$ *be an island. Let* $\mathcal{C}$ *be a colony with starting cell* z_0, *covered at time* u_0 *by a domain of member cells, with no doomed cells of age* $> U - 3Q$. *Then a domain covers* $\mathcal{C}$ *at time* $w_0 = u_1 + \mathsf{heal\text{-}t}$.

Proof. The damage affects directly at most Δ neighbor cells of $\mathcal{C}$ (with Δ defined in (13.4)). At time t, let $L(t)$ be the set of successors in $\mathcal{C}$ of cells

of $\mathcal{C}$ on the left of the affected part, and $R(t)$ the successors from cells to the right. (These sets are not necessarily disjoint.) Both $L(t)$ and $R(t)$ are (possibly empty) domains; indeed, the only rule that would change the domain property is the killing of a doomed cell at age 0, but there is not enough time for this.

1. Let $u_2 = u_1 + \mathsf{purge\text{-}t}$. If the sets $L(u_2)$ and $R(u_2)$ are both nonempty then there is no live cell between them at time u_2 other than possibly germ cells with age ≤ 1.

   Proof. The only cells in question are the up to Δ cells affected directly by the damage, others could not have reborn yet after dying. Consider the up to Δ domains between $L(u_1)$ and $R(u_1)$; they can shrink or merge during the time interval $(u_1, u_2]$. According to Lemma 13.14, each will still have an exposed side, hence each will be killed by the time u_2 by the purge rule (while having no time yet to grow due to the delay in the rule *Adapt*)— unless they are germs so new that the purging did not have time to finish yet: in this case they have age ≤ 1.

2. Any live cell at time u_2 whose body intersects $\mathcal{C}$ and that does not belong to $L(u_2) \cup R(u_2)$ cannot be a member cell.

   Proof. Assume by way of contradiction, that there is such a member cell x. In this case clearly either $L(u_2)$ or $R(u_2)$ is empty; without loss of generality assume that $L(u_2)$ is empty. Consider a trace-back path from x. It cannot lead to $\mathcal{C}$ at time u_0, since then according to part 1 above, x would belong to either $L(u_2)$ or $R(u_2)$. Therefore it must lead to a domain outside; given that $R(u_2)$ is nonempty, this must be on the left. Let y be the right end of this domain at time u_0. It must be outside $\mathcal{C}$, so it is to the left of x whose body, by assumption, intersects $\mathcal{C}$. By Lemma 15.5 (Skirting), the path has at most Δ horizontal links, and one parental link (there is no time for a complete animation delay), so x and y are within distance $(\Delta + 1)B$ from each other. All cells of the short chain from (x, u_2) to (y, u_0) have ages within a few steps of each other. If (y, u_0) is a right colony end then x (assumed to be a member cell to the left of $R(u_2)$) would have been purged away due to its small depth. If it is not then the right edge of (y, u_0) is exposed, since it is either a member cell not at a colony end or a growth cell near the end of the work period, and so during its shrinking phase. But then Lemma 15.17 (Bad Gap Inference) would imply large gap on the right of (y, u_0), making the path from it to (x, u_2) impossible. (Indeed, due to the age of (y, u_0) the conclusion 2) of that lemma is not applicable.)

3. If both $L(t)$ and $R(t)$ survive until time u_2, then the gap between them will have size $\leq \Delta$ during the whole time. If $L(t)$ disappears then the the gap from z_0 to $R(t)$ has size $\leq 2\Delta$ during the whole time.

Proof. If both $L(t)$ and $R(t)$ survive, then there were up to Δ cells affected by damage between them. Only these cells can disappear. If $L(t)$ disappears, this is only since its size was $\leq \Delta$ to begin with.

4. If both $L(t)$ and $R(t)$ remain nonempty then they will become part of a domain.

Proof. As shown above, between times u_2 and w_0, the gap between $L(t)$ and $R(t)$ does not become larger than Δ. Rule **Heal-revive** brings together $L(t)$ and $R(t)$ at some time $t \leq \Delta\tau_1$, at some point $x \in L(t)$, $x + B \in R(t)$. The possible germ cells of age < 2 in the gap are not an obstacle: they are weaker than the cells that are being created, so they will be erased. It can only happen that $L(t)$ and $R(t)$ are not joined if $|\mathit{Age}(x) - \mathit{Age}(x + B) \bmod U| > 1$. But as said in the remark at the end of Example 14.15, then rule **Heal-sync** will bring the two ages together in time at most $2\Delta\tau_0$. This creates a single domain by time $u_2 + \Delta(\tau_1 + 2\tau_0)$.

5. If $L(t)$ does not survive then $z_0 \in R(w_0)$.

Proof. The left endcell of $R(t)$ will be exposed to the left and therefore the rule 14.21 (**End-heal**) will apply to it. Part 2 above shows that no member cell's body intersects the colony $\mathcal{C}$. Other possible cells that might intersect are weaker than the member cells that end-healing creates, and will therefore be killed off by rule **Adapt** of Algorithm 14.6. The end healing thus proceeds in time $2\Delta\tau_1$.

$\qquad\square$

18   Communication

Here, we will give the rules for information retrieval, which eventually will be used in rules **Send** and **Retrieve** at the beginning of the work period as in Algorithm 12.1. Compared to Section 9.3, there are two main differences:

A) the neighbor colonies may not be adjacent;

B) they may be in different stages of their work period.

To deal with issue A), during the time of observation, a colony whose neighbor is not adjacent extends an arm of channel cells. By Definition 12.4, rightward channels would override leftward channels. This way at the time of communication there will be only one channel arm between the neighbor colonies, assuring that their distance is less than QB.

To deal with issue B), within the colony work period, note that the *unsafe* time interval of the colony work period during which information to be sent to neighbor colonies can change starts with the time when one of the cells is within $2Q$ of the beginning of the work period or the end of the **Compute**

rule. The retrieved information from the neighbor colonies will carry the age of sending. The information will be used only when it comes from both of these colonies at a safe ages. The safe interval is designed to be much longer than the unsafe one, so a safe time will be found. We define *Mail-ind* as in Section 9.3, with some additions:

$$Mail\text{-}ind = \{(k,d) : k \in \{-1,0,1\}, d \in \{-1,1\}\}.$$

Recall that we defined $e_{-1} = 0$, $e_1 = Q - 1$ in (14.2). The relation of cells and their originating colonies will define the predicate $Edge_j(x)$. In words: $Edge_j(x) = 0$ if y is a neighbor in the same extended colony, 1 if it is an adjacent neighbor in an adjacent colony, 1.5 if it is a neighbor in an (extended) neighbor colony, but only one of the two colonies at the junction can be an extended one. Formally:

$$Edge_j(x) = \begin{cases} 0 & \text{if } \textsf{Addr}(y) = \textsf{Addr}(x) + j, \\ 1 & \text{if } y = \vartheta_j(x),\ \textsf{Addr}(x) = e_j,\ \textsf{Addr}(y) = e_{-j}, \\ 1.5 & \text{if } Edge_j(x) \notin \{0,1\},\ y = \vartheta_j(x), \\ & (\textsf{Addr}(x) - e_j)/j \geq 0,\ (\textsf{Addr}(y) - e_{-j})/j \geq 0, \\ & \text{but at most one of these is } > 0. \\ \infty & \text{otherwise.} \end{cases}$$

The additional issues that copying has to deal with are the unsafe times, as mentioned above, non-adjacent colonies, and that the mail track may be narrower than the information in any one cell it needs to carry. In particular, the **Payload** field will consist of *slices* **Payload**$_i$ only one of which fits into the mail field. It will be convenient to have a separate mail field for every kind of copying from some track **F** to some track **G**, that is

$$\textsf{Mail}_{k,d,F,G}.$$

However, while discussing the general rules covering all kind of mail, we will suppress the indices F, G. We have three widths to consider: that of the sender, of the receiver, and of $\textsf{Mail}_{k,d}.\textsf{Info}$. The latter will always have the same width as the smaller of the other two, and the bigger width will always be an integer multiple of the smaller one. To deal with these issues, the field $\textsf{Mail}_{k,d}$ for $(k,d) \in Mail\text{-}ind$ will have some additional sub-fields besides the sub-fields defined in Section 9.3. The sub-field **Toaddr** has already been mentioned, but there will also be sub-field **Fromaddr**. The sub-field

$$\textsf{Adj}$$

will show whether the neighbor the direction from which the mail is retrieved is adjacent. Recall the definition of

$$j(k, d), peer(k, d)$$

in Section 9.3: $\textit{Mail}_{k,d}$ of a cell will read, from direction $j(k, d)$, the mail field with index $peer(k, d)$. Recall the definition of $\textit{Mail-to-receive}(k, d)$ in (9.5):

$$\textit{Mail-to-receive}(k, d) = \textit{Mail}_{peer(k,d)}^{j(k,d)}.$$

Now we modify this to account for missing neighbor colonies and to pass the information on whether the mail is coming from an adjacent colony to the bit track $\textit{Mail}_{k,d}.\textit{Adj}$ of the receiving colony.

- If $\textit{Edge}_{j(k,d)} = \infty$ then $\textit{Mail-to-receive}(k, d) = \#$ (no non-adjacent neighbor colony, neighbor big cell will be considered vacant).

- If $\textit{Edge}_{j(k,d)} = 1$ then $\textit{Mail-to-receive}(k, d).\textit{Adj} = 1$ (adjacent neighbor colony).

- If $\textit{Edge}_{j(k,d)} = 1.5$ then $\textit{Mail-to-receive}(k, d).\textit{Adj} = 0$ (non-adjacent neighbor colony).

To deal with asynchrony, we add a "hand-shaking" condition $\textit{Mail-free}_{k,d}(x)$: it says that the cell x is free to rewrite $\textit{Mail}_{k,d}$ with a defined value if the new value is not destined for an earlier address, and either its current value is undefined or if it is for a later or equal address and has already been passed on. (This is sufficient: every piece of mail must be distinct due to ($\textit{Toaddr}$, $\textit{Fromaddr}$).) Let $j = j(k, d)$, where $(k, d) = peer(k, d')$ and note that necessarily $j(k, d) = j(k, d')$. Then

$$
\begin{aligned}
\textit{Mail-free}_{k,d} \Leftrightarrow\ & \text{sign}(\textit{Addr} - \textit{Mail-to-receive}(k, d).\textit{Toaddr}) \neq -j(k, d) \\
& \wedge (\textit{Mail}_{k,d} \in \{\#, \textit{Mail}_{k,d'}^{-j}\} \vee \textit{Mail}_{k,d}.\textit{Toaddr} = \textit{Addr}).
\end{aligned}
\tag{18.1}
$$

Some situations in which mail is used will also need some error-checking, as its output cannot be stored in extra copies for voting. For this, we will check that mail has not gone beyond the cell with $\textit{Addr} = \textit{Toaddr}$ (in the direction $-j$ of mail movement), and that both $\textit{Toaddr}$ and $\textit{Fromaddr}$ are increasing by 0 or 1 from left to right, between a cell and the one to which mail would be forwarded. Here we assume that if $\textit{Mail}_{k,d} = \#$ then $\textit{Mail}_{k,d}.\textit{Toaddr}$, $\textit{Mail}_{k,d}.\textit{Fromaddr}$ and any expressions they enter into have the value $\textit{Undef}$. For $r \in \{-1, 1\}$, $\textit{Mail-legal}_{k,d}(r)$ checks whether mail is legally formed between the cell and its

neighbor in direction r.

$$Mail\text{-}legal_{k,d}(r) \iff \text{sign}(\textsf{Addr} - \textsf{Mail}_{k,d}.\textsf{Toaddr}) \neq r$$
$$\wedge\ \textsf{Mail}^r_{k,d}.\textsf{Toaddr} - \textsf{Mail}_{k,d}.\textsf{Toaddr} \in \{0, r, Undef\}$$
$$\wedge\ \textsf{Mail}^r_{k,d}.\textsf{Fromaddr} - \textsf{Mail}_{k,d}.\textsf{Fromaddr} \in \{0, r, Undef\}.$$

Rule `Move-mail` in Algorithm 18.1 will be more elaborate than its example version in Algorithm 9.5 by checking these two conditions.

Algorithm 18.1: sub-rule `Move-mail`(k, d)

> **let** $j \leftarrow j(k, d)$
> **if not** $Mail\text{-}legal_{k,d}(-j)$ **then** $\textsf{Mail}_{k,d} \leftarrow \#$
> **else if** $Mail\text{-}free_{k,d}$ **and** $Mail\text{-}legal_{k,d}(j)$ **then**
> $\quad \textsf{Mail}_{k,d} \leftarrow Mail\text{-}to\text{-}receive(k, d)$

As $\textsf{Mail}_{k,d}$ has the additional indices $\textsf{F}, \textsf{G}$, eventually we would write `Move-mail`$(k, d, \textsf{F}, \textsf{G})$. The condition $Mail\text{-}legal$ will not let pass any piece of mail whose $\textsf{Toaddr}$, $\textsf{Fromaddr}$ is not gradually non-decreasing from left to right. The replacement with $\#$ eats up the illegal pieces and allows the rest behind it to move. This way the damage can have two effects: one in the place where it occurred, and another, in (or near) the $\textsf{Toaddr}$ where the mail is headed. But it cannot affect several other destinations by substantially changing $\textsf{Toaddr}$.

Mail will be posted by the rule `Post-mail` and received by the rule `Receive-mail`. Both of these will be running for certain intervals of $\textsf{Age}$. We start with defining `Receive-mail`$(k, \textsf{F}, S, a, b, n)$ in Algorithm 18.2, as it is somewhat simpler. Parameter k has the same role as k in $\textsf{Mail}_{k,d}$. Parameter $\textsf{F}$ is the track *into* which the mail is to be received. Parameter S shows how many times the width of this field is a multiple of $\textsf{Mail}_{k,d}.\textsf{Info}$; if $S > 1$ then $\textsf{F}_i$ are the slices with the same width as $\textsf{Mail}_{k,d}.\textsf{Info}$. If $S = 1$ then let $\textsf{F}_0 = \textsf{F}$. Parameters a, b are the starting addresses of the sender and receiver location, and n is the length of the *receiver* location. When the width of the two locations is different then if their intervals could intersect then this would complicate the posting and receiving rules. Therefore we will always arrange that *when the width of the two locations is different then their intervals are disjoint.*

Algorithm 18.2: sub-rule $\texttt{Receive-mail}(k, F, G, S, a, b, n)$

> **if** $k \neq 0$ **then let** $d \leftarrow -1$ **else let** $d \leftarrow \text{sign}(b - a)$
>
> **let** $j \leftarrow j(k, d)$, $A \leftarrow (Mail_{k,d,F,G}^{j}.Fromaddr - a)$
>
> **if** $Addr \in [b, b+n)$ **and** $Addr - b = Toaddr$ **then**
> > $G_{A \bmod S} \leftarrow Mail_{k,d,F,G}^{j}.Info$

In $\texttt{Post-mail}(k, F, G, S, a, b, n)$, Algorithm 18.4, parameters k, a, b have the same role as in $\texttt{Receive-mail}$. Parameter F is the track *from* which the mail is to be sent: its width is S times that of $Mail_{k,d}.Info$, with slices F_i as in $\texttt{Receive-mail}$. Parameter n is the length of the *sender* location: the receiver location has length Sn. It assumes that at its start the $Mail_{k,d}$ track on interval $[a, a+n)$ is covered by #. $\texttt{Post-mail}$ is somewhat more complex than $\texttt{Receive-mail}$, as the order in which the slices F_i are posted depends on the direction in which mail will be sent. For simplicity we first look at $\texttt{Post-mail}(1, F, G, S, a, b.n)$, that is the case for $Mail_{1,1}$: the slices of location $F([a, a+n))$ will be posted starting with address $a + n - 1$, going backwards towards address a, also the slices are posted one-by-one backwards starting from F_{S-1}. The information will just get posted onto the $Mail_{1,1}$ track, relying on $\texttt{Move-mail}$ to forward it to the right. The posting in each cell begins only when $Mail_{1,1} = \#$, and except for the right end-cell, the right neighbor must have already posted all its slices. If a slice F_i with $i > 0$ is currently in $Mail_{1,1}$ then slice F_{i-1} is posted only if the current one has already been forwarded to the right neighbor. When everything has been posted and forwarded from the leftmost cell then it posts #. This will be forwarded by $\texttt{Move-mail}$, and eventually the whole interval $[a, a+n)$ will be covered by #; at this point the posting will start again from the right end. A fault will not make the posting process stuck forever, as $\texttt{Move-mail}$ will keep moving whatever it finds on the mail track. For readability here as well in Algorithm 18.4 below we suppress the indices F, G from $Mail_{k,d,F,G}$.

Case (1) is when the slice of the field F to be posted first can actually be posted. Condition (2) is testing whether the posted value has been passed on (by $\texttt{Move-mail}$): this is similar to part of the condition $Mail\text{-}free_{k,d}$ in (18.1). For the general k, d, recall the direction $j(k, d)$ from which mail is received, defined after (9.4).

Let us estimate the time it takes the posted information to arrive, in a typical special case.

Lemma 18.1 *Consider a case of* $\texttt{Post-mail}(1, F, G, 1, 0, 0, Q)$ *when a track* F *is sent to a track* G *of a right neighbor colony (not necessarily adjacent).*

Algorithm 18.3: sub-rule $\texttt{Post-mail}(1, F, G, S, a, b, n)$

> **let** $Addr' \leftarrow Addr - a$
> **if** $Addr' \in [0, n)$ **then**
>   **if** $Mail_{1,1} = \#$ **then**
> (1)   **if** $Addr' = n - 1$ **or** $Mail^1_{1,1}.Toaddr - b = S(Addr' + 1)$ **then**
> $Mail_{1,1}.(Info, Fromaddr, Toaddr) \leftarrow$
> $(F_{S-1}, Addr, b + S \cdot Addr' + (S - 1))$
> (2) **else if** $Mail_{1,1}.Fromaddr = Addr$ **and** $Mail^1_{1,1} = Mail_{1,1}$ **then**
>   **let** $i \leftarrow (Mail_{1,1}.Toaddr - b) \bmod S$
>   **if** $i \neq 0$ **then** $Mail_{1,1}.(Info, Fromaddr, Toaddr) \leftarrow$
> $(F_{i-1}, Addr, Mail_{1,1}.Toaddr - 1)$
>   **else if** $Addr' = 0$ **then** $Mail_{1,1} \leftarrow \#$

Algorithm 18.4: sub-rule $\texttt{Post-mail}(k, F, G, S, a, b, n)$

> **if** $k \neq 0$ **then let** $d \leftarrow 1$ **else let** $d \leftarrow \text{sign}(b - a)$
> **let** $Addr' \leftarrow Addr - a$, $\delta_j \leftarrow (j + 1)/2$
> **if** $Addr' \in [0, n)$ **then**
>   **if** $Mail_{k,d} = \#$ **then**
>   **if** $Addr' = \delta_{-j}(n - 1)$
>   **or** $Mail^{-j}_{k,d}.Toaddr - b = S(Addr' - j) - \delta_j(S - 1)$ **then**
> $Mail_{k,d}.(Info, Fromaddr, Toaddr) \leftarrow$
> $(F_{\delta_{-j}(S-1)}, Addr, b + S \cdot Addr' + \delta_{-j}(S - 1))$
>   **else if** $Mail_{k,d}.Fromaddr = Addr$ **and** $Mail^{-j}_{k,d} = Mail_{k,d}$ **then**
>   **let** $i \leftarrow (Mail_{k,d}.Toaddr - b) \bmod S$
>   **if** $i \neq \delta_j(S - 1)$ **then** $Mail_{k,d}.(Info, Fromaddr, Toaddr) \leftarrow$
> $(F_{i+j}, Addr, Mail_{k,d}.Toaddr + j)$
>   **else if** $Addr' = \delta_j(n - 1)$ **then** $Mail_{k,d} \leftarrow \#$

Assume that between the sending time t_0 and the receiving time, the space-time area of concern is damage-free. For a cell x let $d(x)$ be the distance of x from the right end of the receiving range. Let x be a cell and y be a cell that is m steps (over neighbors) to the right of x. Then at time $t_0 + 2T^\bullet(m + 2d(x))$ the information from x has reached y.

Proof. Induction on $n = m + 2d(x)$. For $i = 0$ the information did not have to travel any steps. So the statement holds for $i = 0$ and hence also for $n = 0$. Suppose that the statement holds for $n - 1$, we will prove it for n.

Let y' be the left neighbor of y, and let t'_1 be its first switching time after $t_1 - 2T^\bullet$. For y', distance $m' = m - 1$, we have $n' = m' + 2d(x) = n - 1$. By the inductive assumption on n, information from x must have reached y' by the time t'_1. It follows that at the decision time t_1 of y, it must have seen this information.

If $d(y) = 0$ then it has no right neighbor to consider and will receive the information of y' at time t_1. Suppose that it has a right neighbor y'' to consider, and let t''_1 be the first switching time of y'' after $t_1 - 2T^\bullet$. Then y'' is at distance $m'' = m + 1$ to the right of x. Let x'' be the right neighbor of x, then $d(x'') = d(x) - 1$, and for the pair x'', y'' we have $n'' = m'' + 2d(x'') = n - 1$, therefore by time $2T^\bullet(n - 1) = 2T^\bullet n - 2T^\bullet$ the cell y'' has the information from x''. Cell y will see this and hence will be free to receive the information from y' at time t_1 (if it still does not have it). □

The copy command will now be defined with more care than in the example simulation in Algorithm 9.6. Actually, we will only define it for copying within the colony, so from now on we will just write

$$\text{\textsf{Copy}}(loc_1, loc_2).$$

For retrieving information from the neighbor colonies, another mechanism will be used. For the command

$$\text{\textsf{Copy}}(\textsf{F}([a, a + n_1)), \textsf{G}([b, b + n_2,)),$$

we will need only the case when for some $w = |\textsf{Mail}_{k,d}.\textsf{Info}|$ and $S \geq 1$ we have either $|\textsf{F}| = Sw$, $|\textsf{G}| = w$, $n_2 = Sn_1$ or $|\textsf{G}| = Sw$, $|\textsf{F}| = w$, $n_1 = Sn_2$. In the implementation here, we simply run the rules `Post-mail` and `Receive-mail` for a certain length of time. If $S \neq 1$ then we require here the two intervals $[a, a + n_1)$, $[b, b + n_2)$ to be disjoint. In applications we will not insist on this condition because if the intervals are not disjoint we could replace one copy operation with two that satisfy the condition.

Two kinds of information will be retrieved before the computation. The information that is found within the colony itself will be retrieved using the rule `Copy` of Algorithm 18.5. Retrieving from the neighbor colonies is more complex, as these colonies may not be there or may be in different stages of their work period. It will be handled in Section 19.2.

Algorithm 18.5: sub-rule $\mathit{Copy}(F([a, a + n_1)), G([b, b + n_2)))$

> **let** $d \leftarrow \text{sign}(b - a)$
> **if** $n_1 > n_2$ **then let** $S_1 \leftarrow n_1/n_2,\ S_2 \leftarrow 1$
> **else let** $S_1 \leftarrow 1,\ S_2 \leftarrow n_2/n_1$
> $\mathit{Mail}_{0,d,F,G} \leftarrow \#$
> **for** λSQ steps of Age **do**
> > $\mathtt{Post\text{-}mail}(0, F, G, S_1, a, b, n_1)$
> > $\mathtt{Move\text{-}mail}(0, d, F, G)$
> > $\mathtt{Receive\text{-}mail}(0, F, G, S_2, a, b, n_2)$

19   Computation

Before defining the rules that carry out the computation of a colony, we define some of the simple data structures that support it.

19.1   Coding and decoding

Let us say more about the error-correcting code used.

Definition 19.1 (Errors) Our error-correcting code will distinguish, similarly to Examples 5.14 and 5.16, information symbols and error-check symbols. 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. We will work with a code

$$\alpha \tag{19.1}$$

of Example 5.15 that can correct errors in t symbols with just $2t$ error-check symbols. 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. We expect having to deal with at most one damage rectangle. It can corrupt a group of Δ consecutive cells (where Δ was defined in (13.4)). The symbols of the code will be chosen in such a way that a code symbol stretches over Δ cells. A damage rectangle can damage therefore up to 2 symbols, for which 4 error check symbols suffice. ⌟

In the subdivision of the state of our simulation cells into fields, we develop here a more elaborate version of Example 5.16. Like there, we have a field *Work*, to be subdivided into a constant number of fields like *Addr*, *Mail*, and

so on. In Definition 12.1, we introduced a field called *Info* partitioned into sub-fields *Payload*, *Pl-redun* and *Util*. We will build a uniform amplifier as in Lemma 11.7, with an aggregated field *Payload*. Different parts of the *Util* track in the colony will represent the fields *Work**, *Pl-redun**, *Color** and *Util** of the simulated big cell, and contain also the error check symbols for the *Util* track itself. Let

$$\textit{Payload}' = \textit{Payload} \cup \textit{Pl-redun}$$

denote the payload along with its error-check symbols. Just as in Example 5.16, all fields other than *Payload* have relative width $O(w)$; in particular, *Pl-redun* has relative width w. With w_k chosen as in (11.9), this allows the capacity Cap_1 of the base medium M_1 to still be constant, as the sum over all levels k of the relative widths of all non-aggregated fields is bounded.

Our transition function Tr will have bandwidth rate $w = w_k$, as in (11.6), and so the simulated transition function Tr^* has bandwidth rate $w^* = w_{k+1}$. This way the amount of information needed from the neighbor colonies will be small. Small bandwidth requires, however, another complication. All information processing will happen on the *Work* track, which has relative width $O(w)$. As at any one time, this track can only handle a small portion of the the *Payload* track of the colony, we introduce the notion of *packets*.

Definition 19.2 (Packets) As the payload of a simulated cell is the aggregated payload of the cells of its simulated colony, we define the *payload capacity* as

$$Cap'_k = |\textit{Payload}_k| = B_k|\textit{Payload}_1|.$$

We partition the *Payload* field into P sub-fields of relative width w called *field packets* numbered $0, 1, \dots, P - 1$. The ith field packet of the *Payload* field will be denoted *Payload*$_i$. We also partition each track *F* of whole colony into P^* *track packets* of w^*Q cells each. The jth track packet, with addresses in $[jw^*Q, (j + 1)w^*Q)$, will be denoted as the *location* $_F_j$, In particular the jth track packet of the *Payload* track is location $_{\textit{Payload}_j}$ ⌟

Definition 19.3 (Payload error checks) The error checks of track packet $_{\textit{Payload}_j}$, will be in location $_{\textit{Pl-redun}_j}$ on the *Pl-redun* track. Location

$$_{\textit{Payload}'_j} = {}_{\textit{Payload}_j} \cup {}_{\textit{Pl-redun}_j} \tag{19.2}$$

combines location $_{\textit{Payload}_j}$ with its error check symbols for a complete code.

We consider the whole *Payload* track of each consecutive Δ cells in the packet $_{\textit{Payload}_j}$ as an information symbol; the error check symbols of $_{\textit{Payload}_j}$
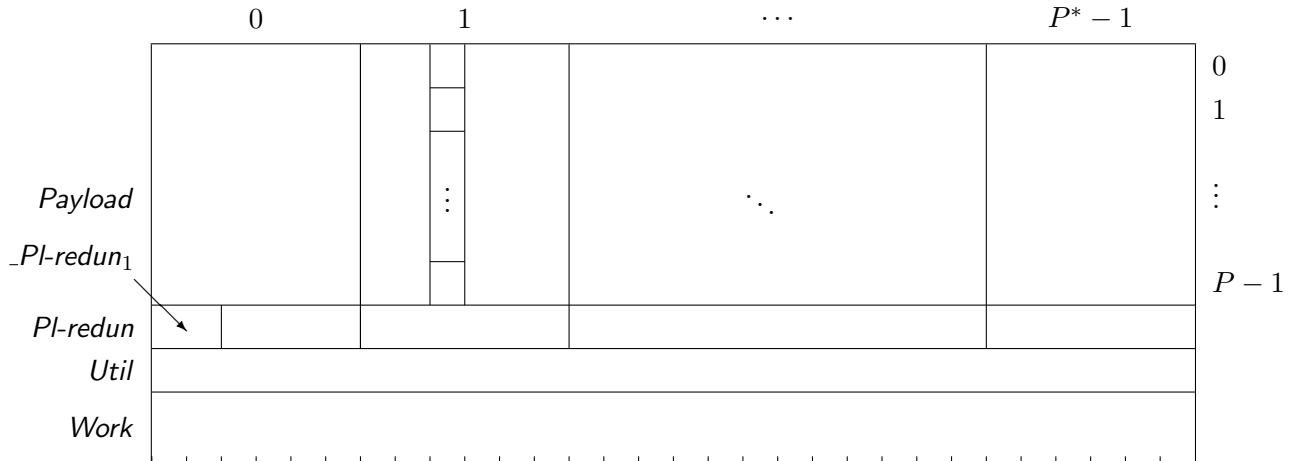
Figure 11: Subdivision of the payload into field packets (horizontal lines) and track packets (vertical lines).

go to location $\_Pl\text{-}redun_j$. To avoid that the damage rectangle affect both an information symbol and an error-check symbol of the same code, we divide the track packet $\_Payload'_j$ into left and right halves. The error checks for the left half of $\_Payload_j$ go into the right half of $\_Pl\text{-}redun_j$, and those for the right half go into the left half of the same. ⌟

The error check symbols in each half of the track packet $\_Pl\text{-}redun_j$ have 4Δ error check symbols of size $< Cap$. The number of bits in $\_Pl\text{-}redun_{j'}$ is Qw^*wCap so these error checks will fit in, according to condition (11.15) on amplifier our parameters.

The leftmost and rightmost packet of the (encoded) payload will be displayed to neighbor colonies as tracks

$$Payload\text{-}to\text{-}nb_j, \; j = \pm 1 \tag{19.3}$$

of width w; these displays will only be updated in the last step of the work period.

The code of the amplifier   Let us define the code and decoding φ_*, Φ^* used in the simulation Φ taking medium $M = M_k$ into medium $M^* = M_{k+1}$.

Definition 19.4 (Encoding) The encoding

$$\varphi_* : \mathbb{S}_{k+1} \to \mathbb{S}_k^{Q_k}$$

takes the state of a cell, applies to it the error-correcting code α_* introduced above in Definition 19.1 and copies the result to the *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. ⌋

There belongs a decoding function $\varphi^* : \mathbb{S}_k^{Q_k} \to \mathbb{S}_{k+1}$ to this encoding, but the actual decoding $\Phi^* : \eta \mapsto \eta^*$ depends on the history, not only on the current configuration $\eta(\cdot, t)$. One reason is that the transition of a colony from one work period to the next one takes some time, due to asynchrony. But we can also use the looking-back to avoid dependency on small noise in the near past.

Definition 19.5 (Decoding) If the (x, t) is not in *Damage**, then define

$$\eta^*(x, t) = \varphi^*(x, t) = \mathit{Vac},$$

unless there is a latest time $t' \in (t - 2QT^\bullet, t]$ that is a switching time of a cell of $\mathcal{C}(x)$ such that the rectangle

$$[x - (Q + 1.1)B, x + (2Q + 1.1B)) \times (t' - \mathsf{split\text{-}t}, t']$$

is damage-free, and the colony is covered by a domain of cells belonging to the same work period (the condition on switching just makes sure that a "latest" value exists). In the latter case, let us define the strings $s = \eta(\mathcal{C}(x), t').\mathit{Util}$ and $p_i = \eta(\mathcal{C}(x), t')._{-}\mathit{Payload}'_i$. For the decoded value to be nonvacant, we also require that each of s, p_i contains at most 4 errors in the sense that $\alpha_*(\alpha^*(s))$ differs from s in at most 4 places of up to 2 consecutive symbols) and the same for each p_i for $i = 0, \ldots, P^* - 1$. If this is satisfied, then $\eta^*(x, t).\mathit{Payload}^c = \alpha^*(s)$ while $\eta^*(x, t).\mathit{Payload}$ is the concatenation of all $\alpha^*(p_i)$. We will say that $\eta(\mathcal{C}(x), t)$ is *error-free* if each of s, p_i are error-free. ⌋

Note that if this definition gave a nonvacant, non-bad value to $\eta^*(x, t)$, then it gave the same value to all $\eta^*(x, u)$ for $u \in [t', t]$ where the time t' was defined above.

Rules for coding and decoding Before introducing the coding and decoding rules, a few more notions are needed. In each rule of the present section, the operation

$$\mathrm{Maj}_{i=0}^2 s_i$$

when applied to strings s_0, s_1, s_2 is the result of taking the bit-wise majority of these three strings. The rule

$$\mathtt{Broadcast}(loc, \mathsf{F}(I)) \tag{19.4}$$

takes the information found in location loc and writes it into the F field of every cell in interval I. The default for I is the whole colony $[0, Q)$.

We will use a general type of rule

$$\mathtt{Check}_0(prop, \mathsf{F}(I), X_1, X_2, \dots) \tag{19.5}$$

checking some global property of some parameters $X_1, X_2, \dots$ each of which is either an explicit string or a location. Here it is assumed that $prop$ 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 location $\mathsf{F}(I)$ (with the help of the rule $\mathtt{Broadcast}$ given above).

The coding and decoding functions can be computed on the universal medium simulated on one of the tracks. Recall rules 9.6 ($\mathtt{Copy}$) and 9.2 ($\mathtt{Write}$) defined in Section 9.3 and again in Section 18.

Definition 19.6 Let

$$Vacant\text{-}str$$

be a string representing the state Vac in our code α as in Definition 19.1. Let $Decode\text{-}prog$ be a program for the medium $Univ$ such that after applying

$$Univ(Decode\text{-}prog \sqcup S; Q, 2Q),$$

location $\_Decode\text{-}output$ on track $\mathsf{Cpt.Output}$ contains $\alpha^*(S)$ (see (19.1)) if $\alpha_*(\alpha^*(S))$ differs from S in at most 4 groups of 2 consecutive symbols each; otherwise, it contains $Vacant\text{-}str$. ⌋

Rule 19.1 ($\mathtt{Decode}$) takes a string from location loc_1, decodes it and copies the result to location loc_2. It will always give a result, even if its input in loc_1 is not a code word. (By the assumption made in Definition 9.2, $Univ$ is commutative, so we do not have to worry about the order of execution in different cells.)

The rule $\mathtt{Encode}(loc_1, loc_2)$ performs encoding in a similar way. It will always output a code word; so applying first $\mathtt{Decode}$ and then $\mathtt{Encode}$ turns every word into a code word. The locations loc_i in the encoding and decoding rules will be allowed to be given indirectly, as explained in Definition 9.17.

In view of Condition 10.16c), for certain information X retrieved from the neighbors we want to make sure that it has the proper form of a code-word.

Algorithm 19.1: sub-rule $Decode(loc_1, loc_2)$

> $Cpt.Input \leftarrow *$;
> $Write(Decode\text{-}prog, \_Prog)$;
> $Copy(loc_1, \_Decode\text{-}arg)$;
> apply $2Q$ times the rule $Univ$ to track Cpt;
> $Copy(\_Decode\text{-}output, loc_2)$

Decoding and then encoding it would do this in the absence of the damage rectangle. The rule $Legalize$ in Algorithm 19.2 will make sure that this happens even in the presence of a damage rectangle. It repeats the decoding-encoding three times (using a temporary location $\_Decoded$) and then takes the majority of the results. Moreover, then repeats the whole procedure again.

Algorithm 19.2: sub-rule $Legalize(F(I))$

> **repeat** 2 **times**
>   **for** $j = 0$ **to** 2 **do**
> $Decode(F(I), \_Decoded)$;
> $Encode(\_Decoded, Vote_j(I))$
>   **if** $Addr \in I$ **then** $F \leftarrow \mathrm{Maj}_{j=0}^2 Vote_j$

Lemma 19.7 *After an application of the rule* $Legalize(F(I))$, *location* $F(I)$ *contains a word with at most 2 errors. Moreover, if* $F(I)$ *has at most 2 errors and decodes to some value* X *then result also decodes to* X.

Proof. Suppose first that $F(I)$ has at most 2 errors and decodes to X. Consider any one of the two repetitions; the decoding-encoding writes $Vote_j(I)$. If damage happens during it then the result is worthless. But otherwise, earlier damage could add at most 2 more errors, and since $Decode$ corrects 4 errors, the result in $Vote_j$ is the code of X. Therefore the result in the first repetition differs from the code of X in the at most 2 errors added possibly in the last majority step. The same reasoning applies to the second repetition.

Suppose now that $F(I)$ is an arbitrary word. Suppose that damage does not occur in the first repetition. Then this part turns $F(I)$ into a code word (without errors), and the previous analysis applies to the second repetition. If damage occurs in the first repetition then $F(I)$ may be again arbitrary as the result of this part. But then the second repetition is damage-free and writes a code word into $F(I)$. $\qquad\square$

19.2 Sending and retrieval

In order to satisfy Condition 10.16e), it is necessary to find a time when the information received from both neighbor colonies can be attributed to the same moment. For this, the communication part of the work period will be substantially longer than the rest. During this time, the colony will continually post the information destined for the neighbor colonies. One of these pieces of information will be the age (the age of one cell would be sufficient, but for simplicity and safety, all cells will send their age). On the other hand, the receiving colony keeps checking (using the sender's age) whether the sender's age of *both* neighbors is from a safe part of their work period: if yes then it stops receiving (by setting *Receiving* $\leftarrow 0$), and keeps the information already has to be used later by `Compute`.

Recall the colony work period in Algorithm 12.1. The rule `Send`, running almost all the time, will try to mail information to the neighbor colonies, running the rules `Post-mail` listed in Algorithm 19.3 simultaneously. The retrieval part is played by two more rules running simultaneously. The rule `Extend` of Algorithm 14.9 tries to extend some arms of the colony (consisting of cells of kind $Channel_j$) left and right, for use in communicating with a possible non-adjacent neighbor colony. The rule `Retrieve` in Algorithm 19.4 (which is very different from the example in 9.4) will try to retrieve information sent by the neighbors this way. It records in track Age_k, etc. the *Age*, *Doomed* and *New* fields sent from the neighbor colonies in direction k (see below the explanation for *New*). For example Age_{-1} at address a is expecting the *Age* of cell at address a of the left neighbor colony. As the neighbor keeps sending this information, the receiving cell keeps updating it.

The main fields of the represented big cell of each colony are encoded onto its *Util* track. This entire track of the left neighbor will be sent to the $Retrieved_{-1}$ track of the receiving colony. The slices of $Payload_i$ of the represented cell ($i = 0, \ldots, P^* - 1$) are encoded into the locations $\_Payload'_i$ of the colony.

Algorithm 19.3: rule `Send`

> **if** $Age <$ grow-start **then**
>> **for** $k \in \{-1, 1\}$ **do**
>>> `Post-mail`$(k, Age, Age_{-k}, 1, 0, 0, Q)$
>>> `Post-mail`$(k, Doomed, Doomed_{-k}, 1, 0, 0, Q)$
>>> `Post-mail`$(k, New, New_{-k}, 1, 0, 0, Q)$
>>> `Post-mail`$(k, Util, Retrieved_{-k}, 1, 0, 0, Q)$

The `Send` rule of Algorithm 19.3 will run most of the time, and each `Post-mail` in it will keep sending its information over and over. At the beginning of the work period the field

$$Receiving$$

gets its default value 1, and the

$$AdjCol_k, k = \pm 1$$

field, indicating whether there is an adjacent neighbor colony in direction k, gets the default value 0. The `Retrieve` rule in Algorithm 19.4 will set $Receiving \leftarrow 0$ as soon as it finds that the information from both directions is *safe*: it has not changed "recently" and will not change "soon" (precise definition is in (19.6)). When *Age* is safe any information sent from any other cell of the colony must have already been from the same work period, and the updating of payload has not yet been started—so all payload information sent from this colony is consistent. Here we use the fact from Lemma 18.1 that the sending process itself is reasonably fast, happening in $O(Q)$ time. Let

$$Safe = \{\, Undef \,\} \cup (K \cdot Q, \text{proc-payload-start} - K \cdot Q], \qquad (19.6)$$

where K is the constant started to be used in (12.3). Information from direction k has not changed lately and will not change soon if $Age_k \in Safe$. Indeed, if $Age_k = Undef$ then for a while before, the neighbor colony in direction k was either vacant or consisted of doomed cells, and for a while after, if it is not vacant or doomed then its *New* track is filled with 1's—in all these cases the neighbor big cell in direction k will be treated as vacant.

The definitions in (12.3) show that, with R and K large enough the overwhelming majority of the work period is safe. Indeed, the safe part has age length $KQ(2RP^* - 1)$, hence has a time interval of length at least

$$KQ(2RP^* - 3)T_\bullet \geq 1.5KRP^*QT_\bullet \qquad (19.7)$$

in which all age is safe. Unsafe are only parts outside it, in up to two time intervals of total age length $Q(RP^* + 6\lambda + 4 + K)$, and hence total time length $\leq 1.5RP^*QT^\bullet$ if R is large.

The `Retrieve` rule uses the *Adj* field of the one of the mails (say the one sending the age information) to set $AdjCol_k$. The bits *Doomed*, *New*, $AdjCol_k$ would normally be the same over the whole colony, and would differ from the common value only at the places directly affected by damage (the damage may have happened elsewhere, with the mail carrying the change to its destination).

Algorithm 19.4: rule `Retrieve`

> **if** $Receiving = 1$ **then**
>> **for** $k \in \{-1, 1\}$ **do**
>>> $\texttt{Receive-mail}(k, Age, Age_{-k}, 1, 0, 0, Q)$
>>> $\texttt{Receive-mail}(k, Doomed, Doomed_{-k}, 1, 0, 0, Q)$
>>> $\texttt{Receive-mail}(k, Util, Retrieved_{-k}, 1, 0, 0, Q)$
>>> $\texttt{Receive-mail}(k, Payload\text{-}to\text{-}nb_{k}, Payload\text{-}from\text{-}nb_{-k}, 1, 0, 0, Q)$
>>> **if** $Doomed_{k} = 1$ **or** $(New_{k} = 1$ **and** $Age_{k} < K \cdot Q)$ **then**
>>>> $Retrieved_{-k} \leftarrow Vacant\text{-}str(Addr)$
>>>
>>> $AdjCol_{-k} \leftarrow Mail_{k, -1, Age, Age_{-k}}.Adj$
>>
>> **if** $\{Age_{-1}, Age_{1}\} \subset Safe$ **then** $Receiving \leftarrow 0$

19.3 Computation rules

Recall that we are proving Lemma 11.7 (Amplifier) by building the transition function of the medium M_k. Given the amplifier parameters and the number k denoting the level, let us fix where the frame information is to be found, using the scheme with parameteres introduced in Definition 9.14. (The rule evaluating the transition function assumes that the collection of amplifier parameters are described in a parameter called *Frame*, the parameter k is described in parameter called *Height*.

Recall also the structure of the colony work period in algorithm 12.1. Here we will describe the `Compute`, `Proc-payload` and `Finish` rules. The main simulation will interpret the program that needs to be applied to the represented states of the big cell and its neighbors. Given that all computation happens on the narrow *Work* track, the rules processing the payload rely extensively on the copying rule of Algorithm 18.5. The copy process, by its simplicity, is fault-tolerant: the damage rectangle can only locally corrupt the information processed by it.

By Theorem 9.2, our rule language can be interpreted by a cellular automaton on some work track. In our example simulation in Section 9.3 the evaluation rule `Eval` worked on a track was named *Cpt*, with its result on the track *Cpt.Output*. Now some post-processing will also follow, but let us discuss first the input information to the rule, some of which will be retrieved from neighbor colonies. That retrieval is defined in Section 19.2. According to the requirement on amplifiers, the transition function *Tr* carries out the

computational part *Pl-trans* via Definition 11.4. Denote by

$$\textit{Payload}^c = \textit{All} \setminus \textit{Payload}$$

the complement of the payload field. As arguments to the transition function, let $\mathbf{r} = (r_{-1}, r_0, r_1)$, and $\mathbf{a} = (a_{-1}, a_1)$: the latter are the bits showing whether the left and right neighbors are adjacent. Recall the notations in Example 4.21. From $\mathbf{r}$, the transition function $Tr(\mathbf{r}, \mathbf{a})$ will depend only on

$$r_0, \; r_j.\textit{Payload}^c, \; r_j.\textit{Payload}.\textit{slice}^{w,-j}, \; j = -1, 1.$$

Thus, from the *Payload* of the neighbors only the last segment from the left neighbor and the first one from the right neighbor is needed. This way, when simulating Tr^* by a colony then due to (11.14), all needed information from the neighbor colonies for computing $Tr^*(\mathbf{r}, \mathbf{a})$ will fit onto the *Work* track of the computing colony. Indeed, for $j = -1, 1$, for $r_j.\textit{Payload}^c$ one needs only the *Util* track of the neighbor colony in direction j. For $r_{-1}.\textit{Payload}.\textit{slice}^{w^*,1}$ one needs only the information in location $\_\textit{Payload}'_{P^*-1}$ of the left neighbor colony as defined in (19.2), and for $r_1.\textit{Payload}.\textit{slice}^{w^*,-1}$ one needs $\_\textit{Payload}'_0$ from the right neighbor colony. (The part of the simulating program that needs the latter information is the rule `Update-payload` as defined in Algorithm 19.10.)

   After retrieval, as described in Section 19.2, the encoded argument $r_j.\textit{Payload}^c$, $j = \pm 1$ for the simulation will be on track *Retrieved*$_j$, while for $r_0.\textit{Payload}^c$ it is on track *Util*. After decoding, the results will appear in some locations $\_\textit{Arg}_m$ for $m \in \{-1, 0, 1\}$. The bits a_{-1}, a_1 can be determined directly from tracks *AdjCol*$_j$, as the `Retrieve` rule of Algorithm 19.4 has set them. (Each of these two tracks contains the repetition of the same single bit, except for the places affected by damage, so decoding can just take the majority.)

   The rule `Eval` of Section 9.3 computes the transition function on the *Cpt* track, using the program *Interpr* similarly to Algorithm 9.7 in Section 9.3. It uses the locations $\_\textit{Arg}_m$ and sub-rule `Initialize` mentioned in (9.6). But in the current version, in Algorithm 19.5, there are two more arguments, in locations $\_\textit{Adj-arg}_j$ for $j = \pm 1$, for the bits a_{-1}, a_1 in $Tr^*(\mathbf{r}, \mathbf{a})$.

   The modification also accounts for payload processing. The program *Interpr* for the universal cellular automaton, used in the rule `Interpret`, will now be enhanced as follows, using some strings, small in size, called *symbolic commands*, from the following list.

$$\text{Update-payload, Refresh-payload,} \tag{19.8}$$
$$\text{Mail}_{k,d}.\text{Info} \leftarrow \text{Payload}_i, \; \text{Payload}_i \leftarrow \text{Mail}_{k,d}.\text{Info},$$

where $i \in \{0, \ldots, P^* - 1\}$ and k, d are numbers, and the role of the field *Mail*$_{k,d}$ is explained after (9.3). When a condition requires it will *broadcast* them (using the rule `Broadcast`(x, F) of (19.4)): write them into the field *Cpt.Output.Pl-commands* of *each cell*. In particular:

- Encountering the command *Payload*$_i \leftarrow$ *Mail*$_{k,d}$.*Info*
  in the program for the big cell, broadcasts `Payload`$_i \leftarrow$ `Mail`$_{k,d}$.`Info` onto track *Cpt.Output.Pl-commands* while on command *Mail*$_{k,d}$.*Info* $\leftarrow$ *Payload*$_i$ it broadcasts `Mail`$_{k,d}$.`Info` $\leftarrow$ `Payload`$_i$.

The broadcast symbolic commands will be executed by the rule `Proc-payload`—eventually, after they have been written onto a locally maintained track *Pl-commands*.

Algorithm 19.5: sub-rule `Eval`

> `Write`$(Interpr, \_Interpr)$;
> `Write-param`$(My\text{-}rules, \_Prog)$;
> **for** $i = 1$ **to** N **do** `Write-param`$(Param_i, \_Param_i)$
> `Initialize`;
> `Interpret`

To deal with the possible damage rectangle, rule `Compute` of Algorithm 19.7 will call `Eval` three times, storing its output on the track *Vote*$_r$ for $r = 0, 1, 2$. The track *Hold* will receive then the majority of these results. A post-processing step `Update-loc-maint` will update the locally-maintained fields

$$Doomed, Growing_j \ (j \in \{-1, 1\}), Pl\text{-}commands.$$

It checks via rule `Check`$_0(Check\text{-}vacant, F(I), loc)$ whether the string represented in location *loc* is *Vac** (and broadcasts the result into track *F* on interval I). Let

$$\_Creating_j$$

be the location of the field *Creating*$_j^*$ of the represented cell on the *Cpt.Output.Util* track. If the represented cell becomes vacant then, as expected, *Creating*$_j^* = 0$. (This is how we will satisfy Condition d) in simulation, as Definition 10.12 requires creator to survive until after the creation.) The value found here will be broadcast to the locally maintained track *Growing*$_j$, since when the represented cell is allowed to be creating, say, to the right, then the representing colony should be allowed to grow to the right in an attempt to create an adjacent new colony to the right. As usual, the argument *F* means $F([0, Q))$.

Algorithm 19.6: sub-rule $\mathtt{Update\text{-}loc\text{-}maint}(r), r = 0, 1, 2$

> $\mathtt{Check}_0(Check\text{-}vacant, \textbf{\textit{Vote}}_r.\textit{Util}, \textbf{\textit{Vote}}_r.\textit{Doomed});$
> **pfor** $j = -1, 1$ **do** $\mathtt{Broadcast}(\textit{\_Creating}_j, \textbf{\textit{Vote}}_r.\textit{Growing}_j)$

The rule $\mathtt{Compute}$ repeats the evaluation process 3 times: after repetition r, we encode the result into $\textbf{\textit{Cpt.Output}}$ and copy it onto track $\textbf{\textit{Vote}}_r$. Finally $\textit{Hold}$ will be obtained by majority vote from $\textbf{\textit{Vote}}_r$, $r = 0, 1, 2$. Recall that $\textbf{\textit{Cpt.Output}}$ has sub-tracks $\textit{Util}$ and $\textit{Pl-commands}$. Only the sub-track $\textit{Util}$ needs encoding, the track $\textit{Pl-commands}$ contains the same value repeated in each cell. Rule $\mathtt{Compute}$ calls some more rules. The sub-rule $\mathtt{Randomize}$ takes the $\textit{Rand}$ bit of the first cell of the colony, and places its α_*-code into the appropriate location on the $\textit{Hold}$ track. This is done only once, without any attempt of error-correction. The line (1) tests whether the colony encodes a germ big cell that expects to become part of a (big) colony. It is interpreted easily, when we stipulate that $\textit{Kind}^*$ and $\textit{Addr}^*$ occupy a special location on the $\textit{Util}$ track. The last lines of the code of $\mathtt{Compute}$ update the locally maintained variables $\textit{Doomed}$, $\textit{Growing}_j$ and $\textit{Pl-commands}$ in a single step, so this step is their update time.

Algorithm 19.7: sub-rule $\mathtt{Compute}$

> **for** $m \in \{-1, 1\}$ **do** $\mathtt{Legalize}(\textit{\_Retrieved}_m)$
> $\mathtt{Legalize}(\textit{Util});$
> **for** $r = 0$ to 2 **do**
> > $\mathtt{Decode}(\textit{Util}, \textit{\_Arg}_0)$;
> > **for** $m \in \{-1, 1\}$ **do**
> > > $\mathtt{Decode}(\textit{\_Retrieved}_m, \textit{\_Arg}_m);$
> > > $\mathtt{Decode}(\textit{AdjCol}_m, \textit{\_Adj\text{-}arg}_m)$
> >
> > $\mathtt{Eval};$
> > $\mathtt{Encode}(\textbf{\textit{Cpt.Output}}, \textbf{\textit{Vote}}_r.\textit{Util});$
> > $\mathtt{Update\text{-}loc\text{-}maint}(r)$
>
> (1)   **if** $\textit{Kind}^* = Germ$ **and** $0 \le \textit{Addr}^* < Q^*$ **then**
> > $\mathtt{Broadcast}(\mathtt{Update\text{-}payload}, \textbf{\textit{Cpt.Vote}}_r.\textit{Pl-commands})$
> > **else** $\mathtt{Broadcast}(\mathtt{Refresh\text{-}payload}, \textbf{\textit{Vote}}_r.\textit{Pl-commands})$
>
> $\mathtt{Randomize};$
> $\textit{Hold} \leftarrow \text{Maj}_{r=0}^2 \textbf{\textit{Vote}}_r;$
> $(\textit{Doomed}, \textit{Pl-commands}) \leftarrow (\textit{Hold.Doomed}, \textit{Hold.Pl-commands})$
> **pfor** $j = -1, 1$ **do** $\textit{Growing}_j \leftarrow \textit{Hold.Growing}_j$

If we had to do with a single cell and not a represented one then no payload refreshing would be needed, but the rule `Refresh-payload` (interpreting the corresponding symbolic command), as it applies to a represented cell, will perform error-correction. It may seem at first sight that some of these commands might contradict each other. However, when the symbolic command `Update-payload` is outputted then it stands alone; indeed, this is done when a germ cell is represented, and germ cells are not part of any colony performing simulation. Similarly, `Refresh-payload` may seem to possibly contradict the interpretation of the symbolic command `Payload`$_i$←`Mail`$_{k,d}$.`Info`; however, as they are both are trying to keep (or transport) the same string, only some new fault can cause a (local) contradiction.

The rule `Refresh-payload` described in Algorithm 19.8 uses the location *_To-refresh* on the *To-refresh* track of the work tape to which the packets of *Payload* will be spread out for processing. To control for the occurrence of possible damage during processing, each packet is processed three times for $r = 0, 1, 2$: after decoding and encoding, the result is stored in a location *_Vote*$_j$ on a track *Vote*$_j$. Then the majority is copied back to the packet.

Algorithm 19.8: sub-rule `Refresh-payload`

> **for** $i = 0$ **to** $P^* - 1$ **do**
> $\quad$ *Copy*(*_Payload*$'_i$, *_To-refresh*);
> $\quad$ **for** $r = 0$ **to** 2 **do**
> $\quad\quad$ *Decode*(*_To-refresh*, *_Decoded*);
> $\quad\quad$ *Encode*(*_Decoded*, *_Vote*$_r$)
> $\quad$ *To-refresh* $\leftarrow$ $\mathrm{Maj}_{r=0}^{2}$ *Vote*$_r$;
> $\quad$ *Copy*(*_To-refresh*, *_Payload*$'_i$)

The computation of *Pl-trans*$^* =$ *Pl-trans*$_{k+1}$ in rule `Update-payload` as shown in Algorithm 19.10, takes place when *Kind*$^* = Germ$ and the represented germ cell can expect to become part of a colony (so $0 \leq$ *Addr*$^* < Q^*$). This level is the highest, and as will be seen the possibility of damage on this level can be ignored, so the rule `Update-packet` in Algorithm 19.9 does not need the vote over three repetitions used elsewhere.

Recall that *Pl-trans*$_{k+1} =$ *Pl-trans*$_1^{B_{k+1}, w_{k+1}}$ is an aggregated and slowed version of *Pl-trans*$_1$. We compute it part-by-part, as we can only use the narrow track *Work*. So we will copy, starting from the left, new and new packets to the *Work* track, perform the updating using three neighboring ones, and copy back the result. The actual simulation of *Pl-trans*$_1$ will take place on a sub-track called *Pl-upd*, of width $w \cdot Cap$. Recall from Definition 19.2 that the

capacity of the medium $Pl\text{-}trans_1$ is Cap_1'. After copying and decoding, we will have on the **Pl-upd** track three consecutive locations of the same length: $\_Pl\text{-}upd_p$ for $p = -1, 0, 1$, A packet $\_Payload_i$ will be distributed in this location. The amount of information in $\_Payload_i$ is $w^*Q \cdot Cap$, so we need

$$w^*Q \cdot Cap/w \cdot Cap = w^*QP \qquad (19.9)$$

cells as the width of the track is $w \cdot Cap.$. Each cell in this location will contain

$$S = w \cdot Cap/Cap_1'$$

symbols of $Pl\text{-}trans_1$, so this is an aggregation code, and each cell can compute on this track the aggregated transition function $Pl\text{-}trans_1^S$ in one step. The number of steps is equal to the number of cells in $\_Pl\text{-}upd_0$, as in (19.9). At the end, only the result in $\_Pl\text{-}upd_0$ will be used. Decoding, updating and encoding will be repeated three times just as in the refreshing case. The rule `Update-packet` assumes that the needed three packets have already been copied to the consecutive locations $\_To\text{-}update_p$ for $p = -1, 0, 1$, and writes its result into location $\_Updated$. In rule `Update-payload`, in order to update the first and last package, the last package of the left and the first package of the right neighbor colonies are needed; but they have already been retrieved by `Retrieve` into $Payload\text{-}from\text{-}nb_j$, $j = \pm 1$.

Algorithm 19.9: sub-rule `Update-packet`

> **for** $p = -1, 0, 1$ **do**
> > $Decode(\_To\text{-}update_p, \_Pl\text{-}upd_p)$
>
> **repeat** w^*QP **times**
> > $Pl\text{-}upd \leftarrow Pl\text{-}trans_1^S(Pl\text{-}upd^{-1}, Pl\text{-}upd, Pl\text{-}upd^1)$
>
> $Encode(\_Pl\text{-}upd_0, \_Updated)$

Algorithm 19.11 (`Proc-payload`) depends on the locally maintained track **Pl-commands**. We will analyze later what happens when it is damaged. (Only the command `Update-payload` is not controlled for damage; however, as it is called only on the highest level, we will be able to assume no damage on that level.) Location $\_Mail_{k,d}^*.Info$ is, just as all fields of the represented cell, on the **Util** track. In line (1), $i = 0$ if $k = -1$ and $P - 1$ if $k = 1$. As said above, now this program outputs not just the candidate values of the fields in $All^* \setminus Payload^*$ onto $\_Cpt.Output.Util$ but also a couple of symbolic commands from the list (19.8) onto $\_Cpt.Output.Pl\text{-}commands$. In particular, every time

Algorithm 19.10: sub-rule `Update-payload`

    $\textit{Copy}(\textit{Payload-from-nb}_{-1}, \_\textit{To-update}_{-1})$;
    $\textit{Copy}(\_\textit{Payload}'_0, \_\textit{To-update}_0)$;
    $\textit{Copy}(\_\textit{Payload}'_1, \_\textit{To-update}_1)$;
    `Update-packet`;
    $\textit{Copy}(\_\textit{Updated}, \_\textit{Payload}'_0)$;
    $\textit{Copy}(\_\textit{To-update}_0, \_\textit{To-update}_{-1})$;
    $\textit{Copy}(\_\textit{To-update}_1, \_\textit{To-update}_0)$;
    for $i = 1$ **to** $P^* - 2$ **do**
        $\textit{Copy}(\_\textit{Payload}'_{i+1}, \_\textit{To-update}_1)$;
        `Update-packet`;
        $\textit{Copy}(\_\textit{Updated}, \_\textit{Payload}'_i)$;
        $\textit{Copy}(\_\textit{To-update}_0, \_\textit{To-update}_{-1})$;
        $\textit{Copy}(\_\textit{To-update}_1, \_\textit{To-update}_0)$;
    $\textit{Copy}(\textit{Payload-from-nb}_1, \_\textit{To-update}_1)$;
    `Update-packet`;
    $\textit{Copy}(\_\textit{Updated}, \_\textit{Payload}'_{P^*-1})$

Algorithm 19.11: sub-rule `Proc-payload`

    if $\textit{Pl-commands}$ contains `Update-payload` **then** `Update-payload`
    if $\textit{Pl-commands}$ contains `Refresh-payload` **then** `Refresh-payload`
    if $\textit{Pl-commands}$ contains $\texttt{Mail}_{k,d}.\texttt{Info}{\leftarrow}\texttt{Payload}_i$ for some k, d, i
      then $\textit{Copy}(\_\textit{Payload}'_i, \_\textit{Mail}^*_{k,d}.\textit{Info})$
    if $\textit{Pl-commands}$ contains $\texttt{Payload}{\leftarrow}\texttt{Mail}_{k,d}.\texttt{Info}$ for some k, d, i
      then $\textit{Copy}(\_\textit{Mail}^*_{k,d}.\textit{Info}, \_\textit{Payload}'_i)$
    for $k \in \{-1, 1\}$ **do**
(1)   **let** $i = (P-1)(k+1)/2$
      $\textit{Copy}(\_\textit{Payload}'_i, \textit{Temp-Payload-to-nb}_k)$

when an instruction $\textit{Mail}_{k,d} \leftarrow \textit{Payload}_i$ is encountered it is not interpreted by an action, only by outputting the corresponding symbolic command.

At $\textit{Age} = U - 1$ a final computation step takes place in the rule `Finish` in Algorithm 19.12, which indeed is only a single step. The part applying to germ cells will be explained later.

Algorithm 19.12: rule `Finish`

    if *Doomed* = 1 **or** (*Kind* = *Germ* **and** *Addr* $\notin [0, Q)$) **then**
        Kind ← *Latent*
    else
        Addr ← *Addr* mod Q
        Age ← 0
        Receiving ← 1
        if *Kind* ∈ {*Growth*, *Germ*} **then**
            Kind ← *Member*
            New ← 1
        else
            New ← 0
            Util ← *Hold*
            for $k \in \{-1, 1\}$ **do** *Payload-to-nb$_k$* ← *Temp-Payload-to-nb$_k$*

19.4   Lifting

Recall that the payload computation was carried out in germ cells. As seen in Section 14.4, a germ is trying to grow into an area occupied by five colonies. Only the middle part will become a new colony, intended to simulate a new big cell: a latent cell on a higher level. Before the germ work period ends, however, the new colony will carry out a much simplified version of Algorithm 19.8 (`Refresh-payload`) called `Lift`, creating the error checks of the new level for the payload. Also, all cells of the new colony are supposed to be of the same color, but this color must be recorded also in the new, represented big cell: assume this information is (along with its error checks) in place _*Color** on the track *Util*.

    As with all germ actions, no damage rectangle is expected, so there is no repetition-and-vote.

Algorithm 19.13: sub-rule `Lift`

    for $i = 0$ **to** $P^* - 1$ **do**
        Copy(_*Payload$_i$*, _*To-encode*);
        Encode(_*To-encode*, _*Encoded*);
        Copy(_*Encoded*, _*Payload$'_i$*)
    Encode into _*Color** the *Color* of cell with address 0.

20 The simulated medium is robust

20.1 Legality

Here we will prove parts a)-d) of Condition 10.16 (Computation Property) for big cells. In the lemmas that follow, we fix a cell x and denote

$$I = [x - (Q + 1.1)B, x + (2Q + 1.1B)) .$$

Lemma 20.1 *For some time t, let $J = (t - 2T^{\bullet *} - \mathsf{split\text{-}t}, t]$. Assume that the rectangle $I \times J$ is damage-free, and at time t the colony $\mathcal{C} = \mathcal{C}(x)$ is covered by a domain of cells belonging to the same work period. Then the string $\eta(\mathcal{C}, t)$ is error-free in the sense of Definition 19.5.*

Proof. Suppose that during the whole interval $(t - 2T^{\bullet *}, t]$, the colony $\mathcal{C}$ is covered by a domain. In the absence of damage, this colony performs at least one complete work period of computation without any interference from errors inside $\mathcal{C}$. It follows then from the definition of the program (summarized in Algorithm 12.1) that the string s, as a result of such a computation, is error-free.

Suppose now that there is a first time $t' > t - 2T^{\bullet *}$ such that the colony is covered by a domain after t', but not before. Leading back a trace-back path from any cell of $\mathcal{C}$, starting at time t' the path is at any time in a domain covering its originating colony. Indeed, otherwise Lemma 15.17 (Bad Gap Inference) implies a bad gap which would not be closed, contradicting the assumption that $\mathcal{C}$ is covered by a domain at time t. The only possibility is thus that of a growing arm that finishes covering $\mathcal{C}$ at time t'.

Now if $t' \leq t - 1.5T^{\bullet *}$ Lemma 16.4 (Colony Trace-back) together with the number of steps allowed between the end of growth and the work period end implies that a new work period starts before time $t - T^{\bullet *}$, leaving a whole work period until time, guaranteeing an error-free colony. If $t' > t - 1.5T^{\bullet *}$ then Lemma 16.4 together with the number of steps allowed between the start and end of growth implies that the growth started after time $t - 2T^{\bullet *}$ using the growth rule, so by the time t' the whole colony encodes a latent big cell in an error-free way. No new errors arise later. $\square$

Lemma 20.2 (Legality) *Assume that, for some number a, the set Damage* does not intersect $[x, x + QB) \times (a - 2T^{\bullet *}, a + 2T^{\bullet *}]$, further that σ_1, σ_2, as in Definition 10.13 (Special switching times) are defined for cell x of medium M^* in the interval $(a - 2T^{\bullet *}, a + 2T^{\bullet *}]$, using Definition 19.5 (Decoding). Then the following holds, using Definition 2.8 (Legality):*

  a) $legal_{k+1}(\eta^*(x, \sigma_2-), \eta^*(x, \sigma_2)) = 1$, thus proving Condition 10.16c).

b) $T_\bullet^* \leq \sigma_2 - \sigma_1 \leq T^{\bullet*}$, *thus proving Condition 10.16b).*

c) *The randomization property $g(\alpha(rand), j, W_0(x, a), \eta^*)$, as defined in Condition 10.16a) will hold: it can be expressed in the needed canonical simulation of Definition 7.13.*

Proof. Let $I = [x, x + QB)$. Our assumption implies that damage is covered by at most one island in $I \times (a - 2T^{\bullet*}, a + 2T^{\bullet*}]$. According to Definition 19.5 (Decoding), there is a latest time v_2 in $(\sigma_2 - 2QT^\bullet, \sigma_2]$ that is a switching time of a cell of colony $\mathcal{C}(x)$ such that I is damage-free during $(v_2 - \mathsf{split\text{-}t}, v_2]$, and at time v_2 the colony is covered by a domain of cells belonging to the same work period. The value $\eta^*(x, \sigma_2)$ was decoded from the state of the colony at time v_2. Let y_1 be the cell with address $\lfloor Q/2 \rfloor$ in colony $\mathcal{C}(x)$, then (y_1, v_2) is traceable, as in Definition 15.4. Build a trace-back path from (y_1, v_2) to the latest time v_1 in $(\sigma_1 - 2QT^\bullet, \sigma_1]$ at which the colony is again covered with member cells belonging to the same work period, moreover I is damage-free during $(v_1 - \mathsf{split\text{-}t}, v_1]$. There is such a time, due to the definition of σ_1. Since σ_1, σ_2 are defined, the colony $\mathcal{C}(x)$ encodes at time v_1 with at most 4 errors a big cell with value different from the one encoded at time v_2.

1. If I is damage-free during $(v_1, v_2]$ then claim a) holds.

   Proof. The computation runs now without faults, and as planned, computes an error-free value. Rule `Compute` in Algorithm 19.7 applies `Legalize` to the information retrieved from the neighbor colonies, so the result computed from this information will be a legal one. The switching time σ_2 occurs just when the last cell of the colony passes to a new work period. The process makes sure that either all cells of the colony are doomed (when the computed value is vacant), or none them. If they are doomed then the whole colony dies. Otherwise, the rule `Finish` installs the remaining changes in one step. In both cases, the state at time σ_2 will be a legal consequence of the state at time σ_2-.

2. If I is not damage-free during $(v_1, v_2]$ then claim a) still holds.

   Proof. Let $t_1 \in (v_1, v_2]$ be some time at which damage intersects I, and let t_0 be the latest time before $\min(t_1 - 1.1\,\mathsf{destr\text{-}t}\,Q, v_1)$ which is the switching time of some cell, with the property that all cells of the colony belong to the same work period. By Lemma 16.1 (Cover), at time t_0, the colony is covered by a domain of cells. Lemma 20.1 implies that the colony encodes an error-free string.

   Let us follow the development of the colony forward in time, starting from t_0. Healing succeeds, due to the Cover Lemma. Locally maintained fields will be restored in a short time following the occurrence of damage, via the

rule `Loc-maintain` while other fields changed in at most an interval of Δ cells, with Δ defined in (13.4). After this, the work of the colony can be followed as before.

The time of the damage may fall into at most one of the three iterations of any of the rules with three iterations (`Refresh-payload`, `Update-payload`, `Legalize`, `Compute`). The output of this one iteration in Vote_r will be outvoted by the other two values in the final majority vote. So the only errors in Hold can be the ones due to their short-term damage in the final majority vote; these change at most Δ cells. In summary, we can make the same conclusion as part 1, except that the encoded string may contain up to 2 errors (in case the island occurs in the last part of computation or later).

There is an exception to this reasoning: in the rules `Refresh-payload` and `Update-payload`, the copy command is not repeated three times, so it is protected from the damage only in a limited way in the rule `Move-mail` of Algorithm 18.1. This protection makes sure that wrong information planted by damage is copied to at most one segment of size Δ; however, that one segment will still be affected, so the damage rectangle can cause 2 errors in the place it occurs and 2 more in the place to which the information is carried. But the error-correcting code, being able to correct 4 errors, will still deal with this.

3. We have $T_\bullet^* \leq \sigma_2 - \sigma_1$.

Proof. Consider a cell y of the colony, for example the one with address $\lfloor Q/2 \rfloor$, and lead a trace-back path from time σ_2 to time σ_1. Suppose it has n links: none of these are parental. At time σ_j- all cells have ages $\geq U - 2Q$, hence the age progress along the path is at least $U - 2Q$.

By Lemma 15.8, the age progress is $U - 2Q \leq n/p_1 + \Delta + 2$, hence

$$n \geq p_1 U - p_1(2Q + \Delta + 2).$$

By the same lemma,

$$\begin{aligned}
\sigma_2 - \sigma_1 &\geq nT_\bullet - (\Delta + 1)T_\bullet \\
&\geq T_\bullet(p_1 U - (2p_1 Q + (p_1 + 1)\Delta + 2p_1 + 1)) \\
&= T_\bullet U' - T_\bullet(2p_1 Q + (p_1 + 1)\Delta + 2p_1 + 1) \\
&= T_\bullet U'\big(1 - (2p_1 Q + (p_1 + 1)\Delta + 2p_1 + 1)/U'\big) \\
&\geq T_\bullet U'(1 - RQ/U') = T_\bullet^*,
\end{aligned}$$

where we used (12.2), further $U' = U_k'$ was defined in Section 11. Indeed, sufficiently large R will satisfy the last inequality.

4. We have $\sigma_2 - \sigma_1 \leq T^{\bullet*}$.

   Proof. Lemma 16.4 (Colony Trace-back) lower-bounds the progress along a trace-back path; then the proof is finished similarly to part 3 above.

5. Let us prove c).

   We need to prove essentially $\mathsf{P}\{\eta^*(x, \sigma_2).Rand = j\} \leq 1/2 + eps' + \varepsilon''$, given how ε'_k is defined in (11.8). Here ε'' bounds the probability that damage occurs in the window $W_0(x, a)$ at all. On the other hand, when it does not occur then each cell at each time within the work period performs its needed action: in particular, it carries out the rule `Compute` of Algorithm 19.7, and within it, the rule `Randomize`. This rule relies on the randomization action of a well-defined cell at a well-defined age of the work period, with the result being j with probability $\leq 1/2 + \varepsilon'$. The probability can only be increased by the probability bound that damage does occur in W_0. It would be now just a tedious exercise to actually express $g(\alpha(rand), j, W_0(x, a), \eta^*)$ formally as needed by a canonical simulation.

   $\square$

Lemma 20.3 *Assuming η is a trajectory, η^* satisfies Condition 10.16d).*

Proof. The proof is similar to that of the above lemma, only simpler. There is a latest time v_2 in $(\sigma_2 - 2QT^\bullet, \sigma_2]$ that is a switching time of a cell of colony $\mathcal{C} = \mathcal{C}(x)$ such that I is damage-free during $(v_2 - \mathsf{split\text{-}t}, v_2]$, and at time v_2 the colony is covered by a domain of cells belonging to the same work period. The value $\eta^*(x, \sigma_2)$ was decoded from the state of the colony at time v_2. Let y_1 be the cell with address $\lfloor Q/2 \rfloor$ in colony $\mathcal{C}(x)$, then (y_1, v_2) is traceable, as in Definition 15.4. Now the a trace-back path from (y_1, v_2) will lead to a cell still near the middle of colony $\mathcal{C}$, where $\mathcal{C}$ is now not covered by member cells. According to the Cover Lemma it the cell is still in a domain without exposed edges, so this domain will cover $\mathcal{C}$ as well as an originating colony. Tracing back further one can see that the cells covering $\mathcal{C}$ (whether outer or germ) were all formed by the rule `Grow.passive` or `Germ-grow.passive` of Algorithms 14.8, 14.16 so they encode a latent big cell. This is in accordance with Condition 10.16d), as the rule `Birth` of Algorithm 14.4 requires that a cell born from a vacant one is latent. The trace-back also shows that the germ growth succeeded only if the created colony is at a distance $\geq 2QB$ from existing big cells, as requires the definition of emergence. $\square$

Lemma 20.4 *Under the same condition on Damage* as in Lemma 20.2, assume that we have a full colony in which all cells have ages before the start of rule `Update-loc-maint` (Algorithm 19.6).*

a) *The locally maintained fields* **Doomed** *and* **Growing**$_j$ *will be homogeneous with the exception of an interval of at most Δ cells.*

b) *The value of* **Doomed** *is true (almost) everywhere if and only if* **Hold** *represents the vacant state* Vac^*.

c) *For $j \in \{-1, 1\}$, the value of* **Growing**$_j$ *is equal (almost) everywhere to* **Creating**$_j^*$ *of the cell state represented by the colony.*

Proof. The statement a) holds by the same reasoning as that of part 2 of the proof of Lemma 20.2, because being part of the rule `Compute`, rule `Update-loc-maint` is also repeated three times. The effects of later damage on these fields will be corrected via the rules `Heal` and `Loc-maintain`; when damage happens at the end of the work period, its effect is still limited to the location of the damage rectangle. The rest of the statement also follows from the error analysis of the `Compute` rule. □

The lemma below follows easily from the above and from Condition 10.9 (Time Marking).

Lemma 20.5 *Assume that, for some number a, $Damage^*$ does not intersect $\{x\} \times (a - T_\bullet^*/2, a + 2T^{\bullet *}]$. If $\eta^*(x, \cdot)$ has no switching time during $(a, a + 2T^{\bullet *}]$ then $\eta^*(x, a+)$ is vacant.*

The following lemma infers about the present, not only about the past as the Attribution Lemma. For an island $[a_0, a_1) \times (u_0, u_1]$, we call the rectangle

$$[a_0, a_1) \times (u_0, u_0 + 4\tau_2] \tag{20.1}$$

its *healing wake*.

Lemma 20.6 (Present Attribution) *Assume that the live cell $c_0 = (x_0, t_0)$ in colony $\mathcal{C}(z_0)$ is not a germ. Assume also that $\mathcal{C}$ at time t_0 does not intersect the healing wake of any island. 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 $Q < Age(c_0) < U - 1 - Q$ then every encoded package of this colony has at most 4 errors.*

2) *c_0 is a member cell from which a path of time projection at most $Q\tau_2$ leads back to a growth cell in $\mathcal{C}$. Assume, say, that it is a left growth cell; then it can be attributed to $\mathcal{C} + QB$. At time t_0, if $\mathcal{C} + QB$ does not intersect the healing wake of an island then $[x_0, z_0 + 2QB)$ is covered by a multi-domain. If $Q < Age(c_0) < U - 1 - Q$ then still every encoded package of this colony has at most 4 errors.*

3) c_0 is an outer cell, attributed to its originating colony, say $\mathcal{C}+QB$. If $\mathcal{C}+QB$ does not intersect the healing wake of an island then $[x_0, z_0 + 2QB)$ is covered at time t_0 by a domain.

4) c_0 is a member cell, and there is in $\mathcal{C}$ a backward path from it, with time projection $\leq 2\,\mathsf{split\text{-}t} + (Q+1)\tau_2$, to a domain of doomed member cells covering $\mathcal{C}$.

Proof.

1. Suppose that c_0 is a member cell.

   Recall the notation K, E_0 from (16.2) and Definition 16.2 (Attribution). By Lemma 16.3 (Attribution), c_0 can be attributed to the originating colony $\mathcal{C}$ which is either the colony of c_0 or not. Let time $t_1 \in K \setminus E_0$ be a time when $\mathcal{C}$ was covered by member cells. Using Lemma 16.4 (Colony Trace-back), we can go back from t_1 to a time t_2 before age compute-start in $\mathcal{C}$. Then as in part 2 of the proof of Lemma 20.2 (Legality), we can follow the development of the colony forwards and see that it forms a continuous domain together with its extension. The locations containing encoded information have at most 4 errors—as seen in the same proof.

   If the computation results in a nonvacant value for the represented big cell then, if $\mathcal{C} = \mathcal{C}(z_0)$ then case 1) holds. The condition $\mathit{Age} > Q$ guarantees that all cells of the colony belong to the same work period. Otherwise the represented field $\mathit{Creating}_j$ of the colony will be broadcast into the field $\mathit{Growing}_j$ of its cells, as shown in Lemma 20.4. The homogeneity of this latter field will be maintained by the healing rule and `Loc-maintain`. Thus, depending on the value of $\mathit{Creating}_j$ of the big cell, growth will take place and the growth forms a continuous domain with the originating colony until the age U when growth cells turn into members, and we have case 2). By the property of the growth rule the colony encodes a latent big cell, so the statement about errors also holds.

   Suppose that the computation results in a vacant value. Then $\mathit{Growing}_j$ will be 0 everywhere but in the healable wake of the damage. Growth cannot start accidentally by a temporary wrong value $\mathit{Growing}_j = 1$ in an endcell since there is enough time to correct this value during the long waiting time of `Grow`. Also, all cells become doomed. After $\mathit{Age} = 1$, any doomed cell dies, and the whole colony decays within $Q\tau_2$ time units. Before that, the colony is full. After that, we have case 4).

2. If c_0 is an outer cell then we have case 3).

   Proof. Lemma 16.3 (Attribution) attributes c_0 to the originating colony which is covered by member cells during $K \setminus E_0$. If c_0 is a channel cell then

this colony would not have time until t_0 even to finish its computation, so it could not be destroyed. If c_0 is a growth cell then the cell represented by this colony could not have become vacant, because then *Creating*$_j^*$ would have become 0, broadcast as *Growing* $= 0$ and so no growth would have occurred.

It forms a continuous domain with its extension, until the age $U - 1$. From that age on, they form a multi-domain. 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: the definitions (15.6), (12.3) show that destr-t Q is much smaller than compute-start.

<div style="text-align: right">□</div>

20.2   Robust media properties

To prove Condition 10.16e) for big cells, assume that for some a,

$$Damage^* \cap [x - 3QB, x + 4QB) \times (a - T_\bullet^*/2, a + 3T^{\bullet *}] = \emptyset. \qquad (20.2)$$

Recall the special switching times $\sigma_0, \sigma_1, \sigma_2$ of Definition 10.13. By Lemma 20.5, if there is no switching time during $(a, a + 2T^{\bullet *}]$ then $\eta^*(x, a+)$ is vacant.

Lemma 20.7 *For a trajectory η, Condition 10.16e) holds for the decoded trajectory η^*.*

Proof. Let us take times v_2, v_1 close from below σ_2, σ_1 as in the proof of Lemma 20.2 (Legality). From time v_1 on, follow the development of colony $\mathcal{C}$. The computation process can be treated similarly to the proof in that lemma; but in the communication with neighbor colonies, we must show that all retrieved information can be attributed to a single time. By event (20.2), the whole space-time area in which this communication takes place contains at most one damage rectangle. Repeated application of Lemma 14.7 (Creation) guarantees the success of extending arms via the rule *Extend*, defined in Section 14.3, over possible latent or germ cells or an opposing extension arm, showing that in each direction, within a good time bound either the extension arm of the colony $\mathcal{C}$ reaches its neighbor $\mathcal{C}'$ (provided it exists) or the neighbor's reaches it. 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 (at most) one arm remains on each side. In what follows, we call the space-time points *free* if they are outside the wake of the damage rectangle (see (20.1)). Cell x is called free at time t and time t is called free at cell x if (x, t) is free. We must find a time during the

communication period $(0, \mathsf{compute\text{-}start}]$ of $\mathcal{C}$ introduced in Definition 12.7 to which the retrieval from both neighbors can be attributed. Let a_1 be the last free time when some cell of $\mathcal{C}$ has age 0, and b_1 the first free time when some such cell has age $\mathsf{compute\text{-}start}$, and let $J_1 = (a_1, b_1]$. By (19.7) and the argument after it, we have

$$|J_1| > 1.5KRP^*QT_\bullet,$$

while the unsafe parts of the work period (the dwell period of the big cell simulated by $\mathcal{C}$) have a total length $< 1.5RP^*QT^\bullet$. Retrieval by rule `Retrieve` in Algorithm 19.4 happens in points of J_1 (provided it did not happen earlier) if a safe *Age* as defined in (19.6) has been seen in both neighbors. We will find a sufficiently large subinterval of J_1 when this condition holds. Let m_1 be the midpoint of J_1.

1. Suppose that at time m_1 the big cell x encoded by colony $\mathcal{C}$ has a left neighbor big cell, encoded by a colony $\mathcal{C}'$. Then there is a time interval $J_2 \subseteq J_1$ of size $\geq 0.7KRQT_\bullet$ in which the *Age*$_{-1}$ read by $\mathcal{C}$ in `Retrieve` is safe.

   Proof. It follows from Lemma 20.6 that m_1 is contained in a dwell period D of the big cell encoded by $\mathcal{C}'$, of length $\geq T_\bullet^*$. By (19.7), D has a subinterval J' of size at least $1.5KRP^*QT_\bullet$ in which the *Age*$_{-1}$ read by $\mathcal{C}$ is safe. Without loss of generality assume that J' is above m_1. In the worst case all the unsafe parts of D may come above m_1: still, the part J_2 of J' falling into J_1 has a length of at least

   $$|J_1|/2 - 1.5RP^*QT^\bullet \geq 0.7KRP^*QT_\bullet$$

   if K is large enough.

2. Suppose that at time m_1 the big cell x encoded by colony $\mathcal{C}$ has no left neighbor big cell. Then there is a time interval $J_2 \subseteq J_1$ of size $\geq 0.4KRP^*QT_\bullet$ in which the *Age*$_{-1}$ read by $\mathcal{C}$ is safe.

   Proof. Let J' be the largest subinterval of J_1 containing m_1 in which $\mathcal{C}$ has no left neighbor. Then either $|J'| > |J_1|/3$ or there is a subinterval of J_1 of size $\geq |J_1|/3$ that either contains a whole work period of a left neighbor of $\mathcal{C}$ or is covered by it. In each case, similarly to the reasoning in part 1 above, a safe time subinterval J_2 is found of size at least

   $$|J_1|/3 - 1.5RP^*QT^\bullet \geq 0.4KRP^*QT_\bullet$$

   if K is large enough.

We found a time interval J_2 of size $0.4KRP^*QT_\bullet$ in which both *Age* and *Age*$_{-1}$ are safe. Repeating the argument we find a time interval $J_3 \subset J_2$ of

size $0.1KRP^*QT_\bullet$ in which also Age_1 is safe. This shows that the `Retrieve` rule will succeed at some time t' during the work period of $\mathcal{C}$ and the simulation result can be attributed to this time point t' just as Condition 10.16e) requires.

<div align="right">□</div>

Lemma 20.8 (Growth) *For a trajectory η, Condition 10.16f) holds for the decoded trajectory η^*.*

Proof. This proof assumes that in case of conflict, growth to the right is preferred, but this choice is clearly arbitrary. As in Condition 10.16f), we assume that there is no non-vacant $\eta^*(y, t)$ with $0 < |y-x| < QB$, $t \in [a, a+3T^{\bullet*}]$, and that for every t in $[a, a+3T^{\bullet*}]$ at least one of $\eta^*(x-QB, t)$ and $\eta^*(x+QB, t)$ is a potential creator of x.

1. Suppose $x - QB$ is a potential creator in η^* at some time $t \in [a, T^{\bullet*}]$.
   Tracing backward and forward the evolution of the colony of big cell $x-QB$, we find a work period $(t_1, t_2]$ containing t. If the computation does not kill the big cell $x-QB$ then it will create the big cell x. Indeed, the only seeming obstacle to the growth to the right could be some non-germ cell z in its way. But Lemma 20.6 (Present Attribution) shows that such a z must be 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 preventing.

   Suppose that the computation kills the big cell $x - QB$. Then $x - QB$ could not become again a potential creator for time of length $\geq p_0 T_\bullet^*$. By our assumption then by the time $t_2 < t + T^\bullet < 2T^\bullet$, the big cell $x + QB$ must be a potential creator. As above, we would find that it has a whole work period $(t_1', t_2']$ containing t_2, and its computation cannot kill it because then neither $x - QB$ nor $x + QB$ would be a potential creator at time t_2'. So $x + QB$ would attempt to create x: let us show that it will succeed before time $a + 3T^\bullet$. As above, an obstacle to this could only be a non-germ cell z that is the right growth cell of a big cell y whose body overlaps that of $x - QB$. But this big cell y must be new and thus not become a creator for a time of length $\geq p_0 T_\bullet^*$.

2. Suppose $x - QB$ is not a potential creator in η^* at any time $[a, T^{\bullet*}]$.
   Then $x + QB$ is a potential creator in η^* at time a. As above, we find a work period $(t_1, t_2]$ of the colony $\mathcal{C}(x + QB)$ containing a. The computation does not kill the big cell $x - QB$ because then at time t_2 neither $x - QB$ nor $x + QB$ would be a potential creator. So $x + QB$ will extend a growth arm to the left, which can only be stopped by some non-germ cell z in the way. As in the above reasoning, z must be a right growth cell of a live big cell

y in $(x - 2QB, x]$. Because right growth is preferred, this would succeed, creating a cell y. If $y = x$ then x has become non-vacant, so we are done.

Let us show that other cases are not possible. We cannot have $y = x - QB$ because we assumed that $x - QB$ is not a potential creator during this time. The body of y cannot overlap that of x (without being equal to it) because this was excluded by the original assumption. In the remaining case, y creates big cell $y + QB$ which overlaps the body of x while not equal to it, but this is again just what has been excluded.

<div align="right">□</div>

20.3   The amplifier parameters

Here we prove Lemma 11.7 (Amplifier), and with this, as noted after the statement of that lemma, we finish the proof of Theorems 7.4 and 7.5 for the case of infinite space. This lemma says that a uniform amplifier complex can be built with the parameters defined in Section 11, with large enough R. The main ingredient is the sequence of media M_k as in (11.19) and the sequence of codes φ_{k*}, Φ_k^*, defined in the course of the proof. It remains to verify the properties of the amplifier complex listed in Lemma 11.7. Let us recall them here.

a) $(\textit{Payload}^k)$ is an aggregated field for (φ_{k*}), as defined in Section 11.

b) The damage map of the simulation Φ_k is defined as in Section 10.1.

c) Tr_k carries $Pl\text{-}trans_k$ as in Definition 11.4.

d) Φ_k has ε_k''-trickle-down.

The first two 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 $T_{\bullet k}$, $T_k^\bullet$ as the work period bounds. In the construction and the proof we relied implicitly on the properties proved in Lemma 11.6, mostly expressed as inequalities.

To prove the two other properties it needs to be shown yet that ε_k' indeed serves as the bound in the definition of $M_k = Rob(\cdots)$, and that the simulation has ε_k''-trickle-down.

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''-trickle-down. We must show that in a big cell transition, the probability that the $\textit{Rand}^*$ 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 island during the whole work period, the field Rand^* was computed with the help of the rule $\mathtt{Randomize}$. 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^*. 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 island intersects the colony work period. Therefore the probability that $\mathit{Rand}^* \neq X$ 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 island occurs during a colony work period. Under such condition, the rule $\mathtt{Refresh\text{-}payload}$ and $\mathtt{Update\text{-}payloas}$ work without a hitch and each cell contains the information encoded by the code φ_* from the state of the big cell. So the simulation has the ε''_k-trickle-down property.

The number of steps in the work period fits into U'_k as long as the requirement (11.16) of amplifier parameters is satisfied. The parts where this may not be obvious is the coding-decoding part of the program. However the codes of Example 5.15 we use, for a fixed number of errors, can be computed in a linear number of algebraic operations (multiplication, division). Indeed, the sets of equations to be solved involve only constant-size matrices, and multiplications/divisions were explicitly allowed in our rule language in Section 9.2.

21 Self-organization

In the present paper, self-organization is not a goal in itself but a tool to achieve reliability without a hierarchical initial configuration. We achieve this goal via defining a kind of amplifier that while working is creating more and more higher-level cells.

21.1 Color control

Consider a robust medium

$$M = Rob(Tr, B, T_\bullet, T^\bullet, \varepsilon, \varepsilon', r). \tag{21.1}$$

The field Color will play an important role in self-organization. Recall Definition 13.4, with two variants if the fitting relation between colors: 1) and 2).

From now on, we will focus on variant 2), as all reasoning can be transferred to the other variant, with simplifications.

Notation 21.1 For an interval $I = [a, b)$ we define its d-neighborhood as

$$\Gamma(I, d) = (a - d, b + d]. \tag{21.2}$$

We will also write $\Gamma(x, d) = (x - d, x + d]$. ⌟

Definition 21.2 (Color control) Suppose we are given a medium M, along with a history η. A set E of space is *color-fitted* in η at time t if $\mathsf{Color}(x, t) \le \mathsf{Color}(y, t)$ for every pair of cells $x < y$ whose body intersects E. A space-time set is color-fitted if it is color-fitted at each time t. Let

$$J_t = (t - (p_0 + 2)T^\bullet, t]. \tag{21.3}$$

The set E is *controlled* in η at time t if every subinterval K of it of size B, is within distance $< 2B$ of the body of a cell at that time, and the rectangle $K \times J_t$ intersects the (space-time) body of a cell. It is *color-controlled* if in addition, the set $E \times J_t$ is color-fitted. It has color c if it is color-controlled, and each cell whose space-time body intersects the set $E \times J_t$ has color c. ⌟

The following lemma shows that if an interval of size $2B$ contains a cell then (in the absence of damage) it will never remain empty long. Indeed, the cell can only be killed by other cells trying to create a new cell; this creation fails only if others are in the way, and so on. The d-neighborhood $\Gamma(I, d)$ of an interval I was defined in (21.2).

Lemma 21.3 *Let $t_1 < t_2$ be times. Assume that interval $K = [x_0, x_0 + B)$ is controlled at time t_1, and*

$$\Gamma(K, 3B) \times (t_1 - (p_0 + 2)T^\bullet, t_2]$$

is damage-free. Then K is controlled at time t_2.

Proof. Let $I = (x_0 - B, x_0 + B]$, then a cell's body intersects K if and only if it is in I. Let $t \in (t_1, t_2]$. Suppose that K is controlled for all $t' < t$, that is for all $t' < t$, each set $I \times (t' - (p_0 + 2)T^\bullet, t']$ contains some cell. Let $t_0 = t - (p_0 + 2)T^\bullet$. We want to show that

  a) I contains some cell z at some time in $(t_0, t]$,

  b) at each such time the interval $\Gamma(K, 2B)$ intersects a cell body.

The statement a) may not hold only if a cell x_1 in I disappears at time t_0, so assume this happens. Then x_1 must have been erased by some cell y_1

about to create an adjacent neighbor whose body overlaps with the body of x_1. Without loss of generality assume

$$x_1 \le x_0, \quad y_1 + B < x_1 < y_1 + 2B. \tag{21.4}$$

As $x_1 > x_0 - B$, we have

$$x_0 - 3B < y_1 < x_1 - B \le x_0 - B. \tag{21.5}$$

If $y_1 \in I$ then let $z \leftarrow y_1$, now assume it is not. According to the rule *Adapt*, cell y_1 must have had $Dying_0 = 0$, $Creating_1 = 1$ to be able to erase, and therefore survives until time $t_0 + p_1 T_\bullet$ as $Die(p)$ always has $p \ge p_1$ by Condition 14.16. Since according to Condition 14.16 only the rule *Create* changes $Creating_1$, and only when a right adjacent neighbor has disappeared, cell y_1 keeps trying to create a right neighbor.
1. Suppose y_1 succeeds in creating $y_1 + B$ before time $t_0 + 2T^\bullet$.
   If $y_1 + B > x_0 - B$ then set $z \leftarrow y_1 + B$. Suppose $y_1 + B \le x_0 - B$. Then $y_1 + B$ turns on $Creating_1$ within time $p_0 T^\bullet$, and then tries to create $y_1 + 2B$. If it succeeds then, by (21.5) we can set $z \leftarrow y_1 + 2B$. If it fails then a cell x_2 with $y_1 + 2B < x_2 < y_1 + 3B$ is in the way, and we can set $z \leftarrow x_2$.
2. Suppose that y_1 does not succeed creating $y_1 + B$ before time $t_0 + 2T^\bullet$.
   Then some cell x_2 with $y_1 + B < x_2 < y_1 + 2B$ interferes. This cell was not there at time t_0 as, by (21.4), it would have overlapped with cell x_1. Being so close to y_1, cell x_2 could not have arisen by spontaneous birth, hence it must have been created by $x_2 + B$ which, as any creating cell, must still be alive after the creating time, so we can set $z \leftarrow x_2 + B$.

We have shown a). The proof also shows that unless x_1 survives y_1 will, therefore for all t' in $(t_0, t]$, it is within $2B$ of interval K, which also shows b).
□

Lemma 21.4 (Lasting color control) *Suppose that in a trajectory η of medium $M = M_k$ of our amplifier, for an intervals E and times $t_1 < t_2$, the interval*

$$E' = \Gamma(E, B(t_2 - t_1)/T_\bullet)$$

is color-controlled at time t_1 and $I \times (t_1 - 2T^\bullet, t_2]$ is damage-free. Then E is color-controlled at time t_2.

Proof. We have to prove that E is color-controlled at time t_2. Lemma 21.3 implies that E is controlled at time t_2; what remains to show is that the rectangle $\Gamma(E, B) \times (t_2 - 3T^\bullet, t]$ is color-fitted. By the definition of control, in the space-time area considered, every interval of size B is within less than

$2B$ distance from some cell body, and this prevents any germ cell from being born, due to Condition 10.16d). Suppose that two non-fitting controlling cells was created: say, a cell with color 1 on the left of a cell with color 0. Then one could construct a path from both of these backwards, one with color 1 and one with color 0. These paths cannot cross, being of different color, nor can reach outside E' during $(t_1, t_2]$ since it needs at least $T_\bullet$ time units to move one cell width away from E. So we would find a cell with color 1 on the left of a cell with color 0 in E' at time t_1, contradicting the assumption. $\qquad\square$

An interval will become controlled soon even if a cell is only nearby:

Lemma 21.5 $K = [x_0, x_0 + B)$ *be a space interval and* t_1, t_2 *times with* $t_2 = t_0 + dp_0 T^\bullet$, *where*

$$d < p_1/\lambda p_0. \tag{21.6}$$

Assume that $\Gamma(K, (d+1)B) \times (t_1 - 3T^\bullet, t_2]$ *is damage-free, and the body of a cell* x_1 *intersects* $\Gamma(K, dB)$ *at time* t_1. *Then* K *is controlled at time* t_2.

Proof. The proof is similar to that Lemma 21.3, so we only sketch it. Assuming x_1 is the closest cell to K, say from the left, it extends an arm of cells to the right, where each step of extension takes time $\leq p_0 T^\bullet$, for a total of $\leq dp_0 T^\bullet$. This can be stopped only in two ways. 1: if a cell x_2 closer to K emerges as an obstacle, in which case we continue the reasoning from x_2, 2: If x_1 gets killed by a cell y_1 from the left, in which case we continue the reasoning from y_1. We know that y_1 cannot die for a time of at least $p_1 T_\bullet$, and by the assumption (21.6) we have $dp_0 T^\bullet < p_1 T_\bullet$. $\qquad\square$

Definition 21.6 In a trajectory η of a medium M, we will say that a space interval at time t is *germ-level*, if it has no colony cells. $\qquad\lrcorner$

According to the lemma below, if only a small subinterval has possibly different color in an otherwise color-controlled interval then the growth of this discrepancy in time is limited.

Lemma 21.7 *Let us be given a time* t_1, *an interval* J, *and*

$$I \supset \Gamma(J, |J|), \quad I' = \Gamma(I, B^*), \quad t_2 = t_1 + T_\bullet^*.$$

In a trajectory η, *assume that the space-time rectangle* $I' \times (t_1, t_2]$ *is free of damage. Assume also that* $\eta(\cdot, t_1)$ *is such that changing it in* J, *at time* t_1 *the whole interval* I' *will become color-controlled and germ-level. Then at time* t_2 *there is an interval* $J' \supset J$, *with size* $\leq 2|J|$ *such that changing* η *in* $\Gamma(J, mB)$, *the interval* I *becomes color-controlled. If it contains any colony cells at this time, they encode cells of age at most* 0.

Proof. Let $m = |J|/B$. If there are no cells in J then repeated application of Lemma 21.5 implies that it will be populated with cells coming from outside it by time $t_1 + mp_1\lambda p_0 T^\bullet$. Assume now that there are some cells in J. Any colony cells in J can only survive if they become part of a colony connected to either the left side or the right side, otherwise they will decay within time $mp_2 T^\bullet$, as they are in a small interval at least one of whose ends is exposed. So assume we have a germ inside J, of a color that is not fitting to its neighbors outside, say, on the left end. Then the left end of this germ cannot extend beyond the left end of J. The distance to which the right end can extend is limited by the rule `Germ-grow.active` of Algorithm 14.15 to mB. After the age `grow-end`, the germ starts decaying, and will disappear by time $t_1 + T_\bullet^*$. During this time, colony cells may intrude into the interval I', but in time $T_\bullet^*$ they can enter to a distance at most B^*.

Any colony cells in J that arose during the time indicated, belong to a brand new colony and as such, can only encode a big cell of age 0 that was not there at time t_1. The time $T_\bullet^*$ is not sufficient to increase this age even by 1. $\qquad\square$

In some simulations, color "trickles down".

Lemma 21.8 (Color trickle-down) *Let η be a trajectory of M, define the times t_0 and $t_1 = t_0 + T^{\bullet*}$. For interval J let $J' = \Gamma(J, \lambda B^*)$. If J' is color-controlled at time t_0 in $\Phi^*(\eta)$, and the area $J' \times (t_1 - (p_0 + 3)T^{\bullet*}, t_1]$ is damage-free in η, then J is color-controlled at time t_1 in η, and its colors can only be ones that were present in J at time t_0.*

Proof. Let K be any subinterval of J with $|K| = B$, and let $K' = [a, a + QB)$ be any subinterval of J' of size QB containing K. As J' is color-controlled at time t_0, the interval K' is intersected by the body of a big cell x_1 at some time t' with $t_0 - (p_0 + 2)T^{\bullet*} < t' \le t_0$. Without loss of generality assume that $x_1 \le a$. By Lemma 20.6 (Present Attribution) we can find a whole work period $(u_1, u_2]$ of x_1 containing t' and ending before time t_1. During this work period its colony will extend some arm to the right. If it does not cover K' then again, by the Present Attribution lemma we would find the left extension arm of another colony meeting this arm from the right. This way, at some time $t'' \in (u_1, u_2]$ the whole interval K' is covered by up to two intervals of adjacent small cells, meeting in a gap of size $< B$. So then the interval K' is color-controlled in η, and by Lemma 21.4, this property will be conserved until time t_1. $\qquad\square$

21.2 Colony birth

We will consider a trajectory η. The final goal in this section is to find, in Lemma 21.11, with large probability, some germ will grow into a big cell.

Recall the definition of ρ in (14.5), and its use in the rule of germ growth. In what follows we will repeatedly rely on the following observation, for any $\rho > 1$. Let $0 < L_1 < \cdots < L_n$ be such that $L_{i+1} \geq \rho L_i$ for all i. Then

$$L_1 + \cdots + L_n \leq \frac{\rho}{\rho - 1} L_n. \tag{21.7}$$

(We will use the fact that with the definition (14.5) of ρ we have $\frac{\rho}{\rho-1} = 5$.)

Lemma 21.9 *In a trajectory η, let $c_1 = (x_1, t_1)$ be a germ cell with G-size = L and Age $> 3RL$. Assume that at time t_1, with a germ cell body $K = [x_1, x_1 + 2B^*)$, the space-time rectangle*

$$\Gamma(K, 25B^*) \times \Gamma(t_1, T^{\bullet *}). \tag{21.8}$$

is free of damage, and the domain containing c_1 has an exposed edge. Then there is an L' with $\rho L < L' \leq 5Q$ such that in

$$\Gamma(x_1, 5L'B) \times [t_1, t_1 + 5RL'\tau_2) \tag{21.9}$$

there is a colony cell, or a germ cell with G-size $\geq L'$, or one with Addr $= e'_j$ for $j = \pm 1$ (recall (14.3)).

When ρL would be larger than $5Q$ then G-size $\geq \rho L$ is not possible, so only the case of a colony cell remains.

Proof. Let us note that if there is a colony cell then by Lemma 20.6 (Present Attribution) there is a whole extended colony containing it.

Let us construct a path P_1 backwards from c_1, staying inside the germ, which we will call G_1, to of an age that is at the beginning of the computation part which set G-size(c_1). Germ G_1 must be without exposed edge during the first half of this computation part, because otherwise it would decay during the second half (which is long enough for this), contradicting the age of c_1. Then at the end of the computation, at some time t_0, the germ G_1 has size $\geq L$, and all its cells have the same *Dominant* value, decided in the first step of the computation.

Let us follow the development of G_1 from time t_0 forwards. At some time one of its edges gets exposed, so some protected edge c' was killed by an attack: without loss of generality, assume it is from the left, and consider the development of the attacker domain G_2 after time t_0. There are the following possibilities according to Algorithm 14.16:

1) G_2 is an extended colony.

2) G_2 has $\textsf{G-edge}_{-1} = e'_{-1}$, and then naturally $\textsf{G-size} \geq 5Q/2$.

3) it has $\textsf{G-size} > \rho L$.

4) it has $\textsf{Dominant} = 1$ and $\textsf{G-size} \geq L/\rho$ while G_1 has $\textsf{Dominant} = 0$.

In case 1), this colony is still there at time t_1, because the period $(t_0, t_1]$ is too short for it to go through another work period and die. In the other cases, G_2 must also be past the computation that led to its $\textsf{G-size}$. Consider case 2). If G_2 does not get exposed before time t_1 then we are done. If it is exposed by an extended colony, we are back to case 1). If it is exposed by a germ then we are back to cases 3) or 4), but with a germ size at least ρ times larger than that of G_1. Repeating this we may get to a distance $L + \rho L + \ldots \rho^n L$. With $L' = \rho^n L$, by (21.7) this distance is $\leq \frac{\rho}{\rho-1} L'$.

Consider case 4). If G_2 gets exposed then we can repeat the whole above reasoning starting with G_2; however, now it is attacking, so case 4) cannot occur again with the same $\textsf{G-size}$. Now assume that G_2 does not get exposed: call this the case X. Then it can override G_1, but it is possible that at the same time, case X occurs also on the right. Still, as a result, within time $\tau_2 RL$, at least half of the size of G_1 will be added to one of the attacking germs, say G_2, so after it finishes its growth (and new computation), it gets

$$\textsf{G-size} \geq L(1/2 + 1/\rho) > L\rho.$$

$\square$

The following lemma strengthens the conclusion of Lemma 21.9.

Lemma 21.10 *Consider a trajectory η with a germ cell $c_1 = (x_1, t_1)$, $K = [x_1, x_1 + 2B^*)$ and $L = \textsf{G-size}(c_1)$, and assume that the large space-time rectangle (21.8) is free of damage. Let $5\tau_2 RL \leq H \leq T^{\bullet*}$. Then at some time $t' \in [t_1, t_1 + H]$ we can find a cell x' with a domain of germ size $L' \geq L$, where*

$$x' \in \Gamma(x_1, 5L'B). \tag{21.10}$$

If $c' = (x', t')$ does not belong to an extended colony or to a germ with maximal size then $t' = t_1 + H$. If $L' \leq \rho L$ then $c' = c_1$, and during $[t_1, t_1 + H]$ the germ of c_1 did not become exposed or have a neighbor that is a colony cell or a germ cell with $\textsf{G-size} \geq \rho L$.

Proof. Let $t_0 = t_1 + H$. We will iterate the application of Lemma 21.9, with $L_1 = L$. If the germ of c_1 does not get exposed or have a neighbor that is a

colony cell or a germ cell with *G-size* $\geq \rho L$ during $[t_1, t_1 + H]$ then its germ has size $\geq L$. As we will see this is the only possibility for $L' < \rho L$.

Suppose now that one of these events does occur. Then Lemma 21.9 gives a cell $c_2 = (x_2, t_2)$ in the area (21.9), without exposed edges, with *G-size*$(c_2) = L_2 \geq \rho L_1$, but with possibly $t_2 < t_0$. Follow the development of the germ of c_2 forward. If it reaches its maximum size or time t_0 without getting an exposed edge then we are done. Otherwise, the germ of c_2 gets an exposed edge at time t_3. Applying Lemma 21.9, to $c_3 = (x_2, t_3)$, we find a cell $c_4 = (x_4, t_4)$ that is either a germ end-cell, or has *G-size*$(c_4) \geq \rho L_2$. Repeat the argument as long as necessary. The relation (21.10) can be proved similarly to the corresponding relation in Lemma 21.9. $\qquad\square$

The following key lemma is the only one in this paper using a canonical simulation with random rectangles as in Section 7.3, and then relying on the fact that η is a *strong* trajectory. Let

$$\theta_k = e^{-Rk^2}. \tag{21.11}$$

Lemma 21.11 *Consider a trajectory η of M_k, with a germ cell $c_1 = (x_1, t_1)$, $K = [x_1, x_1 + 2B^*)$, and assume that the large space-time rectangle (21.8) is free of damage. The probability that the rectangle contains no colony cell is bounded by θ_k. Moreover, there is a canonical simulation in the sense of Definition 7.13 providing this estimate.*

Proof. 1. The following construction iterates that of Lemma 21.10, with cells $c_s = (x_s, t_s)$, $L_s = $ *G-size*(c_s) and time period of size

$$H_s = 10\lambda\tau_2 R L_s \tag{21.12}$$

in step s.

Starting with cell c_1, find a cell c' given by the conclusion of Lemma 21.10, call it $c_2 = (x_2, t_2)$. Repeat, getting to cells $c_3, c_4, \ldots$, until cell c_n when we reach a germ with maximal size, or a colony, or time $t_1 + 2T^{\bullet*}$. Let us call step s a *growth step* if $L_{s+1} \geq \rho L_s$, and *stalling step* otherwise. As seen in Lemma 21.10, in a stalling step a germ of c_s of size L_s does not get exposed, nor does it meet a colony cell or a cell with *G-size* $\geq \rho L_s$.

Let us examine a stalling step starting at time t_s, at the shorter end of the germ G_s of cell c_s, that is on the end in which direction its *Grow-dir* points; without loss of generality, assume that it is the left end. Let x_s be the cell of G_s with address $Q/2$. After t_s, the germ G_s may grow, but since it will not reach ρL_s, it will stall soon, at some time u_1. One possibility is that the

left end has reached its farthest possible address, another that another germ is in the way: one with *G-size* between L_s/ρ and ρL_s; we already excluded the other possibilities. In case the left end has reached the farthest address, we will turn attention to the right end, so without loss of generality, assume that the left end does not reach the farthest address, because another germ G' is in the way.

Let x' be the central cell of G', and look at the next $\lambda + 2$ times in which x_s or x' make a random choice of *Dominant*. Let $\mathcal{E}$ be the event that each choice is *Dominant*$(x_s) = 1$ and *Dominant*$(x') = 0$. We claim that if $\mathcal{E}$ occurs then G_s will advance, so the current step is not a stalling step. To avoid some case distinctions, call $x_s = y_1$, $x' = y_0$. Let j be such that the first choice of y_{1-j}, at some time τ, falls between two choices of y_j at times τ' and τ''. If τ happens at least $0.1RL$ age steps before τ'' them G_s will advance before τ''. Otherwise G_s will advance soon after τ''. Now, τ occurs before $\lambda + 1$ choices made by y_j; indeed, $T^\bullet/T_\bullet \leq \lambda$ implies that the stages of y_{1-j} are at most λ times longer than those of y_j. So the current step is stalling only if $\neg\mathcal{E}$ holds, that is with probability

$$\mathsf{P}(\neg\mathcal{E}) \leq 1 - (1/2 - \varepsilon')^{\lambda+2} < 1 - (1/2 - \varepsilon')^5 < 1 - 2^{-5},$$

because $\lambda \leq 2$ according to (11.2), and ε' can be chosen small.

by Condition 10.16a) on robust media.

2. Let us estimate first the probability that there is no colony in (21.8) informally (that is not in the sense of canonical simulation yet).

We assumed that the rectangle (21.8) is damage-free; the straightforward estimation of the probability of damage will come later. Our construction gives in step s a cell $c_s = (x_s, t_s)$. The number of growth steps is at most $\log(5Q)/\log\rho$. Recalling (14.5), this is $< 4\log Q$ when Q is large. As $L_s < 5Q$, each step has time length $\leq 50\lambda\tau_2 Q$. So the total number of steps is at least

$$n \geq T^{\bullet*}/25\lambda p_2 T^\bullet Q. \tag{21.13}$$

We are using the parameters in (11.9):

$$
\begin{aligned}
Q &= Q_k = R^{k+1}, \\
w &= 1/Rk^2, \\
U' &= RQ/w, \\
T^{\bullet*} &< 2U'_k T^\bullet < 2T^{\bullet*}, \\
\log Q &= (k+1)\log R.
\end{aligned}
\tag{21.14}
$$

so by (21.13) we have

$$T^{\bullet*}/QT^{\bullet} > U'/Q = R/w = R^2 k^2,$$
$$n \geq R^2 k^2 / 25\lambda p_2 =: cR^2 k^2. \tag{21.15}$$

As we have seen the probability for any particular step s to be a stalling step is $\leq 1 - 2^{-5}$. Using $\ln(1 + x) < 1 + x$ we have $\ln(1 - 2^{-5}) \leq -2^{-5}$,

$$r := -\ln(1 - 2^{-5}) > 2^{-5}. \tag{21.16}$$

For any particular m-element subset of steps the probability to be stalling steps is at most $(1 - 2^{-5})^m = e^{-rm}$. As seen, the number of stalling steps is at least $n - 4\log Q$, so the probability of no colony is

$$\leq \sum_{i > n - 4\log Q} \binom{n}{n-i} e^{-r(n-i)} = \sum_{i < 4\log Q} \binom{n}{i} e^{-r(n-i)}$$
$$< 4\log Q \cdot n^{4\log Q} e^{-r(n - 4\log Q)} = 4\log Q e^{-(rn - 4r\log Q + 4\ln n \log Q)}, \tag{21.17}$$

because the term inside the sum is increasing with i. Given (21.15) and (21.14), for large R the right-hand side is $< 4\log Q e^{-rn/2}$, and the sum of (21.17) for all $n > cR^2 k^2$ is

$$\leq 4\log Q \sum_{n > cR^2 k^2} e^{-rn/2} \leq 4(k+1)\log R \cdot e^{-rcR^2 k^2/2} \sum_{n > 0} e^{-rn/2}.$$

The multipliers of $e^{-rcR^2 k^2/2}$ don't change the rate of growth much, so we can bound the probability by θ_k as defined in (21.11).

3. Now we repeat the construction above, turning it into a canonical simulation.

As we constructed the sequence of cells $c_s = (x_s, t_s)$, $s = 1, 2, \ldots$, the times t_s were chosen on the basis of the trajectory η before t_s, so these times t_s are stopping times according to Definition 7.8. It is not important to upper-bound the number of steps, but $n^* = 2T^{\bullet*}/T^{\bullet}$ will serve. Following the reasoning in part 2, choose in every possible way a subset S of size $> n^* - 4\log Q$ of the set $\{1, \ldots, n^*\}$ of indices. Number these subsets as $S_1, S_2, \ldots, S_N$. Subset S_i represents the assumption that the steps $s \in S_i$ are the stalling steps. In order to define the canonical simulation, we repeat the reasoning of part 1 above.

Let us fix one of the sets $S = S_i$. For each element $s \in S$ in increasing order, let us examine a *presumed* stalling step starting at time t_s, at the shorter

end of the germ G_s of cell c_s. Without loss of generality, assume it is the left end. Let u_1 be the start of a growth part of a new work stage of G_s after t_s (started by the center cell of G_i). After u_1, the germ G_s may grow. If it reaches ρL_s, or meets a colony cell or a germ cell with G-size $\geq \rho L_s$, then s is not a stalling step: we set

$$\beta_{S,s} = \omega, \quad W_{i,r} = \emptyset, \tag{21.18}$$

and proceed to $j+1$. We will call this action *bailing out*. Another possibility is that the left end has reached its farthest possible address, another that another germ is in the way: one with G-size between L_s/ρ and ρL_s; we already excluded the other possibilities. In case the left end has reached the farthest address, we will turn attention to the right end, so without loss of generality, assume that the left end does not reach the farthest address, because another germ G' is in the way. Let x_s be the central cell of G_s and x' be that of C'. Now we look at the first $m = \lambda + 2$ times $\tau_1^*, \tau_2^*, \ldots$ when x_2 or x' make random choices, let $y_j \in \{x_s, x'\}$ be the element making its choice. For every possible binary sequence $\mathbf{b} = b_1, \ldots, b_m$ we will bail out unless $y_j = x_s$ if and only if $b_i = 1$. We will also bail out if the choice is Dominant$(y_j) = j$ for all j. Both of these decisions can be made at the last time τ_m, if not earlier. For the remaining cases,

$$\beta_{S,s,\mathbf{b},j} := \alpha(rand, f_j), \quad W_{S,s} := W_0(y_j, \tau_j^*),$$

where $\alpha(rand, d)$ and $W_0(x, t)$ were defined in Condition 10.16a). This construction results in the required canonical simulation. Summing up the non-bailed out bounds results in the same estimate as the more intuitive one given in parts 1 and 2 above.

$\square$

We now strengthen the conclusion of Lemma 21.11 for the cases when more is known about interval we start with.

Lemma 21.12 *Let us be given an interval $K = [x_1, x_1 + 2B^*)$, and let*

$$t_2 = t_1 + 26p_0 T^{\bullet *}, \quad K' = \Gamma(K, 2 \cdot (26p_0 + 2)\lambda B^*).$$

In a trajectory $\eta = \eta^k$ of medium $M = M_k$, assume that the space-time rectangle $K' \times (t_1, t_2]$ is free of damage. Assume also that there is an interval J of size $< |B^*|/3$ in which when changing η at time t_1 the whole interval K' can be made color-controlled in M. Then at time t_2, except for an event of probability θ defined in (21.11), the interval K is color-controlled in trajectory η^* of M^*. If K' is germ-level for η at time t_1 then K is germ-level for η^* at time t_1.

Proof. Assume $\Gamma(J, B^*) \subseteq K'$; if this is not true then we can simply ignore J. Let K'' be the larger of the two intervals or $K' \setminus J$. By Lemma 21.11, except for an event of probability θ, before time $t_1 + T^{\bullet*}$ a colony arises in K'', not farther than $25B^*$ from K. Then applying Lemma 21.5, to η^*, until time t_2 the interval K becomes controlled in M^*. Indeed, given that by Lemma 21.7, any non-fitting germ that started in J can grow only by a factor of 2 in each time interval $T_{\bullet}^*$, the size of such a germ remains much smaller than B^*, and will be overridden by the colonies.

Cells that are not color-fitted can only intrude over germ cells if they are colony cells. In time $T_{\bullet}^*$ they can cover at most one colony length, so in time $26p_0 T^{\bullet*}$, this amounts to $\leq 26p_0(T^{\bullet*}/T_{\bullet}^*) = 26p_0\lambda$ colony lengths, still away from K.

Let us show that K will be germ-level in η^*. At time t_1, there were no colony cells yet in K', so even if a new colony occurs right after t_1, it starts simulating a germ cell of M^*, with age 0. Every dwell period of such a cell takes time at least $T_{\bullet}^*$, so until time t_2 it can have at most $26p_0\lambda$ dwell periods, a constant less than the age U^* needed to become a colony cell of M^*. $\square$

21.3 Lifting the simulation level

Definition 21.13 (Probabilistic color control) Let us define the constants

$$C_1 = (26p_0 + 2)\lambda, \quad C_2 = 26p_0 + 2. \tag{21.19}$$

Let "blue" be some color. For some parameter $\sigma > 0$, let $\mathcal{E}$ be an event derived from the behavior of trajectory η of medium M before time t. We will say that trajectory η is *germ-level blue modulo* $(\sigma, \mathcal{E})$ at time t on the finite or infinite interval H if for every set of space intervals $I_1, \ldots, I_n \subset H$, of size $C_1 B$, at distance $\geq 2C_1 B$ from each other, the probability that $\neg\mathcal{E}$ holds and none of the I_j is germ-level and blue is at most σ^n. When H is the whole space then we will have $\mathcal{E} = \emptyset$, so the "modulo $\mathcal{E}$" qualification can be omitted.

For an amplifier, the parameter θ_k was defined in (21.11). Let

$$\sigma_k = \begin{cases} \theta_1 & \text{if } k = 1, \\ 2\theta_{k-1} & \text{if } k > 1. \end{cases} \tag{21.20}$$

As usual, for level k we will simply write $\sigma = \sigma_k$, $\sigma^* = \sigma_{k+1}$, and so on. ⌟

A special case of the lemma below says that if H is the whole space then for every trajectory η of M, if H is germ-level blue in η at time t_1 modulo σ then it is germ-level blue for the trajectory η^* of M^* modulo σ^* at time

$t_1 + C_2 T^{\bullet *}$. The statement is a little more complex if H is not the whole space, because of intervals I_j that may fall close to its boundary.

Lemma 21.14 (Self-organization) *For a trajectory η of one of the media $M = M_k$, of our amplifier, and $\mathcal{E}$ an event depending on the behavior of η before t_1. For a finite or infinite space interval H, assume that $H' := \Gamma(H, C_1 B^*)$ is germ-level blue modulo $(\sigma, \mathcal{E})$ in η at time t_1, and let $t_2 = t_1 + C_2 T^{\bullet *}$. Let $\mathcal{E}^*$ be the event that $\mathcal{E}$ holds, or not all (0,1 or 2) intervals of size $C_1 B^*$ at the ends of H' are blue in η at time t_1. Then H is germ-level blue modulo $(\sigma^*, \mathcal{E}^*)$ at time t_2 for the trajectory η^* in M^*.*

Proof. Let $I_1, \ldots, I_n \subset H$ be any system of intervals of size $C_1 B^*$, at distance $\geq 2 C_1 B^*$ from each other. We need to bound the probability by $(\sigma^*)^n$ that $\neg \mathcal{E}^*$ holds and none of the intervals I_j are good in η^*. It is sufficient to look at one of these intervals—call it I, with

$$I' = \Gamma(I, C_1 B^*).$$

Indeed, as the space-time rectangles of the argument for all of them are disjoint, and the probability estimates come from a canonical simulation, the bounds σ^* on the probability will multiply. Here is an outline of the argument.

We will work within the space-time rectangle

$$I' \times (t_1, t_1 + C_2 T^{\bullet *}]. \tag{21.21}$$

- Estimate the probability of the event $\mathcal{F}_1$ that damage occurs in the rectangle (21.21).

- Estimate the probability of the event $\mathcal{F}_2$ that at time t_1 it is not true that the interval I' is germ-level blue with the possible exclusion of a subinterval J of size $C_1 B$.

- Show that assuming $\neg(\mathcal{F}_1 \cup \mathcal{F}_2)$, except for an event $\mathcal{F}_3$ of probability $\leq \theta$ the interval I turns germ-level blue for M^* by time t_2.

Condition 10.4 (Restoration Property) gives the upper bound

$$\mathsf{P}(\mathcal{F}_1) \leq \frac{3 C_1 B^* \cdot C_2 T^{\bullet *})}{B T^{\bullet}} \varepsilon \leq 4 C_1 C_2 Q U' \varepsilon. \tag{21.22}$$

Assuming $\neg \mathcal{E}^*$ implies $\neg \mathcal{E}$ as well. Let $\mathcal{F}_2$ be the event that $\neg \mathcal{E}$ holds and there is no interval J of size $C_1 B$ such that, setting $J' = \Gamma(J, 2 C_1 B)$, the set $I \setminus J'$ is blue at time t_1.

1. We have $\mathsf{P}(\mathcal{F}_2) \leq (C_1 Q \sigma)^2$.

Proof. Let $J_i = iC_1B + [0, C_1B)$. Under the assumption of $\neg\mathcal{E}$, which we will omit to write in the estimates below, for some i, h with $|i - h| \geq 4$, let $\mathcal{F}_{i,h}$ be the event that neither J_i nor J_h is germ-level and blue, and both intersect $I'..$ The germ-level blueness of η at time t_1 modulo σ implies $\mathsf{P}(\mathcal{F}_{i,h}) \leq \sigma^2$, and hence $\mathsf{P}\{\exists i, h : \mathcal{F}_{i,h}\} \leq (3C_1Q)^2\sigma^2$. If there is no such pair i, h then it is easy to see that there is an i such that $I \setminus J_i'$ is blue.

2. Let $\mathcal{F}_3$ be the event that $\neg(\mathcal{E} \cup \mathcal{F}_1 \cup \mathcal{F}_2)$ holds and I does not become germ-level blue in η^*. Then $\mathsf{P}(\mathcal{F}_3) \leq \theta$, where θ was defined in (21.11).

Proof. By $\neg\mathcal{F}_2$ there is a J of size C_1B such that $I' \setminus J$ is germ-level blue at time t_1, so we invoke Lemma 21.12.

Summarizing, the probability that under condition $\neg\mathcal{E}$ the interval I does not become blue in η^* is bounded by

$$\mathsf{P}(\mathcal{F}_1 \cup \mathcal{F}_2 \cup \mathcal{F}_3) \leq 4C_1C_2QU'\varepsilon + (C_1Q\sigma)^2 + \theta. \qquad (21.23)$$

Let us show that the latter is $\leq \sigma^*$, using the definitions in (21.20), those in (21.14), and the inequality (11.10). We have, for large enough R and small enough ε:

$$4C_1C_2Q_kU'_k\varepsilon_k = 4C_1C_2R^{2k+4}k^2\varepsilon_k < \theta_k/2,$$
$$(C_1Q_k\sigma_k)^2 = C_1^2R^{2k+2}e^{-2Rk^2} < \theta_k/2,$$

so (21.23) is bounded by $2\theta = \sigma^*$. $\qquad\qquad\square$

21.4   Computing supported by self-organization

Here we will prove the remaining main theorems, which rely on self-organization.

Proof of Theorem 7.4. The field that will be remembered will be *Color*. For the size N of our space, let us define the finite or infinite level K:

$$K = \bigvee\{k : 5B_k \leq N\}.$$

For a certain value of the field *Color* that we call blue, we create an initial configuration $\eta^1(\cdot, 0)$ with blue latent cells covering the whole space. Let the trajectories η^1, η^k be defined as before. The proof will show that by time $C_2T_k^\bullet$ the space will be germ-level blue modulo σ_k; from this via trickle-down we will be able to control the probability of any particular space-time point (x_1, t_1) not being blue.

Let $\mathcal{F}_1$ be the event $Blue \neq \eta^1(x_1, t_1).Color^1$. Eventually, we want to upper-bound $\mathsf{P}(\mathcal{F}_1)$. Let us consider first the case of infinite space, that is $K = \infty$.

Let

$$v_1 = t_1, \qquad\qquad v_{k+1} = v_k - T^\bullet_{k+1},$$
$$I_1 = \{x_1\}, \qquad\qquad I_{k+1} = \Gamma(I_k, \lambda B_{k+1}),$$
$$u_1 = 0, \qquad\qquad u_{k+1} = u_k + C_2 T^\bullet_{k+1}.$$

By Lemma 21.14, for each $k \le K$, in the trajectory η^k, the whole space is germ-level blue at time u_k modulo σ_k. Let $\mathcal{F}_k$ be the event that I_k is blue at time v_k in η^k.

1. We have $\mathsf{P}(\mathcal{F}_{k+1} \setminus \mathcal{F}_k) \overset{*}{<} U'_k Q_k \varepsilon_k$, where we used the notation introduced in (2.1).

   Proof. If damage does not occur in η^k in $I_{k+1} \times (v_k - (p_0 + 2)T^\bullet_{k+1}, v_k]$ and $\mathcal{F}_{k+1}$ holds then Lemma 21.8 (Color trickle-down) implies $\mathcal{F}_k$. The right-hand side above upper-bounds the probability of damage there. Now

$$U'_k Q_k \varepsilon_k < \varepsilon^{2^{n-2} + 2^{(n-3)/2}} < \varepsilon^{2^{n-2}},$$

Let

$$n = \bigwedge\{k : v_{k+1} \le u_{k+1} \text{ or } k = K\}, \quad t_2 = u_n.$$

If $K = \infty$ then we always have $v_{n+1} \le u_{n+1}$, hence

$$v_{n+1} = v_n - T^\bullet_{n+1} \le u_{n+1}, \quad u_n < v_n,$$
$$v_n \le u_{n+1} + T^\bullet_{n+1}, \quad v_n - u_n \le (C_2 + 1)T^\bullet_{n+1}.$$

Let $I' = \Gamma(I_n, (C_2 + 1)B_n T^\bullet_{n+1}/T_{\bullet n})$. Let $\mathcal{B}$ be the event that I' is blue in η^n at time u_n. Since η^n is blue at time u_n modulo σ_n for each n, we have

$$\mathsf{P}(\neg\mathcal{B}) \overset{*}{<} U'_n \sigma_n.$$

We claim $\mathsf{P}(\mathcal{B} \setminus \mathcal{F}_n) \overset{*}{<} (U'_n)^2 \varepsilon_n$. Indeed, if damage does not occur in η^k in $I' \times (u_n, v_n]$ then $\mathcal{B}$ and Lemma 21.4 imply $\mathcal{F}_n$. And the right-hand side upper-bounds the probability of damage occurring there. Thus,

$$\mathsf{P}(\neg\mathcal{F}_1) \le \sum_{k=1}^{n-1} \mathsf{P}(\mathcal{F}_{k+1} \setminus \mathcal{F}_k) + \mathsf{P}(\neg\mathcal{B}) + \mathsf{P}(\mathcal{B} \setminus \mathcal{F}_n)$$
$$\overset{*}{<} \sum_{k=1}^{n-1} U'_k Q_k \varepsilon_k + U'_n \sigma_n + (U'_n)^2 \varepsilon_n.$$

With the parameters of (11.9), large enough R and small enough ε we have

$$\sum_{k=1}^{n-1} \mathsf{P}(\mathcal{F}_{k+1} \setminus \mathcal{F}_i) = O(\varepsilon),$$

$$U'_n \sigma_n = R^{n+3} n^2 e^{-Rn^2} < e^{-Rn^2/2},$$

$$(U'_n)^2 \varepsilon_n = R^{2n+6} n^4 \varepsilon^{2^{n-2} + 2^{(n-3)/2}} < \varepsilon^{2^{n-2}},$$

for any somewhat large n and small ε, giving $\mathsf{P}(\neg \mathcal{F}_1) \overset{*}{<} e^{-Rn^2/2} + \varepsilon$. Let us upper-bound $e^{-Rn^2/2}$ by lower-bounding n. We have

$$t_1 \le (C_2 + 1) \sum_{k=1}^{n+1} T_k^\bullet \overset{*}{<} T_{n+1}^\bullet.$$

From (11.18) for large k:

$$t_1 \le T_{n+1}^\bullet = R^{(n+2)(n+3)/2-2}(n!)^2,$$

$$\ln t_1 < 0.6 n^2 \ln R,$$

$$n^2 > (1/0.6 \ln R) \ln t_1,$$

$$e^{-Rn^2} \le t_1^{-R/0.6 \ln R},$$

which leads to the estimate in Theorem 3.1.

2. Suppose $u_{n+1} < v_{n+1}$, hence $n = K$ and the space is finite.

Given $B_n \le N \le 5B_n$ and the explicit expression for B_n and (within constant factor of) $T_n^\bullet$ in (11.18), the estimate for ε_n in (11.10) and the definition (3.2), we have for large n, N, where for a later application we have been a little more generous:

$$N < B_{k+1} = R^{(n+1)(n+2)/2-1},$$

$$\log N < (n+1)^2 \log R/2,$$

$$n - 2 > (\log N)^{1/2}(2/\log R)^{1/2} - 4, \qquad (21.24)$$

$$2^{n-2} > c^{(\log N)^{1/2}} = h_0(N, c) \text{ for an appropriate } c > 1,$$

$$\varepsilon_n < \varepsilon^{2^{n-2}} \le \varepsilon^{h_0(N,c)}.$$

Let $\mathcal{B}$ be the event that the whole space $\mathbb{Z}_N$ is blue in η^n at time u_n. Since η^n is blue at time u_n modulo σ_n, and $5B_{n+1} > N$, we have

$$\mathsf{P}(\neg \mathcal{B}) \overset{*}{<} Q_n \sigma_n.$$

The event $\mathcal{B} \setminus \mathcal{F}_n$ occurs only if there is any damage in η^n between times u_n and v_n. We can upper-bound the probability of this within a constant factor by

$$\varepsilon_n N t_1 / B_n T_n^\bullet \leq \varepsilon_n t_1 / T_n^\bullet \leq \varepsilon_n t_1 \leq \varepsilon^{h_0(N,c)} t_1,$$

which is what Theorem 3.1 states.

$\square$

Proof of Theorem 7.6. This also proves Theorem 6.1, and the remark at the last part of the proof also proves Theorem 7.5 for the finite space. Let *Tr* be a given standard computing transition function over an alphabet Σ. Construct an amplifier as in Lemma 11.7 with the transition function *Pl-trans*$_1(\cdot) = Tr(\cdot)$. This amplifier defines cell body sizes B_k for all k. Given a string $\rho \in \Sigma_0^n$ in the input-output alphabet, let

$$l = \bigvee \{k : B_k \leq n\}.$$

Create a germ of cells of M_l covering the space interval $[0, n)$, with address $Q_l/2$ somewhere inside it. Fill its *Payload* track first with $*$, then write over the *Payload.Input* track the string ρ. Now apply consecutively the encodings ϕ_{*l}, $\phi_{*(l-1)}$, ..., to get a string ρ_l of $\mathbb{S}_1$. The code ψ_* whose existence is stated in the theorem is defined as $\psi_*(\rho) = \rho_l$, which gives $n_l := |\rho_l| < 2n$ as required by the theorem. From here, with *Init* as in Definition 3.6, we obtain the initial configuration of M_1 as $\eta(\cdot, 0) = Init_{\psi_*}(\rho)$. For readability, we will call the color -1 "blue", color 0 "white" and color 1 "red". First, for simplicity, assume that the space is infinite.

By the construction, the interval $[0, n_l)$ is already an l-level white germ. First we will show how the space outside it will also promptly self-organize to level l: of color blue on the left and red on the right. From then on, the self-organization continues on the whole space, and at the same time, the computation in the white germ proceeds reliably, lifting to a higher level in every step of self-organization. Finally, the trickle-down argument shows that at any time the result can be read out at the lowest level.

For $k = 1, 2, \ldots$, let

$$a_k = \sum_{i=2}^{k} C_1 B_i, \quad v_k = \sum_{i=2}^{k} C_2 T_i^\bullet,$$

$$J_k = [-a_k, n_l + a_k), \quad H_k^- = (-\infty, -a_k), \quad H_k^+ = [n_l + a_k, \infty),$$

then $J_l = [0, n_l)$, $H_1^- = (-\infty, 0)$. For $j = 1, 2, 3$ let $\mathcal{A}_{k,j}$ be the event that in η^k at time v_k, the interval $[-ja_{k+1}, -a_k)$ is not germ-level blue or $[n_l - a_k, n_l + ja_{k+1})$ is not germ-level red. Let

$$\mathcal{D}_k = \bigcup_{i=1}^{k} \mathcal{A}_{i,1}.$$

Note that $\mathcal{A}_{1,1} = \mathcal{D}_1 = \emptyset$.

1. For each k, in η^k at time v_k, the set $H_k^- \setminus H_{k+1}^-$ is germ-level blue and $H_l^+ \setminus H_{l+1}^+$ is germ-level red modulo $(\sigma_k, \mathcal{D}_{k-1})$.

   Proof. Interval H_1^- is germ-level blue at time 0 in η^1 by definition. Lemma 21.14 implies that H_2^- is germ-level blue in η^2 at time v_2 modulo $(\sigma_2, \mathcal{D}_1)$, further H_3^- is germ-level blue in η^3 at time v_3 modulo $(\sigma_3, \mathcal{D}_2)$, and so on. The same argument works for H_k^+.

   For $k \geq l$, let $\mathcal{F}_k$ be the event that there is damage in η^k in the space-time rectangle

   $$\Gamma(J_l, 3a_{k+1}) \times (0, v_{k+1}].$$

2. Assuming $\neg\mathcal{F}_l$, the interval J_l is part of a white germ at time v_l in η^l.

   Proof. We started from the interval J_l of level l cells surrounded by level 1 latent cells. In the absence of damage in η^l, these cells perform according to the program, hence up to time $v_l < 2T_{l+1}^\bullet$, none of them would die.

3. From $\neg(\mathcal{A}_{l,3} \cup \mathcal{F}_l)$ follows that in η^l at time $v_l + 2T_{l+1}^\bullet$, interval J_l is part of a white germ, and $\Gamma(J_l, 2a_{l+1})$ is germ-level color-controlled.

   Proof. At time v_l in η^l, as part 2 shows, $\neg\mathcal{F}_l$ implies that J_l is part of a white germ. And $\neg\mathcal{A}_{l,3}$, implies by definition that $[-3a_{l+1}, -a_l)$ is germ-level blue and $[n_l - a_l, n_l + 3a_{l+1})$ is germ-level red. We can now apply the argument of Lemma 21.7 simultaneously with with $I = [-3a_{l+1}, n_l)$, $J = [-a_l, 0)$, and with $I = [0, n_l + 3a_{l+1})$, $J = [n_l, n_l + a_l)$, to imply that in η^l at time $v_l + 2T_{l+1}^\bullet$, the whole interval $\Gamma(J_l, 2a_{l+1})$ becomes germ-level color-controlled.

4. From $\neg(\mathcal{A}_{l,3} \cup \mathcal{F}_l)$ follows that in η^{l+1} at time v_{l+1}, the interval J_l is part of a white germ, and interval $\Gamma(J_l, a_{l+1})$ is germ-level color-controlled, except for an event $\mathcal{E}_l$ of probability

   $$\leq (2C_1 + 7)\theta_l,$$

   where θ_l was defined in (21.11).

Proof. From $\mathcal{F}_l$ follows that J_l is part of a white germ also in η^{l+1}. By part 3, from $\neg(\mathcal{A}_{l,3} \cup \mathcal{F}_l)$ follows that in η^l at time $v_l + 2T^\bullet_{l+1}$, the interval $\Gamma(J_l, 2a_{l+1})$ is germ-level color-controlled. Note that $v_{l+1} = v_l + 2T^\bullet_{l+1} + 26p_0 T^\bullet_{l+1}$. Applying Lemma 21.12, for case $\eta = \eta^l$, $t_2 - t_1 = 26p_0 T^\bullet_{l+1}$, to any subinterval K of length $2B_{l+1}$ of $\Gamma(J_l, a_{l+1})$, we find that K becomes color-controlled in η^{l+1} except for an event of probability θ_l. If each of the r intervals of the form $[iB_{l+1}, (i+2)B_{l+1})$ intersecting $\Gamma(J_l, a_{l+1})$ becomes color-controlled then so does $[-a_{l+1}, n_l - a_{l+1})$. Let $\mathcal{E}_l$ be the event that any one of them does not become color-controlled, then $\mathsf{P}(\mathcal{E}_l) \leq r\theta_l$. The construction gave $|J_l| < 5B_{l+1}$, we know $a_{l+1} < 2C_1 B_{l+1}$, hence $|\Gamma(J_l, a_{l+1})| < (4C_1 + 5)B_{l+1}$, $r \leq 4C_1 + 9$.
Let us handle now first the case of infinite space $\mathbb{Z}$.

5. Let $k > l$. Assume that in η^k at time v_k the interval J_l is part of a white germ, and the interval $\Gamma(J_l, a_k)$ is germ-level color-controlled. Then from $\neg(\mathcal{A}_{k,2} \cup \mathcal{F}_k)$ follows that in η^{k+1} at time v_{k+1}, the interval J_l is part of a white germ, and interval $\Gamma(J_l, a_{k+1})$ is germ-level color-controlled, except for an event $\mathcal{E}_k$ of probability $\leq (2C_1 + 3)\theta_k$.

Proof. (This proof is similar to the one above.) From $\mathcal{F}_k$ follows that J_l is part of a white germ also in η^{k+1}. From $\neg\mathcal{A}_{k,2}$ follows that in η^l at time v_l also the interval $\Gamma(J_l, 2a_{k+1})$ is germ-level color-controlled. Applying Lemma 21.12, for case $\eta = \eta^k$, $t_2 - t_1 = C_2 T^\bullet_{k+1}$, to any subinterval K of length $2B_{k+1}$ of $\Gamma(J_l, a_{k+1})$, we find that K becomes color-controlled in η^{k+1} except for an event of probability θ_k. If each of the r intervals of the form $[iB_{k+1}, (i+2)B_{k+1})$ intersecting $\Gamma(J_l, a_{l+1})$ becomes color-controlled then so does $\Gamma(J_l, a_{l+1})$. Let $\mathcal{E}_k$ be the event that any one of them does not become color-controlled; then $\mathsf{P}(\mathcal{E}_k) \leq r\theta_k$. The construction gave $|J_l| < B_{k+1}$, we know $a_{k+1} < 2C_1 B_{k+1}$, hence $|\Gamma(J_l, a_{l+1})| < (4C_1 + 1)B_{l+1}$, $r \leq 4C_1 + 5$.
For $k \geq l$ let

$$\mathcal{D}'_k = \bigcup_{i=l}^{k} \mathcal{A}_{l,2}, \quad \mathcal{G}_k = \mathcal{D}_{l-1} \cup \mathcal{A}_{l,3} \cup \mathcal{D}'_{k-1} \cup \bigcup_{i=l}^{k-1} \mathcal{F}_i \cup \bigcup_{i=l}^{k-1} \mathcal{E}_i,$$

where the events $\mathcal{E}_i$ were defined in the proofs of part 4 and part 5.

6. Let $k > l$. Assuming $\neg\mathcal{G}_k$, in η^{l+1} at time v_{l+1} the interval $\Gamma(J_l, a_k)$ is germ-level color-controlled, and the interval J_l is part of a white germ.

Also,

$$\mathsf{P}(\mathcal{G}_k) \leq 2\sum_{i=1}^{l-1} Q_i\sigma_i + 6Q_l\sigma_l + 4\sum_{i=l}^{k-1} Q_i\sigma_i$$

$$+ 33\sum_{i=l}^{k-1} Q_iU_i'\varepsilon_i + (2C_1+7)\theta_l + \sum_{i=l+1}^{k-1}(4C_1+5)\theta_i = O(\sigma_1). \tag{21.25}$$

Proof. The statement before the probability estimate, for $k = l+1$, follows from part 4. For $k > l+1$, it follows by induction from part 5. It remains to prove (21.25). From the definition, $\mathcal{D}_k = \bigcup_{i=2}^{k}(\mathcal{A}_{i,1} \setminus \mathcal{D}_{i-1})$. From

$$\mathsf{P}(\mathcal{A}_{i+1,j} \setminus \mathcal{A}_{i,j}) \leq 2jQ_k\sigma_k$$

we have $\mathsf{P}(\mathcal{D}_k) \leq 2\sum_{i=1}^{k-1}Q_i\sigma_i$, $\mathsf{P}(\mathcal{D}_k') \leq 4\sum_{i=l}^{k-1}Q_i\sigma_i$. Given $n_l < 5B_{l+1}$, $a_{k+1} < 2B_{k+1}$, $v_{k+1} \leq 2T_{k+1}^{\bullet}$, we have

$$\mathsf{P}(\mathcal{F}_k) \leq 11(B_{k+1}/B_k) \cdot 2(T_{k+1}^{\bullet}/T_k^{\bullet}) \leq 33Q_kU_k'\varepsilon_k.$$

The estimates for $\mathcal{E}_k$ were given in parts 4 and 5.

In the reasoning below, we assume $\neg\mathcal{G}_k$. Between times v_k and v_{k+1}, we have a white germ containing J_l, around the germ cell with address $Q_k/2$. Besides lifting from level k to level $k+1$, the trajectory η^k performs its payload computation; for each level $i \leq k$, this proceeds without faults up to time v_{i+1}, in the area $\Gamma(J_l, a_{i+1})$ which is all the area that matters, as the germ does not grow beyond it. But we are interested in the output of level 1, so trickle-down will be invoked. According to the rules *Compute* of Algorithm 19.7 and *Update-payload* of Algorithm 19.10, payload computation happens on level k representing germ cells of level $k+1$ with addresses in $[0, Q_{k+1})$. We will estimate the number of payload computation steps performed, but first assume that time t' is the end of a dwell period of a germ cell x' of η^k, in which $x = x' + a$ for some $a \in [0, B_k)$, and $\eta^k(x', t').\mathit{Payload}(a)$ is supposed to have the value $\zeta(x, t)$. Under the assumption $\neg\mathcal{G}_k$ indeed $\zeta(x, t) = \eta^k(x', t').\mathit{Payload}(a)$. A trickle-down reasoning just like in Section 5.3, in the proof of Theorem 3.3, with $D_k = T_k^{\bullet}$, as done for information storage in the discussion after Lemma 11.7 (Amplifier), will bring the equality down to level 1:

$$\mathsf{P}\{\zeta(x, t).\mathit{Output} \neq \eta(x, t').\mathit{Output}\} \leq \mathsf{P}(\mathcal{G}_k) + \varepsilon_k + \sum_{i=1}^{k-1}\varepsilon_i''$$

$$= O(\sigma_1) = O(e^{-R}).$$

Let us now estimate t' in terms of t. Let t'_k be the time by which the first white cell of M_k will be created, then we have $v_{k-1} < t'_k \leq v_k$. Between times t'_k and t'_{k+1}, there are enough computation steps to simulate in Tr the $B_{k+1} - B_k$ steps of the expansion of an area of size B_k to one of size B_{k+1}. This shows that if $B_k < t \leq B_{k+1}$ then $t_k \leq t' < t_{k+1}$. Recall the definition of bandwidth w_k in (11.9). During the time interval $(t'_k, t'_{k+1}]$, starting at time t'_k until the payload of a germ of size B_{k+1} is filled, each work period of a germ cell simulates at least a constant times $w_k B_k$ steps of Tr. So if $t > B_k$ then to simulate t steps of Tr we need, within constant factor, $t/B_k w_k$ work periods, each of length $\leq T_k^\bullet$, so

$$t T_k^\bullet / B_k w_k.$$

For $t \leq B_k$ we need less, so we are conservative when going with this estimate for all t. Noting that $\prod_{i<k} U'_i$ is within constant factor of $T_k^\bullet$, and using (11.18), we have within constant factor

$$T_k^\bullet / B_k w_k = R^k (k!)^2 < 2^{2.1 k \log k}.$$

Using (11.18) we have for large t,

$$\begin{aligned}
t > B_k &= R^{k(k+1)/2 - 1}, \\
\log t &> \log R \cdot k^2/2, \\
k &< (\log t)^{1/2} (2/\log R)^{1/2}, \\
2^{2.1 k \log k} &< c^{(\log t)^{1/2} \log \log t},
\end{aligned} \qquad (21.26)$$

for a constant c depending on R. So we can estimate $t' \leq t \cdot c^{(\log t)^{1/2} \log \log t}$ as Theorem 3.4 states.

In case of the finite space $\mathbb{Z}_N$ we can only hope to simulate the computation $\zeta(\cdot, t)$ while it fits into a space of size N, so let

$$K = \bigvee \{ k : B_k < N \}.$$

7. In case of a space $\mathbb{Z}_N$ for finite N, adding $\varepsilon^{h_0(N,c)} t$ to the estimate with a constant $c > 1$ will do. The same argument works also to finish the proof of Theorem 7.5 for a finite space.

Proof. After the level has been raised to K, it will not be raised any longer. In every time period of size $T_K^\bullet$, the computation may fail if a fault of level K occurs, that is with probability ε_K. In time t the probability of this

happening at least once can be (generously) be upper-bounded by $\varepsilon_K t$. We estimate, similarly to (21.24), for $k = K$:

$$\varepsilon_K < \varepsilon^{h_0(N,c)}$$

for an appropriate $c > 1$.

$\square$

22   Some applications and open problems

22.1   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 for example [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 [20] (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 histories of M become the space-configurations of M'. Non-ergodicity of M corresponds to phase transition in M'. In the cellular automaton of Theorem 7.5, each infinite sequence ϱ gives rise to a history storing the bits of ϱ in consecutive cells. Now, each of these histories 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.

22.2   Some open problems

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. By now this question has been answered affirmatively, in [8]. The hierarchical construction shares many similarities with the one of this paper, but there is quite a number of additional details.

Non-hierarchical construction in continuous time   The simple three-dimensional reliable cellular automaton in [19] (with its simpler and stronger proofs in [5] and [16]) relies strongly on being in discrete time. Each plane supposed to store copies of a single symbol needs to switch instantly in each step. In fact, currently no (proved) construction is known in any dimension of a simple reliable cellular automaton in continuous time whose work is non-hierarchical. There is a simple way to simulate a discrete-time cellular automaton by a continuous-time one, described in Section 8 and called the "marching soldiers" scheme, and also used in this paper. However, without the hierarchy, its issues caused by randomness and faults are not yet solved. Besides the possibly large local delays during which the self-correction stalls while the faults continue to occur, in dimensions higher than one the faults can also create deadlocks in the synchronization scheme. An important step was taken by Cook and Winfree with a rule that, under plausible conditions, eliminates these deadlocks: see the report [10]. Further progress depends on the ability to bound the probability of large local delays.

Does a 1-dimensional solution have to be "hierarchical"?   Over the years, there have been several attempts to define a non-ergodic one-dimensional cellular automaton that is "simple". All these attempts ended up similar to [18], which is ergodic (at least under some small noise, as shown in [30]). By "complexity" I don't just refer to the number of states, rather to the fact that the history requires inhomogenities (to exist or to arise) on a larger-and-larger scale. Ideally, we will see a result confirming the intuition that this

sort of complexity is necessary, but as a first step, we need a mathematically formulated conjecture.

Relaxation time as a function of space size Recall relaxation time in Definition 6.10. 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 $\lim_{m\to\infty} r_m(0, 1/3) = \infty$, that is 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 [19], 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 [13] and [14]) 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.

Relaxation time as a function of observed area Lemma 6.11 seems to suggest that the issue of information loss in a cellular automaton is solved by the question of ergodicity for $m = \infty$ where m is the space size. 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 22.1 (The contact process.) The contact process is a one-dimensional *CCA* as defined in Section 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 [26]) 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 [26] 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.

Acknowledgement

Bibliography

[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. 22.1

[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. 1.3

[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. 22.1

[4] E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning ways for your mathematical plays*, Academic Press, New York, 1982. 4.4

[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. 1.1, 1.3, 6.2, 8, 22.2, 22.2

[6] C. Bezuidenhout and G. Grimmett, *The critical contact process dies out*, Annals of Probability **18** (1990), 1462–1482. 22.1

[7] R. E. Blahut, *Theory and practice of error-control codes*, Addison-Wesley, Reading, MA, 1983. 5.15, 5.15

[8] Ilir Çapuni and Peter Gács, *A reliable Turing machine*, Theory of Computing (2021), 1–82, Submitted. 22.2

[9] Matthew Cook, *Universality in elementary cellular automata*, Complex Systems **15** (2004), 1–40. 4.4

[10] Matthew Cook, Peter Gács, and Erik Winfree, *Self-stabilizing synchronization in 3 dimensions (draft)*, Tech. report, Boston University, Department of Computer Science, Boston, MA 02215, 2009, www.cs.bu.edu/faculty/gacs/papers/3Dasync.pdf. 22.2

[11] Paula Gonzaga de Sá and Christian Maes, *The Gács-Kurdyumov-Levin Automaton revisited*, Journal of Statistical Physics **67** (1992), no. 3/4, 607–622. 1.1

[12] 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. 1.1

[13] Peter Gács, *Reliable computation with cellular automata*, Journal of Computer System Science **32** (1986), no. 1, 15–78, Conference version at STOC' 83. 1.2, 3.1, 6.2, 6.2, 10.1, 22.2

[14] _____, *Self-correcting two-dimensional arrays*, Randomness in Computation (Silvio Micali, ed.), Advances in Computing Research (a scientific annual), vol. 5, JAI Press, Greenwich, Conn., 1989, pp. 223–326. 1.2, 3, 5, 22.2

[15] _____, *Deterministic computations whose history is independent of the order of updating*, ArXiv e-prints (1995), 1–15, also www.cs.bu.edu/faculty/gacs/papers/commut.pdf. 8

230

[16] _____ , *A new version of Toom's proof*, Tech. report, Department of Computer Science, Boston University, TR 95-009, Boston, MA 02215, 1995, arXiv:2105.05968. 1.1, 22.2

[17] _____ , *Reliable cellular automata with self-organization*, Journal of Statistical Physics **103** (2001), no. 1/2, 45–267, See also arXiv:math/0003117 [math.PR] and the proceedings of STOC '97. (document)

[18] Peter Gács, Georgii L. Kurdyumov, and Leonid A. Levin, *One-dimensional homogenuous media dissolving finite islands*, Problems of Inf. Transm. **14** (1978), no. 3, 223–226, Translation of the Russian version whose page numbers are 92-96. 1.1, 22.2

[19] 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, Short version in STOC '85. 1.1, 1.3, 3, 22.2, 22.2

[20] 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. 22.1

[21] Lawrence F. Gray, *The positive rates problem for attractive nearest neighbor spin systems on* $\mathbb{Z}$, Z. Wahrscheinlichkeitstheorie verw. Gebiete **61** (1982), 389–404. 1.1, 2.4, 2.4

[22] _____ , *The behavior of processes with statistical mechanical properties*, Percolation Theory and Ergodic Theory of Infinite Particle Systems (Harry Kesten, ed.), Springer Verlag, 1987, pp. 131–167. 1.1

[23] Gene Itkis and Leonid A. Levin, *Fast and lean self-stabilizing asynchronous protocols*, Proc. of the IEEE Symp. on Foundations of Computer Science, 1994, pp. 226–239. 4.3, 5.1

[24] G. L. Kurdyumov, *An example of a nonergodic homogenous one-dimensional random medium with positive transition probabilities*, Soviet Mathematics Doklady **19** (1978), no. 1, 211–214. 1.2

[25] F. Thomson Leighton, *Parallel algorithms and architectures*, Morgan Kaufmann, San Mateo, CA, 1992. 9.6

[26] Thomas M. Liggett, *Interacting particle systems*, Grundlehren der mathematischen Wissenschaften, vol. 276, Springer Verlag, New York, 1985, Reprinted with new postface in 2005. 1.3, 2.4, 2.4, 2.4, 22.1

[27] T. S. Mountford, *A coupling of infinite particle systems*, Journal of Mathematics of Kyoto University **35** (1995), no. 1, 43–52. 6.5

[28] Jacques Neveu, *Bases mathematiques du calcul des probabilités*, Masson et Cie, Paris, 1964. 7.2

[29] Nicholas Ollinger and Gaétan Richard, *Four states are enough!*, Theoretical Computer Science **412** (2011), no. 1-2, 22–32. 4.4, 4.4

[30] Kihong Park, *Ergodicity and mixing rate of one-dimensional cellular automata*, Ph.D. thesis, Boston University, Boston, MA 02215, 1996, See https://cs-web.bu.edu/faculty/gacs/papers/1996-015-park-phdthesis.pdf. 1.1, 22.2

[31] Nicholas Pippenger, *On networks of noisy gates*, Proc. of the 26-th IEEE FOCS Symposium, 1985, pp. 30–38. 1.1

[32] Charles Radin, *Global order from local sources*, Bull. Amer. Math. Soc. **25** (1991), 335–364. 5.1

[33] Daniel A. Spielman, *Highly fault-tolerant parallel computation*, Proc. of the 37th IEEE FOCS Symposium, 1996, pp. 154–163. 1.1

[34] Tommaso Toffoli and Norman Margolus, *Cellular automata machines*, MIT Press, Cambridge, MA., 1987. 4.4

[35] Andrei L. Toom, *Stable and attractive trajectories in multicomponent systems*, Multicomponent Systems (R. L. Dobrushin, ed.), Advances in Probability, vol. 6, Dekker, New York, 1980, Translation from Russian, pp. 549–575. 1.1, 6.1, 6.1, 6.2

[36] Boris S. Tsirel'son, *Reliable information storage in a system of locally interacting unreliable elements*, Locally Interacting Systems and their Application in Biology (V. I. Kryukov R. L. Dobrushin and A. L. Toom, eds.), Lecture Notes in Mathematics 653, Springer, 1978, Translation from Russian., pp. 15–30. 1.2

[37] John von Neumann, *Probabilistic logics and the synthesis of reliable organisms from unreliable components*, Automata Studies (C. Shannon and McCarthy, eds.), Princeton University Press, Princeton, NJ., 1956. 1.1

[38] 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. 1.3, 7.4