# A reliable Turing machine

Ilir Çapuni
University of Montenegro
ilir@bu.edu

Peter Gács
Boston University
gacs@bu.edu

March 24, 2022

**Abstract**

We consider computations of a Turing machine subjected to noise. In every step, the action (the new state and the new content of the observed cell, the direction of the head movement) can differ from that prescribed by the transition function with a small probability (independently of previous such events). We construct a universal 1-tape Turing machine that in such noise with a low enough (constant) noise probability bound, performs arbitrarily large computations. For this unavoidably, the input needs to be encoded—by a simple code depending on its size. The work uses a technique familiar from reliable cellular automata, complemented by some new ones.

# 1 Introduction

This work addresses a question from the area of "reliable computation with unreliable components". A certain class of machines is chosen (like a Boolean circuit, cellular automaton, Turing machine). It is specified what kind of faults (local in space and time) are allowed, and a machine of the given kind is built that—paying some price in performance—carries out essentially the same task as any given machine of the same kind without any faults would.

We confine attention to *transient*, *probabilistic* faults: the fault occurs at a given time but no component is damaged permanently, and faults occur independently of each other, with a bound on their probability. This is in contrast to bounding the *number* of faults, allowing them to be set by an adversary. Historically the first result of this kind is [11], which for each Boolean circuit $C$ of size $n$ constructs a new circuit $C'$ of size $O(n \log n)$ that performs the same task as $C$ with a (constant) high probability, even though each gate of $C'$ is allowed to fail with some (constant) small probability.

Cellular automata as a model have several theoretical advantages over Boolean circuits, and results concerning reliable computation with them have interest also from a purely mathematical or physical point of view (non-ergodicity). The simple construction in [7] gives, for any 1-dimensional cellular automaton $A$ a 3-dimensional cellular automaton $A'$ that performs the same task as $A$ with high probability, even though each cell of $A'$ is allowed to fail in each time step with some constant small probability. (A drawback of this construction is the requirement of synchronization: all cells of $A'$ must update simultaneously.) Reliable cellular automata in less than 3 dimensions can also be constructed (even without the synchrony requirement), but so far only at a steep increase in complexity (both of the construction and the proof). The first such result was [5], relying on some ideas proposed in [8].

Here, the reliability question will be considered for a *serial* computation model—a 1-tape Turing machine—as opposed to parallel ones like Boolean circuits or cellular automata. There is a single elementary processing unit (the *active* unit) interacting with a memory of unlimited size. The error model needs to be relaxed. Allowing each memory component to fail in each time step with constant probability makes, in the absence of parallelism, reliable computation seem impossible. Indeed, while in every step some constant fraction of the memory gets corrupted, the active unit can only correct a constant *number* of them per step.

**Remark 1.1** The choice a single tape for the Turing machine seems unnecessarily restricting, given how time-consuming it is to even compare two strings on such machines. However, having two tapes would add to the physical implausibility, assuming some kind of unlimited-length safe connection from the heads to a processing unit (or

at least between each other). ⌟

In the relaxed model considered here, faults can affect only the operation of the active unit. More precisely at any given time the allowed operations of the machine are the usual ones: changing its state, writing to the observed tape cell, moving the head by a step left or right (or not at all). The transition table of the machine prescribes which action to take. So our fault model is the following.

**Definition 1.2** Let *Noise* be a random subset of some set $U$. We will say that the distribution of *Noise* is $\varepsilon$-*bounded* if for every finite subset $A$ we have

$$\mathsf{P}\{A \subseteq \mathit{Noise}\} \leq \varepsilon^{|A|}.$$

⌟

See [10] for an earlier use of this kind of restriction.

**Definition 1.3** Let $\mathcal{C} = (C_1, C_2 \dots)$ be a random sequence of configurations of a Turing machine $T$ with a given fixed transition table, with the property that for each time $t$, the $C_{t+1}$ is obtained from $C_t$ by one of the allowed operations. We say that a *fault* occurred at time $t$ if the operation giving $C_{t+1}$ from $C_t$ is not obtained by the transition function. Let $\mathit{Noise} \subseteq \mathbb{Z}_+$ be the (random) set of faults in the sequence. We say that faults of the sequence $\mathcal{C}$ are $\varepsilon$-*bounded*, if the set *Noise* is. ⌟

The challenge for Turing machines is still significant, since even if only with small probability, occasionally a group of faults can put the head into the middle of a large segment of the tape rewritten in an arbitrarily "malicious" way. A method must be found to recover from all these situations.

Here we define a Turing machine that is reliable—under this fault model—in the same sense as the other models above. The construction and proof are similar in complexity to the ones for 1-dimensional cellular automata; however, we did not find a reduction to these earlier results. A natural idea is to let the Turing machine simulate a 1-dimensional cellular automaton, by having the head make large sweeps, and update the tape as the simulated cellular automaton would. But apart from the issue of excessive delay, we did not find any simple way to guarantee the large sweeps in the presence of faults (where "simple" means not building some new hierarchy), even for some price paid in efficiency. So here we proceed "from scratch".

Many ideas used here are taken from [5] and [6], but hopefully in a somewhat simpler and more intuitive conceptual framework. Like [5] it confines all probability reasoning to a single lemma, and deals on each level only with a numerical restriction on faults (of that level). On the other hand, like [6] it defines a series of generalized objects (generalized Turing machines here rather than generalized cellular automata),

each one simulating the next in the series. In [6] a "trajectory" (central for defining the notion of simulation) was a random history whose distribution satisfies certain constraints (most of which are combinatorial). Here it is a single history satisfying only some combinatorial constraints.

The work [1] seems related by its title but is actually on another topic. The work [4] applies the self-simulation and hierarchical robustness technique developed for cellular automata in an interesting, but simpler setting. Several attempts at the chemical or biological implementation of universal computation have to deal with the issue of error-correction right away. In these cases generally the first issue is the faults occurring at the active site (the head). See [2, 9].

Our result can use any standard definition of 1-tape Turing machines whose tape alphabet contains some fixed "input-output alphabet" $\Sigma$; we will introduce one formally in Section 2.5. We will generally view a tape symbol as a tuple consisting of several *fields*. The notation

$$a.Output, a.Info$$

shows *Output* and *Info* as fields of tape cell state $a$. Combining the same field of all tape cells, we can talk about a *track* (say the *Output* track and *Info* track). For ease of spelling out a result, we consider only computations whose outcome is a single symbol, written into the *Output* field of tape position 0. It normally holds a special value—say $*$ —meaning *undefined*.

Block codes (as defined in Section 3.2 below) are specified by a pair $(\psi_*, \psi^*)$ of encoding and decoding functions. In the theorem below, the input of the computation, of some length $n$, is broken up into blocks that are encoded by a block code that depends in some simple way on $n$. Its redundancy depends on the size of the input as a log power. The main result in the theorem below shows a Turing machine simulating a fault-free Turing machine computation in a fault-tolerant way. It is best to think of the simulated machine $G$ as some universal Turing machine.

**Theorem 1** *For any Turing machine $G$ there are constants $\alpha_1, \alpha_2 > 0$, for each input size $n$ a block code $(\varphi_*, \varphi^*)$ of block size $O((\log n)^{\alpha_1})$, a fault bound $0 \le \varepsilon < 1$ and a Turing machine $M_1$ with a function $a \mapsto a.Output$ defined on its alphabet, such that the following holds.*

*Let $M_1$ start its work from the initial tape configuration $\varphi_*(x)$ with the head in position 0, running through a random sequence of configurations whose faults are $\varepsilon$-bounded in the sense of Definition 1.3. Suppose that at time $t$ the machine $G$ writes a value $y \neq *$ into the Output field of the cell at position 0. Then at any time greater than $t(\log t)^{\alpha_2 \log\log\log t}$, the tape symbol $a$ of machine $M_1$ at position 0 will have $a.Output = y$ with probability at least $1 - O(\varepsilon)$.*

## 2 Overview

The overview, but even the main text, is not separated completely into two parts, namely the definition of the Turing machine, followed by the proof of its reliability. The definition of the machine is certainly translatable (with a lot of tedious work) into just a Turing machine transition table (or "program"), but its complexity requires first to develop a conceptual apparatus behind it which is also used in the proof of reliability. We will try to indicate below at the beginning of each section whether it is devoted more to the program or more to the conceptual apparatus.

### 2.1 Isolated bursts of faults

Let us introduce some basic elements of the *program*. In [3] we defined a Turing machine $M_1$ that simulates "reliably" any other Turing machine even when it is subjected to isolated "bursts" of faults (that is a group of faults occurring in consecutive time steps) of constant size. We will use some of the ideas of [3], without relying directly on any of its details, and will add several new ideas. Here is a brief overview of this machine $M_1$.

Each tape cell of the simulated machine $M_2$ will be represented by a block of some size $Q$ called a *colony*, of the simulating machine $M_1$. Each step of $M_2$ will be simulated by a computation of $M_1$ called a *work period*. During this time, the head of $M_1$ moves around over the current colony-pair, decodes the represented cell symbols, then computes and encodes the new symbols, and finally moves the head to the new position of the head of $M_2$. The major processing steps will be carried out on a working track three times within one work period, recording the result onto separate tracks. The information track is changed only in a final majority vote.

The organization is controlled by a few key fields, for example a field called *Addr* showing the position of each cell in the colony, and a field *Age*, the number of the last step of the computation that has been performed already. The most technical part is to protect this control information from faults. To discover such structural disruptions locally before the head would go far in the wrong direction, the head will make frequent short zigzags. Any local inconsistency will be detected this way, triggering the healing procedure.

### 2.2 Hierarchy

Here, we start the development of the *conceptual apparatus*. In order to build a machine resisting faults occurring independently in each step with some small probability, we take the approach used for one-dimensional cellular automata. We aim at build-

ing a *hierarchy of simulations*: machine $M_1$ simulates machine $M_2$ which simulates machine $M_3$, and so on. Machine $M_k$ has alphabet

$$\Sigma_k = \{0,1\}^{s_k}, \tag{2.1}$$

that is its tape cells have some "capacity" $s_k$. All these machines should be implementable on a universal Turing machine with the same program (with an extra input, the number $k$ denoting the level). For ease of analysis, we introduce the notion of *cell size*: level $k$ has its own cell size $B_k$ and block (colony) size $Q_k$ with $B_1 = 1$, $B_{k+1} = B_k Q_k$. This allows locating each tape cell of $M_k$ on the same interval where the cells of $M_1$ simulate it. One cell of machine $M_{k+1}$ is simulated by a colony of machine $M_k$; so one cell of $M_3$ is simulated by $Q_1 Q_2$ cells of $M_1$. Further, one step of, say, machine $M_3$ is simulated by one work period of $M_2$ of, say, $O(Q_2^2)$ steps.

Per construction, machine $M_1$ can withstand bursts of faults with size $\leq \beta$ for some constant parameter $\beta$, separated by at least some constant number $\gamma$ of work periods. It would be natural now to expect that machine $M_1$ can withstand also some *additional*, larger bursts of size $\leq \beta Q_1$ if those are separated by at least $\gamma$ work periods of $M_2$. However, a new obstacle arises. Damage caused by a big burst of faults spans several colonies. The repair mechanism of machine $M_1$ outlined in Section 2.1 is too local to recover from such extensive damage, leaving the whole hierarchy endangered. So we add a new mechanism to $M_1$ that will just try to restore the colony *structure* of a large enough portion of the tape (of the extent of several colonies). The task of restoring the original *information* is left to higher levels (whose simulation now can continue).

All machines above $M_1$ in the hierarchy live only in simulation: the hardware is $M_1$. Moreover, the $M_k$ with $k > 1$ will not be ordinary Turing machines, but *generalized* ones, with some new features seeming necessary in a simulated Turing machine: allowing for some "disordered" areas of the tape not obeying the transition function, and occasionally positive distance between neighboring tape cells.

A tricky issue is "forced self-simulation". Each machine $M_k$ can be implemented on a universal machine using as inputs the pair $(p, k)$ where $p$ is the common program and $k$ is the level. Eventually, $p$ will just be hard-wired into the definition of $M_1$, and therefore faults cannot corrupt it. While creating $p$ for machine $M_1$, we want to make it simulate a machine $M_2$ that has the same program $p$. The method to achieve this has been applied already in some of the cellular automata and tiling papers cited, and is related to the proof of Kleene's fixed-point theorem (also called the recursion theorem).

Forced self-simulation can give rise to an infinite sequence of simulations, achieving the needed robustness. Let us point out that fixing the program of self-simulation does not prevent universality. A track (which we will call *Payload*) will be set aside for

simulating the machine $G$ of Theorem 1. If this simulation of $G$ does not finish in a certain number of steps, a built-in mechanism will *lift* its tape content to the *Payload* field of the simulated cell-pair, allowing it to be continued in a colony-pair of the next level (with the corresponding higher reliability).

### 2.3 Structuring the noise

From the probabilistic assumptions about the noise, one can draw some combinatorial conclusions. This part of the work is rather simple and self-contained, and is similar to some earlier publications on these topics.

The set of faults in the noise model of the theorem is a set of points in time. It turns out more convenient to use an equivalent model: an $\varepsilon$-bounded *space-time* set of points. Let us make this statement more formal.

**Lemma 2.1** *Let $\mathcal{C} = (C_1, C_2, \dots)$ be the random sequence of configurations of a Turing machine with an $\varepsilon$-bounded set of faults $Noise_1 \subseteq \mathbb{Z}_+$, as in Definition 1.3. Let $h(t)$ be the (random) position of the head at time $t$. Then the random set $Noise_2 = \{(h(t), t) : t \in Noise_1\}$ is an $\varepsilon$-bounded subset of $\mathbb{Z} \times \mathbb{Z}_+$.*

*Proof.* Let $A$ be a finite subset of $\mathbb{Z} \times \mathbb{Z}_+$, and $A' = \{t : (p, t) \in A\}$. If $A \subseteq Noise_2$ then $A' \subseteq Noise_1$ and $|A'| = |A|$. Hence

$$\mathsf{P}\{A \subseteq Noise_2\} \le \mathsf{P}\{A' \subseteq Noise_1\} \le \varepsilon^{|A'|} = \varepsilon^{|A|}.$$

□

The construction outlined above counts with *bursts* (rectangles of space-time containing *Noise*) increasing in size and decreasing in frequency—which is a combinatorial set of constraints. To derive such constraints from the above probabilistic model the we stratify *Noise* as follows. We will have two series of parameters: $B_1 < B_2 < \cdots$ and $S_1 < S_2 < \cdots$. Here $B_k$ is the size of cells of $M_k$ as represented on the tape of $M_1$, and $S_k$ is a (somewhat increased) bound on the time needed to simulate one step of $M_k$.

Here are some informal definitions. For some constants $\beta, \gamma > 1$, a *burst* of noise of type $(a, b)$ is a space-time set that is coverable by a rectangle of size $a \times b$. For an integer $k > 0$ it is of *level* $k$ when it is of type $\beta(B_k, S_k)$. It is *isolated* if it is essentially alone in a rectangle of size $\gamma(B_{k+1} \times S_{k+1})$ First we remove such isolated bursts of level 1, then of level 2 from the remaining set, and so on. It will be shown that with not too fast increasing sequences $B_k, S_k$, with probability 1, this infinite sequence of operations completely erases *Noise*: thus each fault belongs to a burst of "level" $k$ for some $k$.

Machine $M_k$ will concentrate only on correcting isolated bursts of level $k$ and on restoring the framework allowing $M_{k+1}$ to do its job. It can ignore the lower-level bursts and will need to work correctly only in the absence of higher-level bursts.

If we modeled noise as a set of time points then a burst of faults would be a time interval of size $\beta S_k$ and might affect a space interval as large as $\beta S_k$, covering many times more simulated cells of level $k$. Therefore we model noise as a set of space-time points; by Lemma 2.1, this does not change the independence assumption of the main theorem.

**Definition 2.2** Let $\mathbf{r} = (r_1, r_2)$, $r_1, r_2 > 0$ be a two-dimensional nonnegative vector. A *rectangle of "radius"* $\mathbf{r}$ *centered* at point $\mathbf{x}$ is

$$\mathbf{B}(\mathbf{x}, \mathbf{r}) = \{\mathbf{y} : |y_i - x_i| < r_i, i = 1, 2\}. \tag{2.2}$$

Let $E \subseteq \mathbb{Z} \times \mathbb{Z}_{\geq 0}$ be a space-time set (to be considered our noise set). A point $\mathbf{x}$ of $E$ is $(\mathbf{r}, \mathbf{r}^*)$-*isolated* if $E \cap \mathbf{B}(\mathbf{x}, \mathbf{r}^*) \subseteq \mathbf{B}(\mathbf{x}, \mathbf{r})$, that is all points of $E$ that are $\mathbf{r}^*$-close to $x$ are also $\mathbf{r}$-close. A set $E$ is called $(\mathbf{r}, \mathbf{r}^*)$-*sparse* if each of its points is $(\mathbf{r}, \mathbf{r}^*)$-isolated. ⌟

The following lemma will justify talking about bursts of faults.

**Lemma 2.3** (Bursts) *Suppose that the set $E$ is $(\mathbf{r}, \mathbf{r}^*)$-sparse with $\mathbf{r}^* > 2\mathbf{r}$. For an element $\mathbf{x} \in E$ let $E_\mathbf{x} = \mathbf{B}(\mathbf{x}, \mathbf{r}) \cap E$. Each set $E_\mathbf{x}$ is contained in a rectangle of size $r_1 \times r_2$. Every rectangle of size $(r_1^* - r_1) \times (r_2^* - r_2)$ intersects with at most one of the sets $E_\mathbf{x}$.*

*Proof.* We introduce a relation $\mathbf{x} \sim \mathbf{y} \Leftrightarrow \mathbf{y} \in E_\mathbf{x}$ between elements of the set $E$. The relation is clearly symmetric and reflexive, but we claim that it is also transitive. Indeed, suppose that $\mathbf{y} \sim \mathbf{x}$ and $\mathbf{y}' \sim \mathbf{x}$, Given that $\mathbf{r} \leq 2\mathbf{r}^*$, we have $\mathbf{y}' \in \mathbf{B}(\mathbf{y}, \mathbf{r}^*)$, and then by sparsity, $\mathbf{y}' \in \mathbf{B}(\mathbf{y}, \mathbf{r})$.

By the relation $\sim$ we can partition the set $E$ into subsets of the form $E_\mathbf{x}$. We claim that each of these sets is coverable by a rectangle of size $r_1 \times r_2$. Indeed, suppose this is not so: then either the horizontal projection of $E_\mathbf{x}$ is $\geq r_1$ or the vertical one is $\geq r_2$: suppose the former. Then there are elements $\mathbf{y}, \mathbf{y}' \in E_x$ whose horizontal distance is $\geq r_1$ contrary to $\mathbf{y}' \sim \mathbf{y}$.

Suppose that some rectangle of size $(r_1^* - r_1) \times (r_2^* - r_2)$ intersects $E_\mathbf{x}$ and $E_\mathbf{y}$. Then $\mathbf{B}(\mathbf{x}, \mathbf{r}^*)$ contains both $\mathbf{x}$ and $\mathbf{y}$, hence by sparsity $\mathbf{y} \in \mathbf{B}(\mathbf{x}, \mathbf{r}^*)$ and $E_\mathbf{x} = E_\mathbf{y}$. □

**Definition 2.4** Let $\gamma > 1$, $\beta \geq 3\gamma$ be parameters, and let

$$1 = B_1 < B_2 < \cdots, \quad 1 = S_1 < S_2 < \cdots,$$
$$S_{k+1}/S_k,\ B_{k+1}/B_k \geq 2\beta$$

be sequences of integers to be fixed later. For a space-time set $E \subseteq \mathbb{Z}\times\mathbb{Z}_{\geq 0}$, let $E^{(1)} = E$. For $k > 1$ let $E^{(k+1)}$ be obtained by deleting from $E^{(k)}$ the $(\beta(B_k, S_k), \gamma(B_{k+1}, S_{k+1}))$-isolated points. Set $E$ is *k-sparse* if $E^{(k+1)}$ is empty. It is simply *sparse* if $\bigcap_k E^{(k)} = \emptyset$. When $E = E^{(k)}$ and $k$ is known then we will denote $E^{(k+1)}$ simply by $E^*$. ⌟

**Definition 2.5** (Burst) Suppose that the set $E$ is $(\mathbf{r}, \mathbf{r}^*)$-sparse with $\mathbf{r}^* > 2\mathbf{r}$. By the above lemma, it is partitioned into subsets of the form $E_{\mathbf{x}}$. In what follows we will call these sets *bursts*. The definition depends on the parameter $\mathbf{r}$ which will always be clear from the context. Typically, it will be $\beta(B, S)$ where $\beta$ is a constant, $B = B_k$ and $S = S_k$ for some $k$ called the *level*. ⌟

The following lemma connects the above defined sparsity notions to the requirement of small fault probability.

**Lemma 2.6** (Sparsity) *Let $Q_k = B_{k+1}/B_k$, $V_k = S_{k+1}/S_k$, and*

$$\lim_{k\to\infty} \frac{\log Q_k V_k}{1.5^k} = 0. \tag{2.3}$$

*For sufficiently small $\varepsilon$, for every $k \geq 1$ the following holds. Let $E \subseteq \mathbb{Z}\times\mathbb{Z}_{\geq 0}$ be a random set that is $\varepsilon$-bounded as in Definition 1.3. Then for each point $\mathbf{x}$ and each $k$,*

$$\mathsf{P}\{\mathbf{B}(\mathbf{x}, (B_k, S_k)) \cap E^{(k)} \neq \emptyset\} < \varepsilon \cdot 2^{-1.5^{k-1}}.$$

*As a consequence, the set $E$ is sparse with probability 1.*

This lemma allows a doubly exponentially increasing sequence $S_k$, resulting in relatively few simulation levels as a function of the computation time.

## 2.4 Difficulties

We list here some of the main problems that the paper deals with, and some general ways in which they will be solved or avoided. Some more specific problems will be pointed out later, along with their solution.

*Non-aligned colonies* A large burst of faults in $M_1$ can modify the order of entire colonies or create new ones with gaps between them. To deal with this problem, machines $M_k$ for $k > 1$ will be *generalized Turing machines*, allowing for non-adjacent cells.

*Clean areas* On the tape of a generalized Turing machine, based on its content, some areas will be called *clean*, the rest *disordered*. In clean areas, the analysis can count on an existing underlying simulation, and therefore the transition function is applicable. Noise can disorder the areas where it occurs.

*Extending cleanness* The predictability of the machine is decreased when the head enters into disorder. But the model still provides some "magical" properties helping to restore cleanness (in the absence of new noise):

A) escaping from any area in a bounded amount of time;

B) shrinking disorder, as the head passes in and out of it;

C) cleaning an interval when passed over a certain number of times.

While an area is cleaned, it will also be re-populated with cells. Their content is not important, what matters is the restoration of predictability.

*Rebuilding* The need to reproduce these cleaning properties in simulation is the main burden of the construction. The part of the program devoted to this is the rebuilding procedure, invoked when local repair fails. It reorganizes a part of the tape having the size of a few colonies.

## 2.5 Generalized Turing machines

This section, together with Section 2.7, introduces the key concepts used in the proof. Let us recall that a one-tape Turing machine is defined by a finite set $\Gamma$ of *internal states*, a finite alphabet $\Sigma$ of *tape symbols*, a transition function $\delta$, and possibly some distinguished states and tape symbols. At any time, the head is at some integer position $h$, and is observing the tape symbol $A(h)$. The meaning of $\delta(a,q) = (a',q',d)$ is that if $A(h) = a$ and the state is $q$ then the $A(h)$ will be rewritten as $a'$ and $h$ will change to $h+d$.

We will use a model that is slightly different, but has clearly the same expressing power. (Its advantage is that it is a little more convenient to describe its simulations.) There are no internal states, but the head observes and modifies a *pair* of neighboring tape cells at a time; in fact, we imagine it to be positioned between these two cells called the *current cell-pair*. The *current cell* is the left element of this pair. Thus, a Turing machine is defined as $(\Sigma,\tau)$ where the tape alphabet $\Sigma$ contains at least the distinguished symbols $\text{␣}, 0, 1$ where ␣ is called the *blank symbol*. The *transition function* is $\tau\colon \Sigma^2 \to \Sigma^2 \times \{-1,1\}$. A *configuration* is a pair $\xi = (A,h) = (\xi.\text{tape}, \xi.\text{pos})$ where $h \in \mathbb{Z}$ is the *current* (or *observed)* head position, (between cells $h$ and $h$+1), and $A \in \Sigma^{\mathbb{Z}}$ is the *tape content*, or *tape configuration*: in cell $p$, the tape contains the symbol $A(p)$. Though the tape alphabet may contain non-binary symbols, we will restrict input and output to binary. The tape is blank at all but finitely many positions.

As the head observes the pair of tape cells with content $\mathbf{a} = (a_0, a_1)$ at positions $h$, $h+1$ denote $(\mathbf{a}', d) = \tau(\mathbf{a})$. The transition $\tau$ will change the tape content at positions $h$, $h+1$ to $a_0'$, $a_1'$, and move the head to tape position to $h+d$. A *fault* occurs at time $t$

if the output $(\mathbf{a}', d)$ of the transition function at this time is replaced with some other value (which then defines the next configuration).

**Remark 2.7** The informal description of the simulation program below is in some places written as if there was such a thing as an internal state. But this can clearly be implemented for example as follows. In the current cell-pair, one element will always be marked as the one carrying an "internal state", and a field of this cell can be used to represent this state. Anticipating the move of the head to left or right, the transition may move this mark (along with the changed state), if needed, to the other element of the cell-pair. ⌟

The machines that occur in simulation will be a generalized version of the above model, allowing non-adjacent cells and areas called "disordered" in which the transition function is non-applicable. As a convenience feature, two integer parameters are added: the cell body size $B \geq 1$ and and an upper bound $T \geq 1$ on the transition time. These allow placing all the different Turing machines in a hierarchy of simulations onto the same space line and the same time line.

**Definition 2.8** (Generalized Turing machine) A *generalized Turing machine* $M$ is defined by a tuple

$$(\Sigma, \tau, \text{Vac}, \text{New}, \text{Bad}, B, T, \pi, q), \tag{2.4}$$

where $\Sigma$ is the *alphabet*, and

$$\tau : \Sigma^2 \times \{\text{True}, \text{False}\} \to \Sigma^2 \times \{-1, 1\}.$$

is the *transition function*. In $\tau(a, b, \alpha)$ the argument $\alpha$ is True if the pair of observed cells is adjacent (no gap between them), and False otherwise. Among the elements of the tape alphabet we distinguish the input-output element 0, 1, a special symbols Vac, Bad and a subset New.

- Vac plays the role of a blank symbol (the absence of a cell).
- The symbol Bad marks *disordered* areas of the tape.
- The state of newly created cells is in New.
- The parameters $B, T$ were discussed above. The integer $\pi$ will play the role of the number of passes needed to clean an area (see below). The positive real $q$ will help upper-bound the escape time from a disordered area. The parameter $S_k$ used in structuring the noise will be specified in Definition 2.14.

The transition function $\tau$ has no inputs or outputs that are Bad or Vac.

⌟

The effect of the transition function on configurations will be explained in Definition 2.11.

**Remark 2.9** Let $\tau(a, b, \alpha) = (a', b', d)$. We will have $a' \in$ New or $b' \in$ New only if $a, b \notin$ New and $\alpha =$ False, that is the observed cells are not adjacent. Even then we can have $a' \in$ New only if $d = -1$ and $a' \in$ New only if $d = 1$. ⌟

A formal definition of a configuration of a generalized Turing machine is given in Section 3.4, though it is essentially defined by the tape content $A$ and the head position. A point $p$ is *clean* if $A(p) \neq$ Bad. A set of points is *clean* if it consists of clean points. We say that there is a *cell* at a position $p \in \mathbb{Z}$ if the interval $p + [0, B)$ is clean, $A(p) \neq$ Vac and all other elements of this interval are vacant. In this case, we call the interval $p + [0, B)$ the *body* of this cell. Thus, cell bodies must not intersect. If their bodies are at a distance $< B$ from each other, with a clean interval containing both, then they are called *neighbors*. They are called *adjacent* if this distance is 0.

A sequence of configurations conceivable as a computation will be called a "history". For standard Turing machines, the histories that obey the transition function could be called "trajectories". For generalized Turing machines the definition of trajectories is more complex; it allows some limited violations of the transition function, while providing the mechanisms for eliminating disorder. Let $Noise \subseteq \mathbb{Z} \times \mathbb{Z}_{\geq 0}$ denote the set of space-time points at which faults occur. Section 2.7 below will define a certain subset of possible histories called *trajectories*. In order to motivate their choice, we first introduce the notion of simulation.

## 2.6 Simulation

The notion of simulation used in the proof and introduced here, relies on a certain concept of trajectories. On the other hand, the simulation concept helps motivate Section 2.7 where trajectories will be defined.

Until this moment, we used the term "simulation" informally, to denote a correspondence between configurations of two machines which remains preserved during the computation. In the formal definition, this correspondence will essentially be a code $\varphi = (\varphi_*, \varphi^*)$. The *decoding* part of the code is the more important one. We want to say that machine $M_1$ simulates machine $M_2$ via simulation $\varphi$ if whenever $(\eta, Noise)$ is a trajectory of $M_1$ then $(\eta^*, Noise^*)$, defined by $\eta^*(\cdot, t) = \varphi^*(\eta(\cdot, t))$, is a trajectory of $M_1$. Here, $Noise^*$ is computed by the residue operation (deleting isolated elements) as in Definition 2.4. We will make, however, two refinements. First, we require the above condition only for those $\eta$ for which the initial configuration $\eta(\cdot, 0)$ has been obtained by encoding, that is it has the form $\eta(\cdot, 0) = \varphi_*(\xi)$. Second, to

avoid the transitional ambiguities in a history, we define the simulation decoding as a mapping $\Phi^*$ between *histories*, not just configurations: $\Phi^*(\eta, Noise) = (\eta^*, Noise^*)$.

**Definition 2.10** (Simulation) Let $M_1, M_2$ be two generalized Turing machines, and let $\varphi_* : \text{Configs}_{M_2} \to \text{Configs}_{M_1}$ be a mapping from configurations of $M_2$ to those of $M_1$, such that it maps starting configurations into starting configurations. Let $\Phi^* : \text{Histories}_{M_1} \to \text{Histories}_{M_2}$ be a mapping. The pair $(\varphi_*, \Phi^*)$ is called a *simulation* (of $M_2$ by $M_1$) if for every trajectory $(\eta, Noise)$ of $M_1$ with initial configuration $\eta(\cdot, 0) = \varphi_*(\xi)$, the history $(\eta^*, Noise^*) = \Phi^*(\eta, Noise)$ is a trajectory of machine $M_2$. ⌟

In the noise-free case it is easy to find examples of simulations. However, in the cases with noise, finding any nontrivial example is a challenge, and depends on a careful definition of trajectories for generalized Turing machines.

## 2.7 Trajectories

This section completes the definition of the central concept of the proof—modulo the natural definitions spelled out in Section 3.4. A history of a generalized Turing machine $M$ is a trajectory if it obeys certain constraints on its fault-free parts. We discuss these properties first informally.

*Transition Function* This property says—in more precise terms—that in a clean area, the transition function is obeyed.

*Spill Bound* limits the extent to which a disordered interval can spread.

*Escape* limits the time for which the head can be trapped in a small area.

*Attack Cleaning* erodes disorder as the head repeatedly enters and leaves it.

*Pass Cleaning* cleans the interior of an interval if the head passes over it enough times.

The definition below depends on the notions of current cell-pair, switch and dwell period given in in Section 3.4, but should be understandable as it is.

**Definition 2.11** (Transition) Suppose that at times $t'$ before a switching time $t$ but after any previous switch, the current cell-pair $(x, y)$ has state $\mathbf{a} = (a, b)$. Let $(a', b', d) = \tau(a, b, \alpha)$, where $\alpha =$ True if the cell-pair is adjacent and False otherwise. Let $u, v$ be the states of the cells $x, y$ after the transition, and let $x', y'$ be the new current cell pair. We say that the switch is *dictated by the transition function* if the following holds. We state the conditions for $d = 1$, the case $d = -1$ is analogous.

- $u = a'$.
- Suppose $b' \notin$ New; then $v = b'$, $x' = y$. If cell $y$ has a neighbor $z$ on the right then $y' = z$. Else a new adjacent neighbor $z$ is created on the right of $y$ with a state in New, and again $y' = z$.

- Suppose $b' \in$ New (in which case $\alpha =$ False and $x, y$ are not adjacent, see Remark 2.9). Then $v =$ Vac (cell $y$ is erased), $x' = x$, and a cell $y'$ adjacent to $x$ on the right is created with state $b'$. We will say that cell $y$ is *replaced* with the new cell $y'$, and call this a *replacement situation*.

⌟

As a consequence of this definition, new cells are created automatically when the head would step onto a vacant area, and whenever a cell is "killed" another one is created automatically in a place overlapping with its body.

We will use the following constants:

$$c_{\text{Rebuild}} = 8, \ c_{\text{spill}} = c_{\text{marg}} = c_{\text{Rebuild}}, \tag{2.5}$$

and we will assume

$$\gamma > 4(c_{\text{marg}} + c_{\text{spill}}), \tag{2.6}$$

where $\gamma$ was used in Definition 2.4 (sparsity). (Even though we set all these constants to the same value, it helps clarity to give them separate names.)

**Definition 2.12** For a set $K$ on the line, and some real $c$ let us define its *c-interior* $\text{Int}(K, c)$ as the set of those points of $K$ that are at a distance $\geq c$ from its complement. For an interval $K = [a, b)$, this is $[a + c, b - c)$. In this case, will use it also with negative $c$; then "interior" is really an extended neighborhood of $I$. ⌟

In the following definition, it is important to keep in mind the difference between noise and disorder. A noise-free space-time rectangle can very well contain disordered areas on the tape.

**Definition 2.13** (Trajectory) A history $(\eta, \mathit{Noise})$ of a generalized Turing machine (2.4) with $\eta(t) = (A(t), h(t), \hat{\mathbf{h}}(t))$ is called a *trajectory* of $M$ if the following conditions hold, in any noise-free space-time interval $I \times J$.

*Transition Function* Consider a switch, where the current cell-pair $\hat{\mathbf{h}}$ is inside a clean area, by a distance of at least $2.5B$. Then the new state of the current cell-pair and the direction towards the new current head position are dictated by the transition function. The only change on the tape occurs on the interval enclosing the new and old current cells. Further, the length of the dwell period before the switch is bounded by $T$.

*Spill Bound* A clean interval can shrink by at most $c_{\text{spill}}B$.

*Escape* The head will leave any interval of size $\leq \gamma B$ within time $qT$.

*Attack Cleaning* Suppose that the current cell-pair $(x, x')$ is at the right end of a clean interval $[a, b)$ of size $\geq (c_{\text{spill}} + 2)B$, with the head at position $x$. Suppose further that the transition function directs the head right, and not in a replacement situation of Definition 2.11. Then by the time the head comes back to $x - (c_{\text{spill}} + 1)B$, the clean area is extended to the right by at least $B$. Similarly when "left" and "right" are interchanged.

*Pass Cleaning* Suppose that the head makes at least $\pi$ (left-right, right-left) pairs of passes over an interval $I$. Then at some time during this, the interior $\text{Int}(I, c_{\text{marg}}B)$ of $I$ becomes clean.

⌟

Recall that we will have a hierarchy of simulations $M_1 \longrightarrow M_2 \longrightarrow \cdots$ where machine $M_k$ simulates machine $M_{k+1}$. Our construction will set $\pi = 5k + O(1)$ for $M_k$. This can be interpreted as saying that each 5 pairs of passes raise the "organization level": $5k + O(1)$ passes achieve cleanness on the level of $M_k$.

## 2.8 Scale-up

Above, we have set up the conceptual structure of the construction and the proof. Here are some of the parameters:

**Definition 2.14** Let

$$\begin{aligned} Q_k &= c_Q \cdot 2^{1.2^k}, \\ \pi_k &= 5k + c_\pi, \\ q_k &= c_{\text{esc}} Q_{k-1} \pi_{k-1}, \\ U_k &= c_U Q_k \pi_k^9, \end{aligned}$$

for appropriate constants $c_i > 0$. These sequences clearly satisfy (2.3), and define $B_k = B_1 \prod_{i<k} Q_i$, $T_k = T_1 \prod_{i<k} U_i$, $S_k = T_k q_k$, so

$$V_k = S_{k+1}/S_k = U_k q_k / q_{k-1},$$

satisfying (2.3). When we write for example $Q = Q_k$ then we can write $Q^* = Q_{k+1}$. ⌟

The main remaining part is the definition of the simulation program and the decoding $\Phi^*$, and the proof that with this program, the properties of a trajectory $(\eta, \mathit{Noise})$ of machine $M = M_k$ imply that the history $\Phi^*(\eta, \mathit{Noise}) = (\eta^*, \mathit{Noise}^*)$ obeys the same trajectory requirements on the next level. The program is described in Sections 4-5. The most combinatorially complex part of the proof of trajectory properties is in Section 7, bounding and eliminating disorder on the next level. Section 8 wraps up the proof of the main theorem.

# 3 Some formal details

We give here some details that were postponed from the overview section.

## 3.1 Examples

Examples given in Section 11 motivate various complexities of the construction and proof. Some of them may not be completely understandable without the details of the following program: refer to them when wondering about the necessity for some feature. In all examples where a "burst" is mentioned, it is understood in the sense of Definition 2.5, as a space-time set of faults covered by a rectangle of a certain ("small") size. A burst of "level" $k$ is also referred to as a burst (of faults) of machine $M_k$. If $k$ is fixed and we just talk about a machine $M = M_k$ and its history $\eta$ then we will also refer to a burst of $\eta$.

A good-sized neighborhood of the head will contain enough information to prevent a burst from pushing the computation from one phase to another, wrong one. There will also be some reasons for which even a non-constant size neighborhood will need to be checked repeatedly. For this, the head will proceed in zigzags: every step advancing the head in the simulation is followed by some $Z$ steps of going backward and forward again (with parameter $Z$ chosen appropriately below), checking consistency (and starting a healing process if necessary). This will also enable the head to progress into a large disordered area, without being easily fooled into going away.

Example 11.1 raises a problem. The device by which we will mitigate the effect of this kind of capturing is another property of the movement of the head which will call *feathering*: if the head turns back from a tape cell then next time it must go beyond. This requires a number of adjustments to the program (see later).

Examples 11.2 and 11.3 show that disorder may not be eliminated by a bounded number of noise-free slides over it. Our construction will ensure that, on the other hand, $O(k)$ passes (free of $k$-level noise) will restore organization to level $k$. This property of the construction is incorporated into our definition of a generalized Turing machines as the "magical" property (C) above.

## 3.2 Codes

The input of our computation will be encoded by some error-correcting code, to defend against the possibility of losing information even at the first reading.

**Definition 3.1** (Codes) Let $\Sigma_1, \Sigma_2$ be two finite alphabets. A *block code* is given by a positive integer $Q$—called the *block size*—and a pair of functions

$$\psi_* : \Sigma_2 \to \Sigma_1^Q, \quad \psi^* : \Sigma_1^Q \to \Sigma_2$$

with the property $\psi^*(\psi_*(x)) = x$. Here $\psi_*$ is the encoding function (possibly introducing redundancy) and $\psi^*$ is the decoding function (possibly correcting errors). The code is extended to (finite or infinite) strings by encoding each letter individually:

$$\psi_*(x_1, \ldots, x_n) = \psi_*(x_1) \cdots \psi_*(x_n).$$

⌟

## 3.3 Proof of the sparsity lemma

*Proof of Lemma 2.6.* The proof uses slightly more notation than it would if we simply assumed independence of faults at different space-time sites, but it is essentially the same.

Let $\mathcal{E}_k(\mathbf{x})$ be the event $\mathbf{B}(\mathbf{x}, (B_k, S_k)) \cap E^{(k)} \neq \emptyset$. Let $\mathcal{M} = \mathcal{M}_k(\mathbf{x})$ be the set of minimal sets $A \subseteq \mathbb{Z} \times \mathbb{Z}_+$ with $A \subseteq E \Rightarrow \mathcal{E}_k(\mathbf{x})$.

**Claim 3.2** *Each set in $\mathcal{M}_k(\mathbf{x})$ is contained in $\mathbf{B}(\mathbf{x}, 1.5\gamma(B_k, S_k))$.*

*Proof.* The statement is clearly true for $k = 1$. Suppose it is true for $k$, let us prove it for $k + 1$. The event $\mathcal{E}_{k+1}(\mathbf{x})$ holds if and only if for some $\mathbf{x}' \in \mathbf{B}(\mathbf{x}, (B_{k+1}, S_{k+1})) \cap E$ there is some point $\mathbf{y}$ in

$$E^{(k)} \cap \mathbf{B}(\mathbf{x}', \gamma(B_{k+}, S_{k+1})) \setminus \mathbf{B}(\mathbf{x}', \beta(B_k, S_k)).$$

Then there is some minimal set $A'$ with the property $A' \subseteq E \Rightarrow \mathcal{E}_k(\mathbf{y})$. By the inductive assumption these sets are contained in $\mathbf{B}(\mathbf{y}, 2\gamma(B_k, S_k))$. All the minimal sets $A$ with $A \subseteq E \Rightarrow \mathcal{E}_{k+1}(\mathbf{x})$ have the form $A' \cup \{\mathbf{x}'\}$ with some such $\mathbf{x}'$ and $A'$. Also

$$\begin{aligned} A &\subseteq \mathbf{B}(\mathbf{x}', (\gamma B_{k+1} + 2\gamma B_k, \gamma S_{k+1} + 2\gamma S_k)) \\ &\subseteq \mathbf{B}(\mathbf{x}, (\gamma B_{k+1} + (2\gamma + 1)B_k, \gamma S_{k+1} + (2\gamma + 1)S_k)) \subseteq \mathbf{B}(\mathbf{x}, 1.5\gamma(B_{k+1}, S_{k+1})), \end{aligned}$$

assuming $S_{k+1}/S_k > 6$, $B_{k+1}/B_k > 6$. □

Let $f_k(\mathbf{x}) = \sum_{A \in \mathcal{M}} \varepsilon^{|A|}$. By the union bound we have $\mathsf{P}(\mathcal{E}_k(x)) \leq f_k(\mathbf{x})$.

Let $p_k = \varepsilon \cdot 2^{-1.5^{k-1}}$. We will prove $f_k(\mathbf{x}) < p_k$ by induction. For $k = 1$, rectangles $\mathbf{B}(\mathbf{x}_i, (B_1, S_1))$ have size 1, so by the $\varepsilon$-boundedness, $f_1(\mathbf{x}) < \varepsilon$. Assume that the statement holds for $k$, we will prove it for $k + 1$.

Suppose $\mathbf{y} \in E^{(k)} \cap \mathbf{B}(\mathbf{x}, (B_{k+1}, S_{k+1}))$. According to the definition of $E^{(k)}$, there is a point

$$\mathbf{z} \in \mathbf{B}(\mathbf{y}, \gamma(B_{k+1}, S_{k+1})) \cap E^{(k)} \setminus \mathbf{B}(\mathbf{y}, \beta(B_k, S_k)). \tag{3.1}$$

Consider a standard partition of space-time into rectangles $K_p = \mathbf{B}(\mathbf{c}_p, (B_k, S_k))$. Let

$$I = \{p : K_p \cap \mathbf{B}(\mathbf{x}, \gamma(B_{k+1}, S_{k+1})) \neq \emptyset\}.$$

We are only interested in rectangles $K_p$ with $p \in I$. Let

$$K'_p = \mathbf{B}(\mathbf{c}_p, (2\gamma B_k, 1.5\gamma S_k)).$$

If $K_i, K_j$ are the rectangles in this partition containing $\mathbf{y}$ and $\mathbf{z}$, then $K'_i \cap K'_j = \emptyset$. This follows from the fact that $|y_1 - z_1| > \beta B_k$, $|y_2 - z_2| > \beta S_k$, and $\beta \geq 3\gamma$ in Definition 2.4. The event $\mathbf{y} \in E^{(k)}$ can be written as $\bigcup_{A \in \mathcal{M}_k(\mathbf{y})} \{A \subseteq E\}$, and by Claim 3.2 we have $A \subseteq \mathbf{B}(\mathbf{y}, 1.5\gamma(B_k, S_k)) \subseteq K'_i$ for each $A \in \mathcal{M}_k(\mathbf{y})$. Similarly for $\mathbf{z}$ and $K'_j$. Let $\mathcal{M}(i) = \bigcup_{\mathbf{y} \in K_i} \mathcal{M}_k(\mathbf{y})$, then each set $A \in \mathcal{M}(i)$ is in $K'_i$. The disjointness of $K'_i$ and $K'_j$ and the inductive assumption implies

$$\begin{aligned} f_{k+1}(\mathbf{x}) &\leq \sum_{i,j \in I, K'_i \cap K'_j = \emptyset} \sum_{A \in \mathcal{M}(i), A' \in \mathcal{M}(j)} \varepsilon^{|A|+|A'|} = \sum_{i,j \in I, K'_i \cap K'_j = \emptyset} f_k(\mathbf{c}_i) f_k(\mathbf{c}_j) \\ &\leq |I|^2 p_k^2 = |I|^2 \varepsilon^2 2^{-1.5^k} \cdot 2^{-0.5 \cdot 1.5^{k-1}} = p_{k+1} \varepsilon |I|^2 2^{-0.5 \cdot 1.5^{k-1}}. \end{aligned} \tag{3.2}$$

We have $|I| \leq (2\gamma Q_k + 1)(2\gamma V_k + 1)$. Since $\lim_k \frac{\log V_k Q_k}{1.5^k} = 0$, the multiplier of $p_{k+1}$ in (3.2) is $\leq 1$ for sufficiently small $\varepsilon$. □

## 3.4 Configuration, history

A configuration, as defined below, contains a pair of positions $\hat{\mathbf{h}} = (\hat{h}_0, \hat{h}_1)$ called the *current cell-pair*: In difference to the Turing machines of Section 2.5, the position of the head may not be exactly between the current cells: this allows the model to fit into the framework where a generalized Turing machine $M^*$ is simulated by some (possibly generalized) Turing machine $M$. The head $h$ of $M$—made equal to that of $M^*$—may oscillate inside and around the current cell-pair of $M^*$.

**Definition 3.3** (Configuration) A *configuration* $\xi$ of a generalized Turing machine (2.4) is a tuple

$$(A, h, \hat{\mathbf{h}}) = (\xi.\text{tape}, \xi.\text{pos}, (\xi.\text{cur-cell}_0, \xi.\text{cur-cell}_1))$$

where $A : \mathbb{Z} \to \Sigma$ is the tape, $h \in \mathbb{Z}$ is the head position, $\hat{\mathbf{h}} \in \mathbb{Z}^2$ is the current cell-pair. We have $A(p) = \text{Vac}$ in all but finitely many positions $p$. Whenever the interval $h + [-4B, 4B)$ is clean the current cell-pair must be within it. Let

$$\text{Configs}_M$$

denote the set of all possible configurations of a Turing machine $M$. ⌟

The above definitions can be localized to define a configuration over a space interval $I$ containing the head.

**Definition 3.4** (History) For a generalized Turing machine (2.4), consider a sequence $\eta = (\eta(0, \cdot), \eta(1, \cdot), \dots)$, of configurations along with a noise set *Noise*. Let $h(t) = \eta(t, \cdot).\text{pos}$ be the head position.

A *switching time* is a noise-free time when any part of $\eta$ other than $h(t)$ changes ($h(t)$ is also allowed to change at non-switching times). A *dwell period* is the interval between any consecutive pair of switching times with the property that the space-time rectangle containing them and the head is clean and noiseless.

The pair $(\eta, \mathit{Noise})$ will be called a *history* of machine $M$ if the following conditions hold.

- $|h(t) - h(t')| \le |t' - t|$.
- In two consecutive configurations, the tape content $A(p, t)$ of the positions $p$ not in $h(t) + [-2B, 2B)$ remains the same.
- At each noise-free switching time the head is on the new current cell-pair: $\hat{h}_0(t) = h(t)$. (In particular, when at a switching time a current cell becomes Vac, the head must already be elsewhere.)
- The length of dwell periods is at most $T$.

The above definition can be localized to define a history $I \times J$ containing the head. Let

$$\text{Histories}_M$$

denote the set of all possible histories of $M$. ⌟

## 3.5 Hierarchical codes

Recall the notion of a code in Definition 3.1 and of configuration in Definition 3.3.

**Definition 3.5** (Code on configurations) Consider two generalized Turing machines $M_1, M_2$ with the corresponding alphabets and transition functions, where $B_2/B_1$ is an integer denoted $Q = Q_1$. Assume that a block code $\psi_* : \Sigma_2 \to \Sigma_1^Q$ is given, with an

appropriate decoding function, $\psi^*$. Symbol $a \in \Sigma_2$ is interpreted as the content of some tape square. This block code gives rise to a *code on configurations*, that is a pair of functions

$$\varphi_* : \text{Configs}_{M_2} \to \text{Configs}_{M_1}, \quad \varphi^* : \text{Configs}_{M_1} \to \text{Configs}_{M_2}$$

that encodes some (initial) configurations $\xi$ of $M_2$ into configurations of $M_1$: each cell of $M_2$ is encoded into a colony of $M_1$ occupying the same interval. Formally, assuming $\xi.\text{cur-cell}_j = \xi.\text{pos} + (j-1)B_2$, $j = 0, 1$ we set $\varphi_*(\xi).\text{pos} = \xi.\text{pos}$, $\varphi_*(\xi).\text{cur-cell}_j = \varphi_*(\xi).\text{pos} + (j-1)B_1$, and for all $i \in \mathbb{Z}$,

$$\varphi_*(\xi).\text{tape}(iB_2, iB_2 + B_1, \ldots, (i+1)B_2 - B_1) = \psi_*(\xi.\text{tape}(i)).$$

⌟

**Definition 3.6** (Hierarchical code) For $k \geq 1$, let $\Sigma_k$ be an alphabet, of a generalized Turing machine $M_k$. Let $Q_k = B_{k+1}/B_k$ be an integer (viewed as colony size), let $\varphi_k$ be a code on configurations defined by a block code

$$\psi_k : \Sigma_{k+1} \to \Sigma_k^{Q_k}$$

as in Definition 3.5. The sequence $(\Sigma_k, \varphi_k)$, $(k \geq 1)$, is called a *hierarchical code*. For this hierarchical code, configuration $\xi^1$ of $M_1$ is called a *hierarchical code configuration* of height $k$ if a sequence of configurations $\xi^2, \xi^3, \ldots, \xi^k$ of $M_2, M_3, \ldots, M^k$ exists with

$$\xi^i = \varphi_{*i}(\xi^{i+1})$$

for all $i$. If we are also given a sequence of mappings $\Phi_1^*, \Phi_2^*, \ldots$ such that for each $i$, the pair $(\varphi_{i*}, \Phi_i^*)$, is a simulation of $M_{i+1}$ by $M_i$ then we have a *hierarchy of simulations* of height $k$. ⌟

We will construct a hierarchy of simulations whose height grows during the computation—by a mechanism to be described later.

# 4 Simulation structure

In what follows we will describe the program of the reliable Turing machine: a hierarchical simulation in which simultaneously each $M_{k+1}$ is simulated by $M_k$, with an added mechanism to raise the height of the hierarchy when needed. Most of the time, we will write $M = M_k$, $M^* = M_{k+1}$. Ideally, cells will be grouped into colonies of size $Q = B^*/B$. Simulating one step of $M^*$ takes a sequence of steps of $M$ constituting a

*work period*. Machine $M$ will perform the simulation as long as the noise in which it operates is $(\beta(B,T), \gamma(B^*,T^*))$-sparse (as in Definition 2.2). This means, by Lemma 2.3, that a burst affects at most $\beta$ consecutive tape cells, and there is at most one burst in any $\gamma$ neighboring work periods. A design goal for the program is to "correct" a burst within a space much smaller than a colony.

To see whether consistency, that is the basic tape pattern supporting simulation, is broken somewhere, a very local precaution will be taken in each step: each step will check whether the current cell-pair is allowed in a healthy configuration. If not then a *healing* procedure will be called; we will also say that *alarm* will be called. On the other hand, the *rebuilding* procedure will be called on some indications that healing fails.

## 4.1 Error-correcting code

Let us add error-correcting features to the block codes introduced in Definition 3.1.

**Definition 4.1** (Error-correcting code) A block code is $(\beta, t)$-*burst-error-correcting*, if for all $x \in \Sigma_2$, $y \in \Sigma_1^Q$ we have $\psi^*(y) = x$ whenever $y$ differs from $\psi_*(x)$ in at most $t$ intervals of size $\leq \beta$. For such a code, we will say that a word $y \in \Sigma_1^Q$ is $r$-*compliant* if it differs from a codeword of the code by at most $r$ intervals of size $\leq \beta$. ⌟

**Example 4.2** (Repetition code) Suppose that $Q \geq 3\beta$ is divisible by 3, $\Sigma_2 = \Sigma_1^{Q/3}$, $\psi_*(x) = xxx$. If $y = y(1)\ldots y(Q)$, then $x = \psi^*(y)$ is defined by $x(i) = \mathrm{maj}(y(i), y(i+Q/3), y(i+2Q/3))$. For all $\beta \leq Q/3$, this is a $(\beta, 1)$-burst-error-correcting code. Repeating 5 times instead of 3 gives a $(\beta, 2)$-burst-error-correcting code. ⌟

**Example 4.3** (Reed-Solomon code) There are much more efficient such codes than just repetition. One, based on the Reed-Solomon code, is outlined in Example 4.6 of [6]. If each symbol of the code has $l$ bits then the code can be up to $2^l$ symbols long. Only $2t\beta$ of its symbols need to be redundant in order to correct $t$ faults of length $\beta$. ⌟

Consider a (generalized) Turing machine $(\Sigma, \tau)$ simulating some Turing machine $(\Sigma^*, \tau^*)$. We will assume that the alphabet $\Sigma^*$ is a subset of the set of binary strings $\{0,1\}^l$ for some $l < Q$. We will store the coded information in the interior of the colony, since it is more exposed to errors near the boundaries.

**Definition 4.4** Let

*PadLen*

be a parameter to be defined later (in Definition 4.15). A cell belongs to the *interior* of a colony spanning an interval $I$ if it is in $\mathrm{Int}(I, PadLenB)$ (with the interior as in Definition 2.12). The area within $PadLenB$ of a colony end is called the *turn region*. ⌟

Let $(\upsilon_*, \upsilon^*)$ be a $(\beta, 2)$-burst-error-correcting block code with

$$\upsilon_* : \{0,1\}^l \cup \{\emptyset\} \to \{0,1\}^{(Q-2\cdot PadLen)B}.$$

We could use, one of the above example codes, but we require that there are some fixed Turing machines Encode and Decode computing them:

$$\upsilon_*(x) = \mathrm{Encode}(x), \quad \upsilon^*(y) = \mathrm{Decode}(y).$$

In Section 4.6.4 we will be more specific about the choice of code and its space and time requirement.

Recall that our Turing machine has some special states, among others: 0, 1, $\mathrm{new}_0$, $\mathrm{new}_1$. We require that at least some of these, namely $\mathrm{new}_0$ and $\mathrm{new}_1$ have encodings that are especially simple: so $\upsilon_*(\mathrm{new}_0)$ and $\upsilon_*(\mathrm{new}_1)$ can be written down in a single pass of the Turing machine $M$.

Let us now define the block code $(\psi_*, \psi^*)$ used below in the definition of the configuration code $(\varphi_*, \varphi^*)$ outlined in Section 3.5:

$$\psi_*(a) = 0^{PadLen}\upsilon_*(a)0^{PadLen}. \tag{4.1}$$

The decoded value $\psi^*(x)$ is obtained by first removing $PadLen$ symbols from both ends of $x$ to get $x'$, and then computing $\upsilon^*(x')$. It will be easy to compute the configuration code from $\psi_*$, once we know what tracks of the tape need initialization.

## 4.2 Rule language

The generalized Turing machines $M_k$ to be defined differ only in the parameter $k$. We will denote therefore $M_k$ frequently simply by $M$, and $M_{k+1}$, simulated by $M_k$, by $M^*$. Similarly we will denote the colony size $Q_k$ by $Q$.

We will describe the transition function $\tau_k = \tau$ mostly in an informal way, as procedures of a program; these descriptions are readily translatable into a set of *rules*. Each rule consists of some (nested) conditional statements, similar to the ones seen in an ordinary program: "**if** *condition* **then** *instruction* **else** *instruction*", where the condition is testing values of some fields of the observed cell-pair, and the instruction can either be elementary, or itself a conditional statement. The elementary instructions are an *assignment* of a value to a field of a cell symbol, or a command to move the

head. It will then be possible to write one fixed *interpreter* Turing machine that carries out these rules, assuming that the whole set of rules is a string and each field is also represented as a string.

Assignment of value $x$ to a field $y$ of the state or cell symbol will be denoted by $y \leftarrow x$.

Our description of rules is informal, making them sometimes look more like a procedure with many steps. This will just mean that the rule has some of its own dedicated fields to which it can refer and which it can also set: with the help of these, indeed a sequence of actions can take place. One of these fields may indicate which procedure is being performed. Typically, only an element of the current cell-pair would carry the field indicating the procedure being performed, but in some cases it would be a track on a larger area. When a procedure "calls" another one, it will always be clear which one is being performed and which one is just waiting for the called one to finish—there is no recursion. Rules can also have parameters, like $\texttt{MoveFront}(d)$. This parameter can also be seen as just referring to some field. Similarly, when we say that a procedure *returns* a value, it just sets a certain field.

We may refer to two procedures performed one after the other, even when the first one does not move the head, like $d \leftarrow \texttt{ProcessPayload}(j)$ followed by $\texttt{MoveFront}(d)$. The translation of this high-level description into nested if-then-else instructions would combine the two procedures into one.

### 4.3 Fields

A properly formatted configuration of $M$ splits the tape into blocks of $Q$ consecutive cells called *colonies*. One colony of the tape of the simulating machine represents one tape cell of the simulated machine. The two colonies that correspond to the current cell-pair of the simulated machine is scanning are called the *base colony-pair*. A colony-pair can also be formally defined, for the program, based on some field values in cells. Sometimes the left base colony will just be called the *base colony*. Most of the computation proceeds over the base colony-pair. The direction of the simulated head movement, once figured out by the computation, is called the *drift*. The neighbor colonies of the base colony-pair may not be adjacent, in which case there will be a *bridge* between them formed by neighboring (not necessarily adjacent) tape cells. The possible space between neighbor colonies other than the base colony-pair will be filled by stem cells (see below).

Let us describe some of the most important fields we will use in the tape cells; others will be introduced later.

*Procedures* Some fields will just indicate which procedure is currently active: for ex-

ample

`Boot`, `Simulate`, `Heal`, `Rebuild`, `RebuildHeal`.

The basic simulation activity is called the procedure `Simulate`: when it is active, we may also say that the computation is in *normal mode*. The *booting* procedure is used on the highest level of the simulation hierarchy; this level will be raised if the length of computation time of the simulated machine $G$ makes it necessary. When this procedure is active, we may also say that the computation is in *booting mode*. The *healing* procedure tries to correct some local fault in simulation due to a couple of neighboring bursts, while the *rebuilding* procedure attempts to restore the colony structure on the scale of a couple of colonies. When it is active, we may say that the computation is in *rebuilding mode*. The rebuilding procedure may also need some healing; this is handled by `RebuildHeal`.

*Info* The *Info* track of a colony of $M$ contains the string that encodes the content of the simulated cell of $M^*$.

*Payload* The *Payload* field of each cell contains a tape segment of the simulated machine $G$. See Remark 4.6 below on duplication.

*Address* The field *Addr* of the cell shows the position of the cell in its colony: it takes values in $[0, Q)$.

*Drift* The direction in $\{-1, 1\}$ in which the simulated head moves will be recorded on the track *Drift*.

*Age, Sweep* The *Age* field keeps track of the step number of the computation within the work period of a colony pair. The work period will consist of some consecutive *stages*, whose beginning is marked by a certain value of *Age*. When a stage is finished (as seen from other indicators), the *Age* may jump to the starting value of the next stage: the number of actual steps in a stage may not always be the same, but upper bounds are established.

In some parts of the program (like the transfer phase), new cells may be inserted, causing the *Age* field to experience a—harmless—jump. On the other hand, in these parts, the front will make sweeps over the whole colony pair; So instead of the *Age* a field called *Sweep* will be used that just counts which sweep is being performed.

*Kind* Cells will be designated as belonging to a number of possible *kinds*, signaled by the field *Kind* with values New, Booting, Stem, $\text{Member}_0$, $\text{Member}_1$, Bridge, Outer. Here is a description of their role.

- The kind New has been discussed before.
- A cell is of the Booting kind if it is on the top level of simulation (see Section 2.2).

- Cells of the base colony-pair are of type $\text{Member}_0$ and $\text{Member}_1$ respectively. Members of other colonies have the kind Outer.
- If the two base colonies are close but not adjacent then there will be $< Q$ adjacent cells of type Bridge between them, extending the left base colony towards the right one.
- Stem is the kind of cells filling the space in between colonies other than the two base colonies.

*Heal, Rebuild* During healing and rebuilding, some special fields of the state and cell are used, treated as subfields of the field *Heal* or *Rebuild*. In particular, instead of numbering their individual steps by an *Age* field, these procedures make left-right and right-left *sweeps* over their interval of operation. There will be a *Heal*.*Sweep* field (or, respectively, *Rebuild*.*Sweep*) showing the number of the current sweep. A cell will be said to be *marked for rebuilding* if any part of the *Rebuild* track is defined.

**Remark 4.5** The Outer kind is redundant: whether a cell is outer can be computed from its *Drift* and *Age* fields. But we use it for clarity. ⌟

**Remark 4.6** (Duplication) The *Info* track of a colony encodes the content $s^*$ of the simulated cell of $M^*$. In particular, the $s^*$.*Payload* field contains the tape segment of the simulated machine $G$ represented by the simulated cell. The *Payload* track of the colony will represent the same tape segment (cut up into pieces in its individual cells). This duplication will not create too much space redundancy—due to the small number of levels relative to the size of the computation. (It could also be avoided, since at any one time the *Payload* track only needs to represent some small part of $s^*$.*Payload*, the part currently worked on, therefore its "bandwidth" could be kept small. In [6] for cellular automata this approach led to a constant factor space redundancy.)

⌟

## 4.4 Head movement

The global structure of a work period is this:

*Simulation phase* Compute the new state of the simulated cell-pair, and the simulated direction (called the drift). Then check the "meaningfulness" of the result.

*Transfer phase* The head moves into the neighbor colony-pair in the simulated head direction called drift (creating and destroying bridges if needed). In this phase, the number the current sweep is shown on the *Sweep* track.

As the head leaves behind a cell, this remembers the last *Age* and *Sweep* value. In all important cases as we will see, the simulation will recognize from the neighborhood of the head if some age jump happened by error.

**Definition 4.7** (Front) The position towards which the *Age* values of the cells increase both from left and right will be called the *front*. The direction of the front is towards the smaller step values. ⌟

Globally in a configuration, due to earlier faults, there may be more than one front, but locally we can talk about "the" front without fear of confusion.

Bridges between colonies present some extra complication—let us address it.

**Definition 4.8** (Gaps) If the bodies of two cells are not adjacent, but are at a distance $< B$ then the space between them is called a *small gap*. We also call a small gap such a space between the bodies of two colonies. On the other hand, if the distance of the bodies of two colonies is $> B$ but $< QB$ then the space between them is called a *large gap*. ⌟

A large gap between two colonies will be filled by a bridge when they become a base colony-pair. A bridge is always extending the left member colony, except possibly during transfer while the colony pair is moved. Building a bridge or making repairs may involve "killing" some cells that are in the way and replacing them with new ones, via the replacement action of Definition 2.11.

Due to the zigging and feathering requirements mentioned in Section 2.4, moving the front will actually be a complex procedure itself, described below at (4.4). This procedure combines zigging and feathering as described below, and uses the parameters

$$Z = \pi^{2+\rho},\ F = Z\pi^{2+\rho} \tag{4.2}$$

with $0 < \rho < 1/4$. The choices will be motivated in Sections 4.4.2 and 7.3.

### 4.4.1 Zigging

A *zigzag* movement will check the consistency of a few cells around the front. The process creates a *frontier zone* of about $Z/2$ cells around the front, where $Z$ was defined in (4.2). In normal mode, this interval is recognizable just from the *Age* track, but during rebuilding will be marked on a special track (see Section 5.4). In every second step of moving in any direction (say at every even value of *Age*), the head will perform a forward-backward-forward zigzag: going $Z$ steps ahead of the center of the frontier zone, then $Z$ steps behind it.

The turns while doing this are *small turns* defined in Section 4.4.2, meaning that a few more steps (normally at most 2) may be needed to find a turning point.

The step counting for zigging can be done locally, in the current cell-pair.

**Remark 4.9** The size of the parameter $Z$ in (4.2) and with it the size of the frontier zone is motivated, among others, in the proof of Lemma 7.15 where it has to withstand a large number of bursts. The forward-zigging property allows, among other uses, to recognize a pair of opposing frontier zones. ⌟

### 4.4.2 Feathering

Example 11.1 suggests that our Turing machine should have the property that between two turns on the same point $x$, it should pass $x$ at least once. We will call this property *feathering*, referring to the picture of the path of the head in a space-time diagram, as in Example 4.11. In fact in some cases we will require more:

**Definition 4.10** A Turing machine has the *c-feathering* property for a $c > 0$, if after a right-to-left turn on point $x$, the next right-to-left turn at a point $\geq x$ must be at a point $\geq x + cB$, and similarly for left-to-right turns. ⌟

The following example suggests that any computation can be reorganized to accomodate feathering, at the price of at most a logarithmic delay.

**Example 4.11** Suppose that, repeatedly, arriving from the left at position 1, the head decides to turn left. It can then make its turns at the following sequence of positions:

$$1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1, 5, 1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, \ldots$$

⌟

**Definition 4.12** (Digression) Whenever a turn is postponed since the head is not allowed to turn due to feathering, the simulation to be carried out by the head is suspended until the head returns. This is called a *digression*. ⌟

If in the original execution the head turned back $t$ consecutive times to the left from position $p$, then now it will turn back from somewhere in a zone of size $O(\log t)$ to the right of $p$ in each of these times. Computing the exact turning point is not necessary, but the following lemma will be useful.

**Lemma 4.13** *Suppose that the Turing machine has the feathering property. For each $n \geq 0$, if during some time interval the head passes $2^n$ times to the right from point $x$, then during the same interval, it must reach $x + nB$. The analogous statement holds for passing to the left.*

The proof is easy by induction.

**Remark 4.14** The amortized delay caused by $c$-feathering is, in fact, only by a factor of $c$ times the number of turns. Indeed, given a Turing machine $T$ we can build a

Turing machine $T'$ with the $c$-feathering property simulating $T$ as follows. Machine $T'$ is almost like machine $T$, except that it keeps a marker at the place of the simulated head of $T$, and a track *Pass* with values $0, \pm 1$ marking the place of each past turn that has not yet been passed over. It is set to 0 in each cell that the head passes over, 1 when the head turns left (since next time it must be passed to right), and $-1$ if it turns right. (The sign is only for exposition purposes, as it always follows from the context.) Suppose that the head of $T$ must make a right-to-left turn. If $Pass = 0$ then $T'$ just turns (moving the marker with it). Otherwise the head leaves the simulated head in place, and moves right by $c$ cells. Every time it encounters another cell with $Pass = 1$ it resets the counter, moving $c$ cells again. If it encounters no such cell then it turns. As it returns to the simulated head, it continues the simulation.

We claim that when simulating a machine making $n$ turns, the total delay due to these digressions is at most $2cn$. Indeed, every rightward step during a right-to-left digression of $T'$ is within distance $c$ of a past right-to-left turn, which on the other hand can be attributed to a right-to-left turn of the original computation of $T$. ⌟

Our simulating Turing machine will have two different feathering properties: it will obey 1-feathering for all its turns, but on certain kinds of turn called *big turn* will obey $F$-feathering for the parameter $F$ defined in (4.2). The $F$-feathering property will sometimes force the head during the work period to go beyond the boundaries of the colony-pair, but only to a limited extent:

**Definition 4.15** Let

$$PadLen = 4F \log Q. \tag{4.3}$$

For an interval $I$ spanned by a colony pair we call the the *turn region* the set $\mathrm{Int}(I, -PadLen) \setminus C$, that is the close outside neighborhood of $I$. ⌟

*Small turn* Whenever the head needs to turn (for example during zigging), the event will be called a *small turn*.

*Pass*,

whose default value is 0, will be set to $\pm 1$. Consider right-to-left turns, the left-to-right turns are analogous. The head then arrives at a cell-pair $(x, y)$ from the left. If $y$ has $Pass = 1$ then the head is not allowed to turn left: it continues right. If $y$ has $Pass = 0$ and the head turns left then $y$ gets $Pass \leftarrow 1$. In both cases, $x$ gets $Pass \leftarrow 0$. The event when a cell with $Pass = 0$ is not found within $3\Delta$ steps (with $\Delta$ defined in (5.2) below) will be called *small turn starvation*. Then start or restart healing, (see below) but still don't turn.

Suppose that the current cell-pair is $(x, y)$, and the next step replaces $y$ by some $y'$ as in Definition 2.11. Then $y'$ inherits the *Pass* value of $y$.

The set New of states will only have two elements, $\text{new}_0$ or $\text{new}_1$, where $\text{new}_i$ is a state with field $Pass = i$. When a replacement operation takes place as in Definition 7.13, then the *Pass* field of the cell being replaced will be inherited by the cell replacing it.

**Remark 4.16** As a consequence of the above rule, a cell will *never be killed* just when the head turned back from it. ⌟

In normal and rebuilding mode, a zigging move is done only after every two steps of moving the front: this leaves every second cell with $Pass = 0$ in the wake of this movement.

*Big turn* The turns of the front will be called *big turns*, carried out by the procedure

$$\texttt{MoveFront}(d), d \in \{-1, 0, 1\}. \tag{4.4}$$

The possibility of $d = 0$ will be used in Section 4.6.1; let it simply mean calling first $\texttt{MoveFront}(1)$ and then $\texttt{MoveFront}(-1)$. A big left turn will be performed by calling $\texttt{MoveFront}(-1)$. The procedure is governed with the help of the field:

$$BigDigression \in \{-1, 0, 1, \ldots, F\} \cup \{*, \omega, \delta_{-1}, \delta_1\}$$

where $*$ means "undefined", this is the default value. We will have $Z/2$ consecutive cells traveling with the head, storing one and the same value of *BigDigression* (except while it is updated): we will call this the *digression marking zone*, or *D-zone*.

Let us describe the actions performed on account of a big right-to-left turn of the front in normal mode; left-to-right turns are similar. The somewhat simpler case of big turns of the rebuilding procedure will be described In Section 5.4. So assume that the front has been moving right, calling $\texttt{MoveFront}(1)$ repeatedly. It carries the D-zone, and increases *BigDigression* in it by 1 at every step until $BigDigression = F$ or until it encounters $Z/2$ consecutive cells marked $BigDigression = \omega$, called a the *footprint* of an earlier right-to-left turn. In this case, it resets $BigDigression \leftarrow 0$ in the D-zone as it continues right (erasing the old footprint in the process).

When $\texttt{MoveFront}(-1)$ is called, the right movement still continues (shifting the D-zone, and making zigs of size $Z$ ahead and behind its center) until $BigDigression = F$. The area between the front and the D-zone is filled with $\delta_1$ (it would be $\delta_{-1}$ in case of a digression towards the left). The old footprint is erased as it is passed over. Once a big left turn is allowed since $BigDigression = F$, we set $BigDigression \leftarrow \omega$ in the D-zone, leaving a new footprint. The head moves back to the front, carrying

*BigDigression* $= -1$ in the D-zone. Once the D-zone is at the front, the front is moved one step left, and the D-zone is moving with it, setting *BigDigression* $\leftarrow 0$.

In normal mode, when the search for a place of big turn takes longer than $3F \log Q$ steps this will be called a *big turn starvation*. To recognize this, the D-zone can also maintain a field *FrontAddr* to keep track of the address difference between it and the front. This will be recognizable from the distance of the D-zone from the front as seen from the address and age tracks, and then rebuilding will be called.

**Remark 4.17** The structure of the program will be such that `MoveFront`$(d)$ or `ProcessPayload`$(j)$ (see later) will only be called when a previous execution of `MoveFront`$(\cdot)$ finished. ⌟

Machine $M$ will have the property that after a fault-free path passed over a clean interval, both small turns and big turns can happen without too long digressions. We give here only an informal argument; formal proof must wait until a (more) complete definition of the simulation. Zigs are by the definition spaced by $\geq 2$ cells apart, making sure that the points with *Pass* $\neq 0$ are in general at a distance of $\geq 2$ apart. Healing can create only a constant number of segments of *Pass* $\neq 0$ of size $\leq 3\Delta$ with $\Delta$ defined in (5.2). As for big turns, Example 4.11 (for $F = 1$) shows that a big turn attempt will be delayed by at most $F$ times the logarithm of the total number of big turns inside a colony or a rebuild area.

The simulated Turing machine will also have the feathering property, therefore the simulation will not turn back from one and the same colony repeatedly, without having passed it in the meantime.

**Remark 4.18** The size of the parameter $F$ is motivated by the proof of Lemma 7.18. Here is a sketch of the argument (it can be safely skipped now). At some time $t_0$ in some interval $I$ we will have clean subintervals $J_k(t_0)$, $k = 1, 2, \ldots$ of size $\geq 6ZB$ in which no fault will appear, and which are separated from each other by areas of size $O(\pi^2 ZB) \ll FB$. For times $t > t_0$ we will track the maximal clean intervals $J_k(t)$ containing the middle of $J_k(t_0)$.

Assume that the head passes over $I$ noiselessly left to right and later also noiselessly from right to left. If the head moves in a zigging way to the right then the Attack Cleaning property will clean out the area between $J_i(t)$ and $J_{i+1}(t)$, joining them. This does not happen only in case of a big turn from the right end of $J_i(t)$. But then in the next pair of passes over $I$, the $F$-feathering property implies that the big turn from the end of $J_i(t)$ is at least a distance $FB$ to the right. Our choice of $F$ implies that then $J_i(t)$ will be joined to $J_{i+1}(t)$. So two noise-free passes would join all the intervals $J_i(t)$ into a clean area. ⌟

## 4.5 Simulation phase

As mentioned in Section 4.3 the work period will be divided into *stages* starting at specific values of the *Age* field. These stages are grouped into two big phases: the *simulation* phase and the *transfer* phase. The simulation phase, governed by the `Compute` rule, computes new values for the current cell-pair of the simulated machine $M^*$ represented by the current (base) colony-pair, and the move direction of the head of $M^*$. The cell state of $M^*$ will be stored on the track *Info* of the representing colony. The move direction of $M^*$ will be written into the *Drift* field of *each* cell of the base colony-pair (filling the whole track with the same symbol $d \in \{-1, 1\}$).

Let

$$\beta' = \beta + 2c_{\text{spill}}, \tag{4.5}$$

$$c_{\text{stain}} = 2\beta' + 1. \tag{4.6}$$

The rule `Compute` relies on some fixed $(c_{\text{stain}}, 2)$ burst-error-correcting code, moreover it expects each of the words found on the *Info* track to be 2-compliant (Definition 4.1).

The rule starts with checking that the input colonies are compliant using a rule `ComplianceCheck`. Then essentially repeats for $j = 1, 2, 3$ the following *stages*: decoding, applying the transition, encoding. It uses some additional tracks like *Work* for most of the computation, and outputs its result onto the $Hold[j]$ track. The *Info* track will not be modified before all the $Hold[j]$ tracks are written.

In more detail:

1. At the start, the current cell-pair is the left pair of cells of the left member of the base colony-pair.

   Everywhere outside the base colony-pair, the *Drift* values (as well as the increasing *Age* values) are pointing towards it.

2. During the execution, the big turns may occur outside the colony-pair, in the turn region as in Definition 4.15. If there is no neighboring colony then outer adjacent stem cells will be used, or added as needed.

3. For $j = 1, 2, 3$, call `ComplianceCheck` on the *Info* track of both colonies of the pair, and write the resulting bit into the $Compliant_j$ track of each.

   Then pass through each colony of the pair and for each address $i$, if in the cell with this address, the majority of $Compliant_j$, $j = 1, 2, 3$ is false, then turn this cell into the $i$th cell of the colony representing the state $\text{new}_0$. Recall that in Section 4.1, we required that the codes of the states $\text{new}_i$, $i = 0, 1$ are simple enough so that they can be written in a single pass.

   (We could have used any other state instead of $\text{new}_0$ here: just some simple default state is needed.)

4. For $j = 1, 2, 3$ do the following, using some work tracks:

   a. Calling by $\mathbf{a}$ the pair of strings found on the *Info* track of the interiors $\mathrm{Int}(C, PadLen)$ of the base colonies $C$, decode it into the pair of strings

   $$(\tilde{\mathbf{a}}_0, \tilde{\mathbf{a}}_1) = \tilde{\mathbf{a}} = \upsilon^*(\mathbf{a}) \tag{4.7}$$

   (the current state of the simulated cell-pair), and store it on some auxiliary track in the base colony-pair. Do this by computing $\tilde{\mathbf{a}} = \mathrm{Decode}(\mathbf{a})$ on some simulated Turing machine. (The time complexity of this procedure will be discussed in Section 8.2.)

   b. Compute $(\mathbf{a}', d) = \tau^*(\tilde{\mathbf{a}}, \alpha)$, where $\alpha$ = True if the pair of colonies is adjacent, else False. This step needs elaboration for two reasons. First, part of this transition is the processing of track *Info*.*Payload*, which represents the tape of the Turing machine $G$ simulated by the machine $M^*$. Second, the program of the transition function $\tau^*$ is not written explicitly anywhere (this is a "self-simulation" situation).

   Both issues will are discussed in detail in Section 4.6.

   c. Write the encoded new cell states $\upsilon_*(\mathbf{a}')$ onto the $Hold[j]$.*Info* track of the interior of the base colony-pair. Write $d$ into the $Hold[j]$.*Drift* field of *each cell* of the left base colony.

   A field called *Replace* is used. Its value can be $*$ (undefined), or an element of the set New. If one of the new states of the simulated cell pair belongs to New (that is, the rules dictate a replacement, as in Definition 2.11), then write it onto the $Hold[j]$.*Replace* track everywhere; else the values on the track will be undefined. There is enough capacity in a cell to record this value of a simulated cell (which can have many more states), since the set New has only two possible elements in $M^*$ as well as $M$.

5. Sweeping through the base colony-pair, at each cell compute the majority of $Hold[j]$.*Info*, $j = 1, \ldots, 3$, and write it into the field *Info*. Proceed similarly, and simultaneously, with *Drift* and *Replace*.

6. If the *Output* field of the simulated cell is defined, write it into the output field of the left end-cell of each colony.

Part 6 achieves that when the computation finishes on some simulated machine $M_k$, its output value in cell 0 of $M_k$ will "trickle down" to the output field of cell 0 of $M_1$, as needed in Theorem 1.

The transfer phase (see Section 4.7) will use the information in the *Replace* track to carry out, in simulation, the replacement action of Definition 2.11.

## 4.6 TM simulation and self-simulation

Let us elaborate stage 4b of Section 4.5.

### 4.6.1 Handling the payload

The tape of machine $M_k$ contains a track called *Payload* that represents the tape of the target simulated Turing machine $G$ of Theorem 1. In a simulation work period, the *Payload* track on level $k$ will come into play only if the head of $G$ is simulated on level $k+1$. This simulation will be performed only when the procedure

$$\texttt{ProcessPayload}(j)$$

is called. Here, $j \in \{0,1\}$ shows whether the represented head of $G$ is inside the left or the right colony of the current colony-pair. The procedure returns a number $d \in \{-1,0,1\}$, to be used as an argument to $\texttt{MoveFront}(d)$. Here are the details.

Consider the simulation of machine $M^*$ on machine $M$. The track *Payload* (after decoding) consists of *Payload*.*Tape* and *Payload*.*Pos*. The track *Payload*.*Tape* contains the tape segment of $G$ represented by the colony. Track *Payload*.*Pos* is used only from the left neighbor colony in case of $\texttt{ProcessPayload}(0)$, else from the right one. For definiteness, assume it is in the left one. Its value has the form $Pos = (a, j')$ with $j' \in \{0,1\}$. Here, $a$ is the address of the cell of $M$ in the colony containing the represented head of $G$. The rules ensure that when $\texttt{ProcessPayload}(0)$ is called then we will never have $a = 0$, that is $a$ will never be at the very left end of the left neighbor colony. This way, the current simulated cell-pair is always inside the current colony-pair.

If the level $k$ is larger than 1 then each cell of $M$ itself represents a tape segment of $G$; in this case the value $j'$ shows whether left cell or the right cell of the cell-pair of $M$ at address $a$ contains the represented head of $G$.

The rule works on the decoded states $(\tilde{\mathbf{a}}_0, \tilde{\mathbf{a}}_1)$ of the current cell-pair of $M^*$, as in (4.7). It copies $\tilde{\mathbf{a}}_0$.*Payload*.*Tape* and $\tilde{\mathbf{a}}_1$.*Payload*.*Tape* as consecutive strings onto the *Payload*.*Tape* track of the current colony-pair. Then (assuming $j = 0$) it recovers $(a, j') = \tilde{\mathbf{a}}_0$.*Payload*.*Pos*, goes to address $a$ of the left neighbor colony, and repeats the following action (see below for how many times).

(G) If $k = 1$ then the cells $a-1, a$ of the *Payload*.*Tape* track actually represent the content $\mathbf{a}$ of two neighbor cells, so apply the transition function computing $(\mathbf{a}', d') \leftarrow \tau_G(\mathbf{a})$, and set $\mathbf{a} \leftarrow \mathbf{a}'$.

If $k > 1$ then call $d' \leftarrow \texttt{ProcessPayload}(j')$ on the cell-pair at address $a$ (recursively). In both cases, follow this by $\texttt{MoveFront}(d')$.

How many times to perform action (G)? At most $Q$ times, but stop earlier if the head would reach the left end of the left colony or the right end of the right one.

Return $d = -1$ if the head arrives in the left half of the left colony, $d = 1$ if it is in the right half of the right colony, and $d = 0$ otherwise.

### 4.6.2 New primitives

In what follows describe the tools of self-simulation. The simulation phase makes use of the track *Work* mentioned above, and the track

$$Index$$

that can store a certain address of a colony. Recall from Section 4.2 that the program of our machine is a list of nested "**if** *condition* **then** *instruction* **else** *instruction*" statements. As such, it can be represented as a binary string

$$R.$$

If one writes out all details of the construction of the present paper, this string $R$ becomes explicit, an absolute constant. But in the reasoning below, we treat it as a parameter. Let us provide a couple of *extra primitives* to the rules. First, they have access to the parameter $k$ of machine $M = M_k$, to define the transition function

$$\tau_{R,k}(\mathbf{a}).$$

The other, more important, new primitive is a special instruction

`WriteProgramBit`

in the rules. When called, this instruction makes the assignment $Work \leftarrow R(Index)$. This is the key to self-simulation: *the program has access to its own bits*. If $Index = i$ then it writes $R(i)$ onto the current position of the *Work* track.

### 4.6.3 Simulating the rules

The structure of all rules is simple enough that they can be read and interpreted by a Turing machine in reasonable time:

**Theorem 2** *There is a Turing machine* Interpr *with the property that for all positive integers $k$, string $R$ that is a sequence of rules, and a pair of bit strings $\mathbf{a} = (a_0, a_1)$ with $a_j \in \Sigma_k$,*

$$\text{Interpr}(R, 0^k, \mathbf{a}) = \tau_{R,k}(\mathbf{a}).$$

The proof parses and implements the rules in the string $R$; each of these rules checks and writes a constant number of fields. Implementing the `WriteProgramBit` instruction is straightforward: Machine Interpr determines the number $i$ represented by the simulated *Index* field, looks up $R(i)$ in $R$, and writes it into the simulated *Work* field. There is no circularity in these definitions:

- The instruction `WriteProgramBit` is written *literally* in $R$ in the appropriate place, as "`WriteProgramBit`". The string $R$ is *not part* of the rules (that is of itself).
- On the other hand, the computation in Interpr$(R, 0^k, \mathbf{a})$ has *explicit* access to the string $R$ as one of the inputs.

Let us show the computation step invoking the "self-simulation" in detail. In the earlier outline, step 4b of Section 4.5 said to compute $\tau^*(\tilde{\mathbf{a}})$ (for the present discussion, we will just consider computing $\tau^*(\mathbf{a}) = \tau_{k+1}(\mathbf{a})$), where $\tau = \tau_k$, and it is assumed that $\mathbf{a}$ is available on an appropriate auxiliary track. We give more detail now of how to implement this step:

1. Onto the *Work* track, write the string $R$. To do this, for *Index* running from 1 to $|R|$, execute the instruction `WriteProgramBit` and move right. Now, on the *Work* track, add $0^{k+1}$ and $\mathbf{a}$. String $0^{k+1}$ can be written since the parameter $k$ is available. String $\mathbf{a}$ is available on the track where it is stored.
2. Simulate the machine Interpr on track *Work*, computing $\tau_{R,k+1}(\mathbf{a})$.

This implements the forced self-simulation. Note what we achieved:

- On level 1, the transition function $\tau_{R,1}(\mathbf{a})$ is defined completely when the rule string $R$ is given. It has the forced simulation property by definition, and string $R$ is *"hard-wired"* into it in the following way. If $(\mathbf{a}', d) = \tau_{R,1}(\mathbf{a})$, then

$$a_0'.\mathit{Work} \leftarrow R(a_0.\mathit{Index})$$

  whenever $a_0.\mathit{Index}$ represents a number between 1 and $|R|$, and the values $a_0.\mathit{Sweep}$, $a_0.\mathit{Addr}$ satisfy the conditions under which the instruction `WriteProgramBit` is called in the rules (written in $R$).
- The forced simulation property of the *simulated* transition function $\tau_{R,k+1}(\cdot)$ is achieved by the above defined computation step—which relies on the forced simulation property of $\tau_{R,k}(\cdot)$.

**Remark 4.19** This construction resembles the proof of Kleene's fixed-point theorem, and even more some self-reproducing programs (like a program $p$ in the language C causing the computer to write out the string $p$). ⌟

### 4.6.4 The length of a work period

We claim that the number of steps in the work period is $O(|R|QFZ^2)$, where $R$ is the program string, so $|R|$ is actually just a constant. Below we will first ignore the extra burden of zigging and feathering responsible for the factor $FZ$: without this, we get a bound $O(|R|QZ)$.

*Coding-decoding* The computing part of the work period performs three repetitions of coding, the actual simulation work and decoding. We need a code that corrects at least 3 bursts of $\beta$ cells each. One way to do this is via a Reed-Solomon code. There are encoding and decoding procedures for an $n$-symbol Reed-Solomon code that take time $O(n^2)$. Since this is not linear, we can subdivide the $Q$ cells of a colony into segments of, say, $Z$ cell each and encode-decode each of these separately. This way the total number of steps spent on these procedures is $O(QZ)$.

*Payload* The procedure `ProcessPayload`$(j)$ can take up to $Q$ simulation steps. (It may make fewer steps if the simulated head reaches the edge of the colony-pair, but then the next work period will have at least $Q$ simulation steps.)

*The rest* The rest of the simulation takes only $O(|R|Q)$ steps. Indeed, the number of fields is $O(|R|)$, and notice that each field other than the one having to do with *Payload* is a number whose size is $O(Q^2)$. We are now processing a colony representing a cell of $M^* = M_{k+1}$, so the numbers may have size $(Q^*)^2 = Q_{k+1}^2 = 2^{2\cdot 1.5^{k+1}}$. A cell of $M$ has size $\prod_{i<k} Q_i = 2^{1+1.5+\cdots+1.5^{k-1}}$, more than enough to store such a number. The program involves comparing fields represented in cells found in the left and the right colony of the colony pair, so it may need to copy the fields found in the right colony to the left one, necessitating $O(|R|Q)$ steps; after this the comparisons can be done locally before the results are carried back.

## 4.7 Transfer phase

Before the transfer phase, members of the base colony-pair $C_0, C_1$ have cells of kind $\text{Member}_0$ and $\text{Member}_1$ correspondingly, with a possible bridge between them. In the transfer phase, control will be transferred to the neighbor colony-pair in the direction of the simulated head movement which we called the *drift*, found on the *Drift* track. Whenever the *Replace* track holds a defined value, we will say that this is a *replacement* situation.

During the transfer phase, the range of the head includes the base colony-pair and a neighbor colony called *target colony* in the direction of the drift. At the beginning of the phase, the current cell-pair is the first cell-pair of $C_0$. Big turns happen in the turn region as in part 2 of the description of the `Compute` rule.

In this phase, the front will move in sweeps, and the track *Sweep* will be rewritten to the number of the sweep being currently performed. The first sweep will bring the head to the end of the turn region beyond the new neighbor colony. Subsequent turns will therefore all happen closer.

Consider $Drift = 1$.

1. Suppose that we don't have a replacement situation. In the first sweep, the head will travel right. Turn all elements of $C_0$ into outer cells, and turn the elements of $C_1$ into $\text{Member}_0$ cells. Then continue to the right, start a bridge (if necessary) towards the right (killing all possible non-adjacent stem cells in the way). If the right end of the bridge reaches an outer colony $C_2$ before $Q$ bridge cells are created, then pass to the right edge of $C_2$. If $Q$ bridge cells were created, stop at the right edge of this bridge—call $C_2$ the bridge just created. Then sweep back to the left end of $C_1$, while turning the cells of $C_2$ to kind $\text{Member}_1$.

   In both cases, actually go to a distance $3F \log Q$ from the right end of $C_2$ before attempting to turn. (This way, all later attempted left turns in the next work period will happen to the left of this one, and so any possible cause for alarm is encountered already now, in the transfer process.)

2. In the replacement situation, build a new colony $C_1'$ adjacent on the right to $C_0$. In the first sweep, perpetuate the value $r$ found on the *Replace* track. Write onto the *Info* track of $C_1'$ the encoding of the value $r$. (This requires two steps for each created cell $x_i$ of $C_0'$: first it has value $r \in \text{New}$, then it gets address $i$, and its *Info* field gets the $i$th symbol of the encoding of $r$.) Then continue to the end of $C_1$ (which should be beyond the end of $C_0'$). On the way back, replace the remaining elements of $C_1$ with stem cells and set the kind of elements of $C_1'$ to $\text{Member}_1$.

A similar program is executed when $Drift = -1$. The values of *Drift*, *Replace*, *Sweep* and *Addr* always determine what step to perform.

Fault-checking during zigging will notice when a burst compromises this process (for example when the end of a bridge would "bite" into another colony), by checking whether all boundaries it finds are legal (see the definition of health in Section 5.1) and trigger healing (see later).

## 4.8 Booting

Ideally, the work of machine $M$ starts from a single active cell-pair of the Booting kind, with addresses $Q - 1$ and 0, the middle cell-pair of a yet to be built colony-pair. The *Payload* track of the cell-pair holds a tape segment of the simulated Turing machine $G$, along with the simulated head. Such a cell-pair will be called a *booting*

*pair*. The segment consisting of this cell-pair will be extended left-right by booting cells, eventually creating a colony-pair, as follows.

*Main work* Process the payload just as in Section 4.6.1, for at most $Q$ processing steps (using `ProcessPayload()` and `MoveFront()`). All new cells encountered must be stem cells (blanks), and all the ones in the segment already created must be of the Booting kind; otherwise call alarm. In all this, use zigging and feathering.

*Lifting* Create the colony-pair around the original pair of booting cells (and turn its cells into member cells). *Lift* (copy) the *Payload* track of its cells into the *Payload* track of the cell-pair simulated by it. Start the booting procedure on the simulated cell-pair.

No decoding-encoding and repetition mechanism is used to correct computational faults during the booting phase, since we do not expect faults during it—see the probability analysis in Section 8.1. (Of course faults could introduce booting cells during other parts of the computation; this will be caught by the zigging mechanism.)

# 5 Healing and rebuilding

Here we define the part of the simulation program that repairs local inconsistencies.

## 5.1 Health

Structure is maintained with the help of a small number of fields. The required relations among them allow the identification of local inconsistency, and its correction provided it was caused locally.

**Definition 5.1** The tuple *Core* of fields is the tuple

$$Core = (Kind, Drift, Replace, Addr, Age, Sweep, Rebuild, BigDigression, FrontAddr).$$

An interval of non-stem adjacent neighbor cells is a *homogenous domain* if its core variables with the exception of *Addr* and *Age* have the same value. *Addr* increases one step at a time left to right. If we are in the simulation phase then *Age* increases one step at time, either left to right or right to left. We don't require this in the transfer phase: there, the *Sweep* track serves for health check instead. The *left end* of a domain is the left edge of its first cell, and its *right end* is the right edge of its last cell. A *left boundary* is the left end of a homogenous domain with either no left neighbor cell or with a neighbor cell belonging to another homogenous domain. Right boundaries are defined similarly. ⌟

Health can be defined formally on the basis of the informal descriptions given here, but the details would be tedious. Recall Definition 4.7 of the front.

H1) A configuration consists of intervals of non-stem neighbor cells, with possibly stem cells between them. The health for each of these intervals is defined locally. No cell is marked for rebuilding, that is *Rebuild* is undefined.

As a non-local condition we will require that exactly one of these intervals contains the front: let us call this the *principal interval*, and that the drift in all other intervals is directed towards the principal one.

H2) In the principal interval there is a base colony-pair, with possibly a bridge going from one member of the pair to the other one. Let $I$ denote the interval containing this pair.

H3) During the `Compute` phase, outside the base colony-pair (both in the principal interval and elsewhere), all non-stem cells have their *Drift* value directed towards the base colony-pair. There are possibly colonies of type Outer adjacent to it and each other. They are called *left outer* colonies and their cells left outer cells, or *right outer* colonies depending on whether their drift is $+1$ or $-1$. Stem cells are also called outer cells (both left and right).

In part 5 of this phase, the values of *Drift* and *Replace* can change at the front.

H4) During transfer, the base colony (of the pair) that is in the direction of the drift is possibly extended by a bridge towards the target colony. At this point there is no other bridge, and the front is at the tip of the new bridge.

In the later part of this sweep, the new bridge already reaches the target colony. If the bridge extends to a full colony then this is converted to the appropriate kinds of member cells in the backward sweep. Domains ahead and behind the front show the changes done. Another possible change occurring at the front is replacement, when dictated by the *Replace* track. In this case the front has the property that, for example when replacing a colony in the right direction, for each member cell created to replace an old member cell, *the address of the new cell is not larger than the address of the one it replaces*.

H5) There is a D-zone (see Section 4.4.2) of length $Z/2 \pm 1$ cells, either at or ahead of the front, and the head is in or adjacent to it. There are possibly footprints of big turns. The length of such a footprint is $Z/2$ when the head is not in it, possibly less as it is being created or erased.

**Definition 5.2** Let us call a bondary *legal* if it can occur in a healthy configuration. We say that a configuration is *pre-healthy* if it is healthy except possibly condition in (H5) above on the length of the D-zone and the length of the footprints of big turns, and some rebuilding marks. ⌟

The following lemma shows that health of an interval of non-stem neighbor cells is locally checkable.

**Lemma 5.3** *If an interval of neighbor cells in a tape configuration has only legal boundaries (including those at its ends) then it is pre-healthy.*

*Proof.* In what follows we don't repeat it but each statement is forced by the kind of boundaries allowed. There are several kinds: a colony boundary, the front, the boundaries of any footprint of a big turn, those of the big digression zone, and within that zone, the place where *BigDigression* changes by 1. What matters is the values of the *Core* variables in the cell pair around the boundary.

1. Consider an interval $I$ of neighbor cells. If $I$ contains cells that are not outer, or it contains both left and right outer cells then it also contains the front.
2. Assume that $I$ contains only left outer cells. Then these consist of colonies possibly separated by stem cells, with drift pointing to the right, and possibly a front in a right turn region. The situation is similar when $I$ consists of right outer cells.

   In all other cases $I$ also contains an interval $K$ consisting of neighbor non-outer cells.
3. Let $s$ be the maximum age found in $K$. If $s$ is in the computing phase then $K$ consists of a base colony-pair with a possible inner bridge connecting it, and the possible boundaries will force this.
4. Suppose now that $s$ is in the transfer phase: then the the drift over $K$ is constant. Suppose that, for example, $Drift = 1$. Look at the description of the transfer phase in Section 4.7. Depending on whether we are in a replacement situation (defined by the *Replace* track) and whether the we are in the first or second sweep, the boundary at the front completely determines the possibilities. The restriction on the addresses mentioned in (H4) above makes sure that there is enough space for the replacement to succeed.

□

**Corollary 5.4** *Let $\xi$ be a tape configuration that is micro-healthy on intervals $A_1, A_2$ where $A_1 \cap A_2$ contains a whole cell body of $\xi$.*

a) *Then $\xi$ is also micro-healthy on $A_1 \cup A_2$.*

b) *If $A_1 \cap A_2$ contains at least $Z/2$ cells and $\xi$ is healthy on both $A_1, A_2$, then it is also healthy on $A_1 \cup A_2$.*

*Proof.* The first part follows from the fact that micro-health is defined by boundaries. In case of the second one, if a $D - zone$ intersects both $A_1$ and $A_2$ then it is contained entirely in one of them. □

**Lemma 5.5** *In a healthy tape configuration, over any interval of size* $< (1/4)ZB$ *there are at most*

$$c_{\text{bndr}} = 5 \tag{5.1}$$

*boundaries between domains.*

*Proof.* Two colony-ends can be close to each other in case of a big gap between two neighbor colonies. Add to this the front, one end of a digression zone, and one point where *BigDigression* changes. □

In a healthy configuration, the possibilities of finding non-adjacent neighbor cells are limited.

**Lemma 5.6** *An interval of size* $< Q$ *over which the configuration* $\xi$ *is healthy contains at most two maximal sequences of adjacent non-stem neighbor cells.*

*Proof.* By definition a healthy configuration consists of intervals covered by full colonies connected possibly by bridges, and possibly stem cells between these intervals. An interval of size $< Q$ contains sequences of adjacent cells from at most two such intervals. □

**Lemma 5.7** *In a healthy configuration, the state of a cell-pair shows whether they are outer, and also their direction towards the front. The Core track of a homogenous domain can be reconstructed from any pair of its cells.*

*Proof.* Whether a cell is outer is computed from its age field. If the cell is outer then *Drift* shows its direction from the front, else the increase direction of *Age* shows it. □

## 5.2 Stitching

We will show that a configuration admissible over an interval of size $> (1/2)QB$ can be locally corrected; moreover, in case the configuration is clean, this correction can be carried out by the machine $M$ itself.

**Definition 5.8** (Substantial domains) Let $\xi(A)$ be a tape configuration over an interval $A$. A homogenous domain of size at least $4c_{\text{stain}}\beta B$ will be called *substantial*. The area between two neighboring maximal substantial domains or between an end of $A$ and the closest substantial domain in $A$ will be called *ambiguous*. It is *terminal* if it contains an end of $A$. Let

$$\Delta = (4c_{\text{bndr}} + 9)c_{\text{stain}}\beta. \tag{5.2}$$

⌟

In Section 6.2, we introduced the notion of *islands*: intervals of size $\leq c_{\text{stain}}\beta B$ with the property that if the configuration is changed in the islands it becomes healthy. Under normal circumstances, there will be at most 3 islands in any interval of size $QB$. The size of a substantial domain assures that at least one of its cells is outside an island, since even three neighboring islands have a total size $\leq 3c_{\text{stain}}B$.

**Lemma 5.9** *In an admissible configuration, each half of a a substantial domain contains at least one cell outside the islands. If an interval of size $\leq QB$ of a tape configuration $\xi$ differs from a healthy tape configuration $\chi$ in at most three islands, then the size of each ambiguous area is at most $\Delta B$.*

*Proof.* The first statement is immediate from the definition of substantial domains. By Lemma 5.5, there are at most $c_{\text{bndr}}$ boundaries in $\chi$. There are at most 3 islands. Between islands and boundaries there are at most $c_{\text{bndr}}+3-1$ non-substantial domains: of sizes $< 4c_{\text{stain}}\beta B$. The islands have a total size $< 3c_{\text{stain}}\beta B$ and the space between boundaries may add at most $c_{\text{bndr}}B$. Adding all these up we get the following multiple of $B$:

$$4 \cdot (c_{\text{bndr}} + 2)c_{\text{stain}}\beta + 3c_{\text{stain}}\beta + c_{\text{bndr}} < (4c_{\text{bndr}} + 9)c_{\text{stain}}\beta = \Delta.$$

□

The following lemma forms the basis of the healing algorithm.

**Lemma 5.10** (Stitching) *In an admissible configuration, inside a clean interval, let $U, W$ be two substantial domains separated by an ambiguous area $V$. It is possible to change the tape on $U, V, W$ using only information in $U, W$ in such a way that the tape configuration over $U \cup V \cup W$ becomes micro-healthy (see Definition 5.2). Moreover, it is possible for a Turing machine to do so gradually, changing and/or enlarging the tape in $U$ or $W$ at the expense of $V$, making a constant number of sweeps over $U \cup V \cup W$, with "small turns" as defined in Section 4.4 at the ends of sweeps.*

After the stitching, the tape configuration may only be *micro*-healthy, as the length of a digression zone or a footprint of a big turn may change slightly.

*Proof.* At any step below, if we find that $U \cup V \cup W$ is micro-healthy then we stop.

1. If $U$ consists of colony cells then let it be extended towards $V$ until a colony boundary or $W$ is hit. Assign all core variables in a way to keep the domain $U$ homogenous.

   Then do the same with $W$. If $V$ gets eliminated then the boundary-pair between $U$ and $W$ is necessarily a legal one.

Assume now that the above operations have still left $V$.

2. If for example $U$ consists of colony cells but $W$ does not, then extend $W$ towards $U$ until $V$ is erased: the boundary obtained must be legal.
3. Assume that both $U$ and $W$ consist of colony cells.
   If both colonies are outer then we can turn all elements of $V$ into stem cells. This situation will not be actually encountered since the front is never near the boundary between two outer colonies.

   If both colonies are inner then turn the cells between them into a bridge from $U$ to $W$.

   If for example $U$ is inner and $W$ is outer then, if the age is not in a the transfer phase towards $W$ then fill $V$ with stem cells; else fill $V$ with bridge cells extending $U$.
4. The case remains when neither $U$ nor $V$ consist of colony cells.
   If one of them consists of stem cells then extend it (does not matter which) towards the other until they meet. If both consist of bridge cells then extend either one of them towards the other until they meet. We will always have to end up with a legal boundary.

□

## 5.3 The healing procedure

The healing procedures `Heal`, `RebuildHeal` and the rebuilding procedure *Rebuild* look as if we assumed no noise or disorder. The rules described here, however (as will be proved later), will also clean an area locally—under the appropriate conditions.

Healing performs only local repairs of the structure: for a given (locally) admissible configuration, it will attempt to compute a satisfying (locally) healthy configuration. If it fails—having encountered an inadmissible configuration—then the rebuilding procedure is called, which is designed to repair a larger interval.

To protect from noise, any one call of the healing procedure will change only a small part of the tape, essentially one cell: so a burst during healing can only have limited impact. Every healing operation starts with a survey zig around its starting point: an illegal boundary or a boundary of a D-zone, called the *center*.

If the survey finds some possible healing to do then it performs one step of it, and returns. Otherwise the "attempt" *fails*; in this case, it will build a rebuilding base, an interval is defined with the help of a new field

$$Rebuild.Base \in \{*, 1\}.$$

Its default value is $*$.

**Definition 5.11** [Rebuilding base] A *rebuilding base* is an interval of $4\Delta$ cells with $Rebuild.Base = 1$. ⌟

Here are the details of healing. Suppose that the `Heal` procedure is called at some position. In what follows, turns will always be small turns, as defined in Section 4.4.2. In any newly created cell, *Drift* is set backwards to the creating cell—to make sure that the head does not get lost on the edge of the infinite vacant space when hit by a burst. When finding homogenous intervals or boundaries, we ignore the track *Rebuild.Base*.

*Survey* Look over an interval $I$ consisting of $4\Delta$ cells left and as many right from the center. (As usual, when a neighbor cell is not found, one is created.) Let $I' \subset I$ be an interval consisting of $\Delta$ cells to the left and as many to the right of the center.

If the ambiguous areas in $I$ cannot be covered by 3 intervals of size $\leq \Delta$ (separated by substantial homogenous intervals) then go to part *Fail*.

If $I'$ is pre-healthy then go to the center and then pass to part *Move*.

Find the first illegal boundary $x$ in $I'$. Attempt to find a substantial domain within $\Delta$ steps on the left of $x$; let $S'$ be the first one.

Attempt to find a substantial domain $S''$ within $\Delta$ steps on the right of $x$. Let $S''$ be the first one. If $S'$ and $S''$ are adjacent (so there is an illegal boundary between them) then go to part *Fail*. Else pass control to part *Stitch*.

*Stitch* Attempt one stitching operation, go to the center and pass control to part *Survey*.

(Note that an ambiguous interval (between $S'$ and $S''$) does not trigger stitching unless it contains an illegal boundary.)

*Move* Repeat:

If you are at an illegal boundary go to part *Survey*.

Else if you are not in a D-zone then make a step towards it.

Else go to part *Adjust*.

*Adjust* You come here only if you are in a D-zone. Let us call the *target position* the point inside the D-zone where the value of *BigDigression* changes, or it center, if it is not changing.

Repeat:

Survey $Z$ cells ahead and $Z$ cells behind the center.

If you find an illegal boundary jump to part *Survey*.

Else if the D-zone has $Z/2$ cells then: if there are rebuilding marks (for example belonging to a rebuilding base), remove one; else go to the target position and finish.

If the D-zone has $Z/2 \pm 3\Delta$ cells then make a step bringing its size closer to $Z/2$ cells; else go to part *Fail*.

*Fail* Survey $Z$ cells ahead and $Z$ cells behind the center.

If you find a rebuilding base then start rebuilding, from a center in its middle.

Else contribute one cell to a rebuilding base (if there is none, start at an illegal boundary) then restart at part *Survey*.

## 5.4 Rebuilding

Just as with healing, we will not mention disorder in describing the rebuilding procedure (since the program does not see it)—but the analysis will take disorder into account. Rebuilding will normally start from a rebuild base as in Definition 5.11 (its middle point will be taken as the center $z$ of rebuilding). This makes sure that, if rebuilding gets triggered by a burst in normal mode, zigging will notice this and call alarm. Rebuilding could also start from big turn starvation (see Section 4.4.2). In order to find the center, a track

$$Rebuild.Half \in \{*, -1, 1\}$$

will show whether we are in the left or the right half of the rebuilding area. The center is the place separating the two halves. We will use the following notion.

**Definition 5.12** Suppose that in a configuration $\xi$, there is an interval $I$ ending in substantial homogenous domains, with at most 3 ambiguous intervals inside, in which it can be changed to become healthy, with a colony $C$ in $\mathrm{Int}(I, 0.2QB)$. Then we will say that $C$ is a *valid* colony of $\xi$ with a *neighborhood* $I$. ⌟

The goal is that

r1) we end up with a new decodable area extending at least one colony to the left and one colony to the right of $z$.

r2) the process does not destroy any valid colony.

Rebuilding happens in a bounded number of sweeps. The number of the current sweep is shown on a track *Rebuild.Sweep*. Zigging is done similarly to the ordinary simulation, but now a frontier zone of $Z/2$ cells similar to the one defined in Section 4.4.1 will be marked explicitly in the track *Rebuild.Addr* of the frontier zone, which is undefined outside it. It will show the (positive or negative) distance of the frontier from the center. Since the center is the place separating the negative and positive values of *Rebuild.Half*, returning to it does not need to rely on the distance as shown in the value of *Rebuild.Addr* carried in the rebuild frontier zone. The actual distance can namely deviate from this value somewhat: it can grow if repairs insert new cells.

In every zigging pass, as the frontier gets advanced, the head shifts the frontier zone and changes *Rebuild.Addr* accordingly (increasing by 1 at right shifts, decreasing

it at left shifts). In more detail: for example at a right shift, the head first moves right from the center of the zone by $Z$, adding a new element to the zone on the right, then left by $2Z$ while adding 1 to *Rebuild.Addr* in the zone, then deletes leftmost element, then moves back to the center. If during all this operation some inconsistency is seen in the tracks *Rebuild.Sweep*, *Rebuild.Half* or *Rebuild.Addr*, then the healing procedure for rebuilding is called (see Section 5.5).

Now there is only a bounded number of big turns, at the end of the sweeps. They are handled similarly but simpler than in normal mode. The *BigDigression* field is used; however, now the frontier will simply move along with the D-zone. Big turn starvation happens when $|Rebuild.Addr|$ grows larger than $5Q$; this will trigger restart.

Here are the stages. Recall the stitching operation from Section 5.2.

*Mark* Extend a rebuilding area over $4Q$ cells to the right and $4Q$ cells to the left from the center $z$. At start, erase the *Rebuild.Base* track of the rebuilding base. So the distance to the edge of rebuilding from the center is $\leq 8QB$, which led us to define $c_{\text{Rebuild}}$ in (2.5).

In a clean area and in the absence of noise, rebuilding can be interrupted only in the following ways by the content of the area encountered.

i1) A *restart event*:

If at the beginning or during leftward marking, a zig movement found on the left at least $Z/4$ cells of the frontier of a rightward marking stage of another rebuilding process, then move back there and start a new rebuilding from there. Similarly, if at the beginning, or during leftward marking, a zig movement finds at least $Z/4$ cells of a rightward frontier zone (of normal mode), then start a new rebuilding from there (after moving back there in a zigging motion).

(In both cases, at the time of restart, zigging motion still shows at least $2\Delta$ cells of the abandoned rebuilding frontier, so for example a normal mode frontier cannot be triggered into a rebuilding restart by a burst.)

i2) Turn starvation as defined in Section 4.4.2. Small turn starvation triggers alarm (thus healing). If $|Rebuild.Addr|$ grows large then rebuilding will restart due to big turn starvation.

*Survey and Create* More details of this stage will be given below. It looks for existing valid colonies, and possibly creates some. As a result, we will have one colony called $C_{\text{left}}$ on the left of $z$ along with its neighborhood as in Definition 5.12, one called $C_{\text{right}}$ on the right of $z$, and possibly some colonies between them. Make all newly created colonies represent stem cells. Direct all the other colonies with drifts and bridges towards $C_{\text{left}}$. The interval covering $C_{\text{left}}$ and $C_{\text{right}}$ will be called the *output interval*

of rebuilding. The pair of neighbor colonies with $C_{\text{left}}$ on its left will be made the current colony-pair.

*Mop* Remove the rebuild marks (address and sweep), shrinking the rebuilding area $R$, starting on its left end, onto the right end of $C_{\text{left}}$.

Marked cells from some interrupted rebuilding may remain even after the mop-up operation. (These may trigger new healing-rebuilding when the head meets them sometime later.)

**Details of the Survey and Create stage**

The complexity of this stage is due mainly to guaranteeing property (r2) above.

s1) Going from left to right, pass through the marked area, and stitch every pair of consecutive substantial domains separated by an ambiguous area of fewer than $\Delta$ cells, just as during healing. But don't create new rebuilding cells as in healing: if the stitch result leaves some illegal boundary, just leave it there.

s2) Pass through again, and look for whole colonies. Mark the cells belonging to whole colonies as such, and mark all other cells as such. The marks should go to a special track $R_1$.

s3) Repeat steps (s1) and (s2), writing the resulting marks onto a special track $R_2$. The following steps will rely on the two tracks $R_1, R_2$ having identical content. If it is discovered that this is not the case, alarm is called. This is important since the colony creation operations are destructive, they should not be triggered by a single burst.

s4) Check if there is a marked colony at least three quarters to the left of the center, whose whole neighborhood as in Definition 5.12 is healthy. If yes, find the closest one. If not, create one making sure it does not intersect any marked whole colony, and its neighborhood is healthy (if necessary overwrite part of the neighborhood with stem cells). Call this colony, found or created, $C_{\text{left}}$. Proceed similarly in finding or creating a colony $C_{\text{right}}$ (disjoint from $C_{\text{left}}$) at least three quarters to the right of the center.

s5) Fill in the area between $C_{\text{left}}$ and $C_{\text{right}}$ and the other marked whole colonies between them: fill these gaps with adjacent stem cells, creating a new colony every time an interval of $Q$ adjacent stem cells has been created.

s6) Make all newly created colonies represent stem cells. Let $C_0$ be the first colony towards the left of the center with at least half of it to the left of the center. Direct all drifts to the left end of $C_0$. Make $C_0$ and its right neighbor the new current

colony-pair (create a bridge from $C_0$ to its right neighbor if needed), and let them represent the start of a healing process on the level of $M^*$.

**Remarks 5.13**

1. Once rebuilding completes, since the state of the current colony-pair simulates the start of healing in $M^*$, the head will continue to the right. There, rebuilding may be called again, but it will not rewrite the colony $C_{\text{left}}$. It may rewrite its right neighbor colony, but not destroy it; so repeated calls to rebuilding will result in progress.
2. The precedence given to rebuilding from left to right will be used in the proof of Lemma 7.18. In the absence of new noise, after a constant number of passes, a certain clean interval $J$ will extend to the right until it reaches the end of a colony pair or of a rebuilding area. If marking was unstoppable in both directions then this might not happen soon, since after disorder is entered and exited, no assumption can be made of the state of the current cell-pair. Reaching the left end of $J$ a new rebuilding process may send the head back to the right end, from which a new rebuilding process can send it back to the left end, and so on. But because marking to the right has precedence, a new left-directed marking started by disorder on the right end would not stop it.
3. Giving precedence to rightward rebuilding has the drawback that one can design a initial configuration in which even in the absence of noise, higher-level structure will never arise even locally. Namely, we can fill the line with short intervals $\ldots, J_{-1}, J_0, J_1, \ldots$ each of which is the start (say of size $3Z$) of a right-directed marking process. Then the head, after moving left on $J_0$, will be captured by the process on $J_{-1}$, then later captured by the similar process on $J_{-2}$, and so on. But we will not need to consider such pathological configurations.

⌟

## 5.5 Healing during rebuilding

Some healing may be needed even within the rebuilding process in case of a new burst, (as shown in Example 11.5). We will call it `RebuildHeal`. We can define a notion of rebuild health similarly to health, using the tuple

$$\mathit{Rebuild.Core} = (\mathit{Rebuild.Sweep}, \mathit{Rebuild.Addr}, \mathit{Rebuild.Half}, \mathit{BigDigression}).$$

During the marking stage in rebuilding, the area outside what is already marked is not subject to any requirements on tracks other than *BigDigression*. (The circumstances mentioned in part i1 above can trigger a restart event, but not a healing process.)

We will say that the configuration is *pre-healthy for rebuilding* if it satisfies all boundary requirements, and *healthy for rebuilding* if also the rebuild frontier zone and the digression zone have $Z/2 \pm 1$ cells. The analogue of Lemma 5.3 and Corollary 5.4 will hold for rebuild pre-health. As there, checking for the actual rebuild health, zigging is needed. The analogue of Corollary 5.4 for rebuild health holds:

**Lemma 5.14** *Let $\xi$ be a tape configuration that is healthy for rebuilding on intervals $A_1, A_2$ where $A_1 \cap A_2$ contains at least $Z/2$ cells. Then $\xi$ is also healthy for rebuilding on $A_1 \cup A_2$.*

Healing for rebuilding is very similar to healing, trying to repair the health of the rebuilding process. In its Survey and Stitch stages, it will repeatedly check, similarly to `Heal`, whether it is possible to recreate pre-health for rebuilding by stitching up to three ambiguous areas. (Stitching is simpler now as there are no rigid colony boundaries to consider.) If yes, it performs one stitching step, otherwise it adds a cell to a new rebuilding base. If stitching is successful, it goes to the Move stage to gradually move back to the D-zone. Once there, it goes to the Adjust stage to remove possible rebuilding base marks, and to correct the size of the zone before finishing.

# 6 Scale-up, isolated bursts

This section first shows how health is restored in the absence of disorder and noise. Then it defines the code $\Phi$ mapping a history $(\eta, Noise)$ of a machine $M$ into a history $(\eta^*, Noise^*)$ of the simulated machine $M^*$. For this, it introduces the notion islands in the framework of an "annotated" history. Finally, using the introduced terminology, it shows that the healing procedure indeed deals with isolated bursts. For the elimination of disorder created by faults we will rely on the Escape, Spill Bound and the Attack Cleaning properties of a trajectory in Definition 2.13.

## 6.1 Restoring health in the clean, noiseless case

An interval rewritten by noise can have $Pass \neq 0$ everywhere or many footprints of a big turn too close to each other even if it is clean, so we define a property of intervals avoiding this.

**Definition 6.1** An interval will be called *safe for small left turns* if it has no more than $3\Delta$ consecutive cells with $Pass = 1$. And if it has more than $\Delta$ then it is preceded by a footprint of a big left turn. It is *safe for big left turns* if it has no sequence of more than $3F \log Q$ footprints of a big left turn closer than $2F$ cells to each other. And if it has more than $2F \log Q$, then on the left of this sequence there is a colony encoding a big

cell with $Pass = 1$. It is *safe for left turns* if it is safe for both small and big left turns. It is *safe for turns* if it is safe for both left and right turns.

We say that the interval is *weakly* safe for turns if the upper bounds $3\Delta$ and $3F \log Q$ are replaced with $6\Delta$ and $6F \log Q$. ⌟

Note that in order to be safe for turns, the interval only has to be safe for left turns on its part to the right of the head and safe for right turns on the part to the left.

**Lemma 6.2** *If a clean interval is passed over from left to right by a path $P$ having at most one burst, then it becomes safe for turns. If it is passed from right to left then it becomes safe for small turns; but if this pass happens after a left-to-right pass then it also stays safe for all turns.*

*Proof.* We will present the proof for a pass from left to right, and point out the only difference for the case when the pass is from right to left, in part 2 below.

We will consider the case when no burst occurs. The path with a possible burst can be divided into three parts $P_1, P_2, P_3$: before the burst, the part when the disorder created by the burst is crossed over, possibly several times, and the part when the path leaves this place behind definitively. Parts $P_1, P_3$ are handled below. Due to attack cleaning, $P_2$ can cross the disorder at most $\beta$ times, these can modify the estimates in parts $P_1, P_3$ only by $\beta$ cells.

Let $x_1 < x_2 < \cdots < x_n$ be the points of the interval left behind with $Pass = 1$, and let $t_i$ be the times when this happens at $x_i$.

1. Consider the space-time points $(x_i, t_i)$ where the head makes a big turn in rebuilding mode. In rebuilding mode, big turns happen at the end of sweeps—there is only a constant number of them—so before time $t_i$, the head must have performed at least one complete sweep of rebuilding. If this rebuilding succeeds then it leaves a healthy area containing at least one colony on the left and one on the right of its center. Another rebuilding can only start at the left or right of this interval. It cannot be on the left, since $t_i$ was the last time when $x_i$ was passed. So a next rebuilding big turn can only happen about $QB$ cells to the right of $x_i$.

   If the pass is from left to right then this rebuilding can only fail by big left turn starvation, see Section 4.4.2. This must happen at a distance at least $2QB$ to the intended left turn. If any later rebuilding has its left end within $QB$ of $x_i$ then it will already succeed, and we can reason as above.

2. In case the pass is from right to left then another possible way that the rebuilding can fail is when the leftward rebuilding is overridden by a new rightward rebuilding, see the Marking part of the rebuild procedure in Section 5.4. As a result, the right-to-left turns may get close to each other, so safety for big turns will not be guaranteed.

However, when the interval was previously passed from left to right then, as seen above, interrupted rebuildings can only happen because of turn starvation. As seen above, they are placed at a distance $> 2QB$ from each other, and big turns will remain safe.

3. Consider now the space-time points $(x_i, t_i)$ where the head makes a big turn in normal mode. Suppose first that the simulation program may be interrupted by healings, but these healings all succeed, so rebuilding is not triggered. We only need to consider big turns made in the turn region at a left end of a colony $C$ which is the left element of a colony-pair, during the starting and ending sweeps: the footprints inside will be overwritten by the ending sweeps. If the work period finishes normally, and a healthy colony $C$ remains to the right of $x_i$, then big turns in normal mode not belonging to this work period will only be made at least $\approx QB$ to the right.

   In normal mode, $C$ may be replaced with another colony $C'$, overlapping it. Then $C'$ will already not be replaced (without going to another colony on its left), so the big right turns on its left are necessarily separated on the right from others by at least $\approx QB$, but the turns on the left of $C'$ can be close to the previous ones on the left of $C$.

   It is also possible that rebuilding will be called before the work period over $C$ finishes. (This can only happen if then this rebuilding experiences big left turn frustration on its right, since otherwise it would sweep over $x_i$.) Such a big left turns in normal mode at the left end of a rebuilt colony $C'$ near $x_i$ can occur only once, since rebuilding created a colony-pair $(C', C'')$.

4. Consider the points $(x_i, t_i)$ where the head turns during healing. If $i < n$ this healing cannot fail, since then the subsequent rebuilding would bring the head to the left of $x_i$, contradicting the assumption that $t_i$ was the last time when it was there. It could, though, experience small turn starvation. But if healing eventually restarts near $x_i$ then it will not experience small turn starvation again and succeeds, so any following turn points $(x_j, t_j)$ due to healing will be at least $\Delta$ away.

   The analysis is similar for points $(x_i, t_i)$ where the head turns in the part of rebuilding when it is attempting to stitch an ambiguous area.

5. What remains is space-time points $(x_i, t_i)$ where the head makes a small turn in normal or rebuilding mode. In these modes the head makes a zig only in every second step, so normally these places also don't occur consecutively; the violations of this may happen only during healing and are limited as discussed above.

□

**Lemma 6.3** (Combined heals) *Let $d = 3ZB$. Assume that the head moves in a noise-free and clean space-time rectangle $J \times K$ with with $2d < |J|$, $|K| \geq T^*$, touching every cell of $J$ at least once, and never in rebuilding mode. Assume also that at the beginning, $J$ is safe for small turns. Then during $K$, the area* $\mathrm{Int}(J, d)$ *becomes healthy.*

*Proof.* Since $J$ is safe for small turns, in what follows we will not see small turn starvation. Since big turn starvation would trigger rebuilding and we don't see rebuilding, we will not encounter big turn starvation either.

If healing was not called then after the head touched every cell of $J$ the health of the area is proved. If the head entered $J$ during healing then before finishing healing, it touches over an area of size $\leq 2ZB$. The upper bound $3\Delta$ on the number of consecutive cells with $Pass \neq 0$ makes sure that the turns happen within $3\Delta$ cells of both ends of this area, increasing its size to at most $d = 3ZB$.

Consider some time when healing is started while the head is in $J$, and let $I_1$ be the interval $I$ defined in the healing procedure for this point. Since rebuilding is not started, the stitching part of healing succeeds, with the interval $I_1'$ becoming pre-healthy, while staying in an area of size $d$. After this, the Move part of healing is trying to move the head towards a D-zone. New healing starts only at some illegal boundary $x$ outside $I_1'$, and the interval $A$ containing both $x$ and $I_1'$ is pre-healthy at this time. If the head does not leave $J$ during this procedure (which is the case if $x \in \mathrm{Int}(J, d)$), this stitching also succeeds, creating a new pre-healthy interval $I_2'$ that intersects $A$ in an interval of size $\geq \Delta B$. Hence $A \leftarrow A \cup I_2'$ becomes pre-healthy.

This process continues until the head arrives at a D-zone, when it starts the Adjust part of healing in a pre-healthy interval $A$; this can be interrupted by an illegal boundary again, triggering new stitching, but since rebuilding is not started, it eventually succeeds, creating a healthy interval $A$ that contains all areas surveyed until now. Now normal mode resumes, moving the D-zone and eventually moving the front, until a new illegal boundary is found. This way the healthy interval in which the head is moving is getting extended, and the only parts of $J$ not becoming healthy are confined to the borders of size $d$.

The healing procedure can only create at most $\Delta$ consecutive cells with $Pass \neq 0$. Zigging occurs only every 2 steps, so it does not create solid intervals with $Pass \neq 0$.

The number of steps in normal mode is at most as much as our bound on the number of steps in a work period. This changes by at most a factor of 2 due to the delays in healings, so the bound $|K| \leq T^*$ is sufficient. □

## 6.2 Annotation, scale-up

Let us define the notion of "almost healthy" (admissible) for histories.

Informally, an admissible configuration may differ from a healthy one in a small number of intervals we will call "islands". Even a healthy configuration may contain some intervals called "stains": places in which the *Info* track differs from a codeword. These pose no obstacle to the simulation, and if they are small and few then will be eliminated by it, via the error-correcting code.

Annotation will interpret parts of a history, "covering up" small segments that are not quite healthy. It will leave other parts uninterpreted.

**Definition 6.4** (Annotation) Recall the definition of $\beta'$ and $c_{\text{stain}}$ in (4.5). Let

$$Q' = Q - 2PadLen$$

be the number of cells in the interior of a colony. Then any healthy interval of size $Q'B$ intersects with at most one turn region at the end of some colony.

A *annotation* for a history $(\eta, Noise)$ over a time interval $[0, u]$ is a tuple

$$(R, \chi, \mathcal{I}, \mathcal{S})$$

with the following properties, for a certain constant

$$c_{\text{relief}}.$$

a1) $R$ is a subset of $\mathbb{Z} \times [0, t]$ called the *range* of the annotation; denote $R(t) = \{x : (x, t) \in R\}$.

a2) $\chi$ is a history over $R$, there are no bursts of $Noise^*$ over $R$, and $\chi(\cdot, t)$ is healthy over $R(t)$.

a3) The disorder in $R$ is covered by a set $\mathcal{I}$ of connected space-time regions called *islands*. Elements of the set $\mathcal{S}$ are connected space-time regions in $J$ called *stains* whose space projection has size $\leq c_{\text{stain}}B$. Each island is contained in a stain. We will write $\mathcal{I}(t)$ for the set of islands at time $t$; in other words, $L$ is in $\mathcal{I}(t)$ iff $L \times \{t\}$ belongs to $\{K \cap (\mathbb{Z} \times \{t\}) : K \in \mathcal{I}\}$. Similarly for stains.

At any time, $\mathcal{I}(t)$ and $\mathcal{S}(t)$ consist of intervals.

$\eta$ differs from $\chi$ over $R$ only in the islands.

a4) In any one island, the head does not spend more total time (even when entering and exiting possibly several times) than $c_{\text{relief}}(\beta^2/\gamma)\pi qT$.

a5) At any time $t$, any segment of size $QB$ contains at most 3 islands, and at most 1 can be in the interior of a colony. At most 2 stains can be in the interior of a colony.

a6) If at some time no cell of a colony of $\chi$ belongs to the update phase then the *Info* track of the interior can be changed in the stains in such a way that it becomes a codeword of the code $\upsilon$ as in Definition 4.4.

a7) Every interval of size $Q'B$ is weakly safe for turns.

In an annotated history, we will say that the head is *free* if it is at a distance of at least $Z$ steps from any island. ⌟

A configuration may allow several possible annotations; however, since the code defined in Section 4.5 is $(c_{\text{stain}}\beta, 2)$-error-correcting, the codewords recoverable from it do not depend on the choice of the annotation.

Formally, the proof of error-correction would proceed by proving that annotation can essentially be extended forward in time; however, we will retain an informal language whenever it is clear how to translate it to annotation. Example 11.4 shows how three islands may arise, along with an informal argument that local correction does not have to deal with more than three islands in any area of size $Q'B$.

Let us now define formally the codes $\varphi_{*k}, \Phi^*_k$ needed for the simulation of history $(\eta^{k+1}, Noise^{(k+1)})$ by history $(\eta^k, Noise^{(k)})$. Omitting the index $k$ we will write $\varphi_*, \Phi^*$. To compute the configuration encoding $\varphi_*$ we proceed first as done in Section 3.5, using the code $\psi_*$ there, and then initialize the kind, sweep, drift and address fields appropriately. The value $Noise^*$ is obtained by a residue operation as in Definition 2.4; it remains to define $\eta^*$. In the parts of the history that can be locally annotated, and which we will call *clean*, if no colony has its starting point at $x$ at time $t$, set $\eta^*(x,t) =$ Vac. Otherwise $\eta^*(x,t)$ will be decoded from the *Info* track of this colony, in its work period containing time $t$. More precisely:

**Definition 6.5** (Scale-up) Let $(\eta, Noise)$ be a history of $M$. We define $(\eta^*, Noise^*) = \Phi^*(\eta, Noise)$ as follows. Consider position $x$ at time $t$, let $I = [x - QB, x + 2QB)$, $J = (t - T^*, t]$. If $\eta(\cdot, t)$ cannot be annotated in $I \times J$ then $\eta^*(x,t) = \text{Bad}^*$. If $x$ is not the start of some colony $C$ in this annotation then let $\eta^*(x,t) =$ Vac; assume now that it is. Then let $t'$ be the last time when the head is not in an island and its age is not in the update phase, and let $\eta^*(x,t)$ be the value decoded from $\eta(C, t')$. In more detail, as said at the end of Section 4.1, we apply the decoding $\psi^*$ to the interior of $C$ to obtain $\eta(x,t)$. ⌟

## 6.3 Dealing with isolated bursts

Definition 6.5 decodes trajectories $(\eta, Noise)$ into histories $(\eta^*, Noise^*)$. We don't know yet whether trajectories of $M$ are decoded into trajectories of $M^*$. Let us give an informal argument first.

Isolated bursts don't create disorder larger than $\beta$. The head escapes a disorder interval $I$ via the Escape property; while it is inside, the spreading of this interval is limited by the Spill Bound property. Every subsequent time when the head enters and exits $I$ this gets decreased via the Attack Cleaning property, so it disappears after $O(\beta)$ such interactions—see Lemma 6.10 below.

Let us first show that annotation implies that the job of simulation proceeds as required.

**Lemma 6.6** *Consider an annotation $(R, \chi, \mathcal{I}, \mathcal{S})$ of a trajectory $(\eta, Noise)$ over a rectangle in $R' \subseteq R$ which the head never leaves.*

a) *The decoded history $(\eta^*, Noise^*)$ over $R'$ satisfies the Transition Function property of trajectories (Definition 2.13).*

b) *Assume that, in addition, no bursts occur over $R'$, let $t$ be the end of a complete work period of some colony-pair, and $C = [x, x + QB)$ one of its colonies. Then*

$$\xi = \eta(\cdot, t) \restriction C = \phi_*(\eta^*(x, t)).$$

*In other words, the colony $C$ will have no stains at time $t$; each of its cells has the value assigned to it by the code $\phi_*$.*

*Proof.* Consider a sequence of configuration in $R'$ starting from one corresponding to the beginning of a working period of a colony-pair, at a time when the head is free. Properties (a3) and (a4) limit the space occupied by each island as well as the time that a head can spend in it. While the head is free it is carrying out the simulation program. Property (a7) makes sure that no turn starvation slows it down. The error-correcting decoding will recover the state of the simulated cell from each colony since it needs to recover from at most two stains.

Any island in the interior of the colonies will be eliminated during the initial sweeps of the work period, as this is the only way the head can be freed from it. The computation part of the simulation is repeated three times. A new burst can occur in at most one of these repetitions; if it appeared in the $i$th repetition, the result on the $Hold_i$ track may be worthless. But the two other $Hold_j$ tracks will contain the correct results, and the majority voting will recover it. If no new bursts occur then the majority voting will recover everywhere the correct result, showing (b).

The length of the work period is upper-bounded by $T$ times the number $O(QFZ^2)$ of computation steps (see Section 4.6.4) while the head is free plus the time while the head was not free. Given that the total time spent in any one island is $\leq c_{\text{relief}}\beta^3 qT$ and there are at most 3 islands per colony, the total time in islands is $O((\beta^2/\gamma)\pi qT)$: adding this to $O(QFZ^2T)$, we still stay below the upper bound $T^*$.

□

Let us proceed to proving that in the absence of $Noise^*$, annotation can be extended on the range $R$. More precisely, we will introduce a game.

**Definition 6.7** [Annotation game] The game is played over a trajectory $(\eta, Noise)$ of a machine $M$. Its two players are Range Extender and Annotator. Range Extender is in charge of a sequence of times $0 = t_0 < t_1 < t_2 < \ldots$, and in extending the range. At any stage of the game, it assumed that the annotation is defined up to time $t_i$, that is part of the range $R \subseteq \mathbb{Z} \times [0, t_i)$ is defined as well as the annotation over it. Now the Range Extender extends the range over the time interval $[t_i, t_{i+1})$ in two possible ways.

1) It deletes an interval $I$ containing the head from $R$. (We still talk about "extending" since the range is extended in time.) More precisely, it defines

$$R \leftarrow R \cup (R(t_i - 1) \setminus I) \times [t_i, t_{i+1}) .$$

It also trims all islands and stains, deleting their parts in $I$, and keeps $\mathcal{I}(t)$, $\mathcal{S}(t)$ constant over $[t_i, t_{i+1})$.

2) It adds an interval $I$ containing the head to $R$. More precisely, it defines

$$R \leftarrow R \cup (R(t_i - 1) \cup I) \times [t_i, t_{i+1}) ,$$

where $I$ has the property that by defining $\chi(x, t_i) = \eta(x, t_i)$ over $I$ and setting $\mathcal{I}(t_i) = \mathcal{I}(t_i - 1)$, $\mathcal{S}(t_i) = \mathcal{S}(t_{i-1})$ we still get an annotation. Also $\eta$ is safe for turns over $R(t_i)$.

In both cases, the Range Extender can extend the range only in such a way that $Noise^*$ remains empty over the new range.

The Annotator must respond by extending the definition of $\chi, \mathcal{I}, \mathcal{S}$ over the new range, that is up to $t_{i+1}$, in such a way that the resulting structure is still an annotation.

⌟

Two remarks:

- The players are "clairvoyant": they see the whole trajectory $(\eta, Noise)$ ahead as well, not just the parts that has been annotated already.
- The players should not be considered adversaries: rather, they cooperate in creating the annotation of the whole trajectory.

Below, we will show that under the conditions, Annotator can always respond; this way, the effect of sparse bursts will be corrected, leading to the Transition Function property of the simulated trajectory $(\eta^*, Noise^*)$. In later sections we will show the other properties, and also how they can lead to appropriate choices of the Range Extender player. Note in particular that in this game, as Lemma 6.8 shows, every island is the result of a burst that occurred during $R$.

In a clean configuration, whenever healing started with an alarm, the procedure will be brought to its conclusion as long as no new fault occurs. However, every time the head emerges from disorder, we cannot assume anything about the state of the cell-pair to which it arrives. This complicates the reasoning, having to consider several restarts of the healing procedure. By design, this procedure can change the *Core* track only in one cell. The following lemma limits even this kind of possible damage.

**Lemma 6.8** *In the absence of noise, no new island will arise.*

*Proof.* The islands are defined only by the tracks in the *Core* group. In normal mode, these tracks change only at a boundary point.

The healing procedure changes the *Core* as part of a stitching operation, or removing or adding a rebuild mark. The proof of Lemma 5.10 shows that inside a healthy area, healing can only change the *Core* track by moving the boundaries around. These operations don't affect pre-health; they may temporarily change the size of the frontier zone or the D-zone by up to $3\Delta$, but then the adjust stage restores these sizes. □

The following lemmas are central to the analysis under the condition that bursts are isolated. Let us first show how the Escape and Pass Cleaning property of trajectories helps bringing the head into clean intervals, even if these are rather small.

**Lemma 6.9** *Let $\gamma' = \gamma/2$. Consider a noise-free path $P$ starting at some time $t_0$ at a point $b_0$. Create points $b_j = b_0 + j\gamma' B$ for (positive and negative) values of $j$. Let us call the intervals $\big[b_j, b_{j+1}\big)$* blocks*. Assume that the disorder that the path may encounter is covered by $n$ blocks. Then there is a sequence of times $t_0 < t_1 < \ldots$ with $t_{i+1} - t_i \le qT$, such that during the time intervals $(t_i, t_{i+1}]$ called* skips *the head passes over some block $H_i$ either leftwards or rightwards, further except for $2n\pi$ skips, the interval $H_i' = \mathrm{Int}(H_i, (c_{\mathrm{marg}} + c_{\mathrm{spill}})B)$ is clean. Consequently, the first such skip happens for some time $t_i \le (2n\pi + 1)qT$.*

By (2.6), $|H_i'| > 0$.

*Proof.* Suppose that time $t_i$ has been defined and the head is at point $b_j$ at this time. The interval $\big[b_{j-1}, b_{j+1}\big)$ has length $2\gamma' B = \gamma B$, so by the Escape property of trajectories and (2.6), the head will escape it within time $qT$. Let $t_{i+1}$ be the time when it arrives at $b_{j-1}$ or at $b_{j+1}$.

We claim that the number of skips $i$ in which $H_i'$ is not clean is at most $2n\pi$. Indeed, the total number of possible blocks containing disorder originally is $\le n$. If $H = H_i$ for a right skip $i$ for $\pi$ times, then the Pass Cleaning property implies that $\mathrm{Int}(H, c_{\mathrm{marg}}B)$ becomes clean; then by the Spill Bound property, $H'$ stays clean. Similarly for left skips, so $H = H_i$ without this for at most $2\pi$ values of $i$. □

**Lemma 6.10** (Healing) *In the annotation game, Annotator can always respond.*

*Proof.* A space-time point is a *distress event* if either a fault occurs there or the head steps onto an island. The extension of annotation is straightforward as long as no distress event is encountered. If after a distress event the head becomes free (according to Definition 6.4), then we will say that *relief* occurred. Let us see what can occur between a distress and relief event.

1. Every time that the head enters disorder, it will leave within time $21(\beta/\gamma)q\pi T$.
   *Proof.* The disorder is covered by 3 intervals of size $\beta'$, so we can set in Lemma 6.9, using (4.5) and (2.6):
   $$n = 3(\beta'/\gamma' + 2) \le 10\beta/\gamma.$$
   According to the lemma, the head will leave within time $(2n\pi + 1)qT$.
2. The head can leave disorder in any one island at most $\beta' + 1$ times. Hence in any interval of size $Q'B$, if there are $\le 3$ islands then the head can leave the disorder at most $3(\beta' + 1) \le 4\beta'$ times.
   *Proof.* The size of the smallest interval covering disorder in any island is at most
   $$d = \beta' B.$$
   By the Attack Cleaning property, every time the head leaves it on a side of the disorder interval $D$ where it had entered before, it shrinks $D$ by $B$. So every such visit does this, except possibly the one time when it exited without entering before.
3. The size of stains will never grow beyond $c_{\text{stain}}B$. Also the total change of the front and the *Age* variable between distress and relief events connected with one island is not more than $3\Delta$.
   *Proof.* The disorder created by a burst can always be covered by an island identical to it. We can always define islands to end at illegal boundaries. Suppose that the boundary cell-pair is clean. If the head entered it in normal mode it would start healing. If it entered as part of a healing procedure started in the island, then this procedure would not change the boundary in the direction of increasing the island. Rebuilding would only start in the island by a disorder. These rebuilding cells are counted as part of the island, so by this bound on their number, a rebuilding base will not be created, and rebuilding will not be started by the healing procedure.

   So an island can increase only as a consequence of the head leaving disorder. It increases by at most $B$ in each such stage, and by part 2 there are at most $\beta' + 1$ such stages. We started from an island size $\le \beta' B$, therefore an island never grows larger than $(2\beta' + 1)B$.

Since between distress and relief the *Age* and address of the front would only change at the boundaries of the substantial domains, the estimate on the size of change on them follows.

4. Between two visits to disorder, if the head does not become free, it can spend only $O(\beta Z)$ steps.

   *Proof.* Indeed, there can only be $O(\Delta) = O(\beta)$ steps involving a (Survey) followed by a (Stitch). Each (Move) steps takes one closer to the D-zone, and at the start of healing, the D-zone was within a distance of $(3/4)Z$ cells. The number of calls to (Adjust) is at most $4\Delta$, each is a sweep of $\leq 4Z$. The same counting holds for (Fail).

5. The total time between distress and relief is $\leq 64(\beta^2/\gamma)\pi qT$. Outside distress, the head is at the front in normal mode.

   *Proof.* Initially there are no islands. By the Spill Bound property, the disorder can grow to at most $3\beta' B$.

   By 2, between a distress and relief, there are at most $\beta' + 1$ visits to disorder.

   By 1, every time the head enters disorder, it leaves within time $21(\beta/\gamma)\pi qT$.

   By 4 the time between two such visits to the same disorder is $O(\beta ZT)$. So the total time between visits is $O(\beta^2 ZT)$, which is dominated by our bound $3\beta' \cdot 21(\beta/\gamma)\pi qT$ on the total time the head needs to leave disorder.

6. By Lemma 6.6 if a rectangle $R'$ was annotated until time $t$ when the head is at the beginning of a work period of some colony-pair and then the annotation is extended to the end of the work period, the resulting configuration will be according to the transition function of the simulated machine $M^*$. After the decoding and the computation, the earlier stains get eliminated; new stains only arise in new islands, so they remain bounded again.

7. Condition (a5) on the number of islands is met.

   *Proof.* A crucial observation is that in the normal course of simulation, if the head encounters an island and must pass over it then the island will be eliminated, since repeated zigging will notice it again and again.

   We started with a clean configuration. A burst can leave an island $I_1$ in a colony $C$, if it happens in the last sweep of the work period. Much later the head may return, say from the left. Then it may not pass over $I_1$ only if $I_1$ was in the right turn region of $C$ and after a work period it the head moves left again. In the last sweep it may leave another island $I_2$, and this is the only way for two islands to arise. But in this case the cell simulated by $C$ has $Pass = 1$. So when returning a third time, the simulated head will continue right and will eliminate necessarily both islands. Before doing this it may create a new island, this way temporarily increasing the number of islands to

three; however, after it leaves $C$ there will be at most one island left.

8. The extended annotation satisfies property (a7) of annotation.

   *Proof.* An interval of $Pass \neq 0$ can be created by the healing procedure as it makes several sweeps needed for its stitching operations. Given the bound $\Delta$ on the size of ambiguous areas, only $\Delta$ stitches should be needed, so $3\Delta$ is a generous upper bound on the size of an interval of cells where new turns were made. Once an island is eliminated and its place is passed over, these traces of small turns are also erased, leaving only the ones at the bottom of zigs. But zigs are made only after every two steps of progress, so these new *Pass* signs will (almost) never be consecutive. By the restriction on the player Range Extender, each extension starts with a configuration that is safe for turns, so has a bound of $3\Delta$ on the number of consecutive cells with $Pass \neq 0$; the additional $3\Delta$ may increase this bound temporarily to $6\Delta$, but any new islands will give rise to at most $3\Delta$ consecutive cells with $Pass \neq 0$.

   A similar argument applies to footprints of a big turn. The number of big turns in a work period is less than $Q^2$, so by the argument seen in Section 4.4.2, they give rise to no more than $2 \log Q$ new consecutive footprints of a big turn, say in the right turn region of a colony. Safety for turns imposed on the Range Extender allows $3 \log Q$, but even when counting $2Q^2$ new ones in two new work periods, the bound $6 \log Q$ of (a7) is satisfied. After two work periods, these footprints will necessarily be erased due to feathering in the simulated machine $M^*$.

   □

# 7 Cleaning

This section will scale up the Spill Bound, Escape, Attack Cleaning and Pass Cleaning properties of trajectories, proving them for the history $(\eta^*, Noise^*)$ decoded from a trajectory $(\eta, Noise)$.

## 7.1 Escape

We will scale up the Escape property in Lemma 7.9 below; here is an outline of the argument. Consider some fault-free path during a time interval $J$ (later we will allow a single burst) over some space interval $G$ of size $|G| = \gamma QB$. For the times $t \in J$, let $K(t)$ denote the set of those clean points in $G$ that the head passed at least once since they were clean. Then by Lemma 6.2, this set consists of intervals that are safe for small turns. The goal is to show that the path will not stay too long in $G$. This will be since if it stays long then it enlarges $K(t)$, and then builds up colonies in it. These simulate the machine $M^*$, which commands its head to swing wide according

to the program (in zigging or healing), and thus leave $G$. Initially, the clean intervals of $K(t)$ can be created using the Pass Cleaning property of trajectories. Every time the head leaves such an interval, the latter grows via the Attack Cleaning property; so we will mainly be concerned with longer stays. The notion of "long" will be chosen here to make sure that most of it has to be spent in simulating $M^*$, since both healing and rebuilding finish relatively fast.

**Definition 7.1** In the work period of a colony-pair, let us call the phase during which the *Info* track and the *Drift* track is updated, the *update phase*. We will say that the pair is *right-directed* if the following holds.

If the age of all cells is before the update phase, and the colony-pair represents a pair of cells $a, b$ of the machine $M^*$, then the transition function of $M^*$ applied to $a, b$ will direct the head right.

If the age is after the update phase then the *Drift* track contains 1.

Left-directedness is defined similarly. ⌟

Time intervals of length $T$ we may consider as *steps*, since under clean and noiseless conditions, the machine $M$ will perform at least one step of computation during each. Recall $U_k = U = c_U Q \pi^9$ in Definition 2.14. This is an upper bound on the number of computation steps in one work period, even allowing some calls for healing. The lemmas below follow the development of a maximal interval $I(t)$ of $K(t)$. We will need some new, temporary concepts. Recall the notion of a *restart event* from Section 5.4, when the leftward marking process of rebuilding encounters a right-directed frontier and this causes the rebuilding to restart. In the present context, we will consider such restart events within $K(t)$ only if it was preceded by leftward marking that covered an area of at least $Z$ cells.

**Lemma 7.2** *The $\geq Z/4$ cells involved in a restart event (or their descendants if they are shifted) will not be involved in any other restart event later.*

*Proof.* The restart event overwrites the frontier cells by the new frontier of rebuilding. Any new frontier created later can be left behind only due to small turn starvation. (For big turns, the rebuilding process does not leave behind the frontier zone.) But when the head arrived from the right over an area of at least $Z$ cells, this area became safe for turns, and in the absence of bursts, it will remain so. □

**Corollary 7.3** *The total number of restart events is at most $4\gamma Q/Z$.*

**Lemma 7.4** a) *No maximal subinterval of $K(t)$ ever decreases by more than $c_{\mathrm{spill}}B$ on either side.*

b) *In any subinterval of $K(t)$ of size $\leq nQB$, the amount of time the head can spend is at most the time spent on rebuildings (possibly interrupted but only if done so by restarts), plus $2nUT$.*

*Proof.* 1. (a) follows from the No Spill property of trajectories.

2. On (b): While no rebuilding starts, we can apply Lemma 6.3. Thus, eventually the computation in normal (or, similarly, booting) mode leads to the simulation of cells in $M^*$. The program of $M^*$, just like that of $M$, proceeds by sweeps (zigging or healing). Even the shortest of these sweeps has size at least $2\beta QB > \gamma QB$, therefore a full sweep will exit $I(t)$ in $\leq nUT$ steps.

   If any rebuilding has been started, it will finish in $O(QZ)$ steps unless interrupted by a restart event (see above) or big turn starvation, see Section 4.4.2. If it succeeds, it creates a colony-pair that simulates a cell-pair at the start of healing, hence starting a sweep of size $> \beta QB$ to the right. It may trigger rebuilding again, but this rebuilding results in a new colony-pair of the same kind, in $O(QZ)$ steps, at the place where it started, at least $QB$ to the right of the last working colony-pair. So exit happens again in $\leq nUT$ steps.

   Consider now rebuildings that don't succeed. We are counting the time spent on rebuilding interrupted by a restart separately, so consider the ones interrupted by big turn starvation. These can occur at most $\gamma n/4$ times, since they occur only after the head moved to a distance $\geq 4QB$ from the rebuilding center—removing the center of a new rebuilding at least this far from the previous one.

□

An interval $I$ of size $QB$ in $\mathrm{Int}(K(t), 2B)$ will be called a *manifest colony* if it is healthy with the possible exception of having some rebuild marks (only for survey, not for decision), and has undergone a complete simulation work period as part of a colony-pair in a clean subinterval of $K(t)$. In a manifest colony, unless it is at a distance $\leq 2QB$ from the head in the same interval of $K(t)$, the *Drift* track points towards the head.

**Lemma 7.5** *The number of manifest colonies does not decrease.*

*Proof.* Simulation or healing does not destroy any part of a colony. It may shift a colony, if the simulation work period of a colony-pair encounters a replacement situation, see Section 4.7. Let us see that rebuilding does not destroy them either.

In the definition of manifest colonies we did allow some (possible leftover) rebuild survey marks, but not decision marks. In order to destroy a colony, the rebuilding process needs to create two decision tracks. One has to consider also the case when the rebuilding process is at one end of $I(t)$, hence is fed some uncontrollable information.

The head can exit during a rebuilding process only if it is in its starting stage, marking its interval of operation. Zigging along with attack cleaning implies that even if the head enters and exits $K(t)$ multiply, by decision time the whole rebuilding interval will have to be incorporated into $K(t)$, therefore the decision will be a correct one, not destroying a manifest colony. □

Suppose that during a stay in $K(t)$ a rebuilding process is started and completed. Its result is a pair of neighbor manifest colonies, on the left and right of the center from which rebuilding started. Let us call these the *left result* and *right result* of rebuilding.

**Lemma 7.6** *A manifest colony can only become a left result once, and a right result once.*

*Proof.* The smallest interval $D$ containing the resulting colony-pair will be healthy and safe for turns at the time when rebuilding finishes. The following development will never introduce inconsistency into $D$, other than rebuild marks resulting from some rebuilding process started outside it.

The only way in which a rebuilding process can start even in a healthy area is big turn starvation, as defined in Section 4.4.2. But in the present case, the rebuild process looking for a big turn must have started looking for a turn outside $D$, and since $D$ is safe for turns, it would have found a turning point close to an end of $D$, so the left colony of $D$ could not become its left result, nor the right colony of $D$ its right result. □

We will say that the stay of the head in some maximal interval of $K(t)$ is *short* if the part of it remaining *after subtracting the time spent on rebuild procedures interrupted by a restart* is smaller than for $2UT$ for $U$ as in Definition 2.14, otherwise it is *long*.

**Lemma 7.7** *Each long stay either adds a new manifest colony, or joins two subintervals of $K(t)$ of size $\geq QB$, or creates a new left result and a new right result as defined in Lemma 7.6.*

*Proof.* Consider some maximal interval $I(t)$ of $K(t)$, and a long stay in it.

1. Suppose first that no rebuilding process is triggered or continued during the stay; then only healing and computation steps are possible.

   Suppose that there was no manifest colony in $I(t)$ before entry. The stay is long enough that at least one complete work period will be performed on a neighbor colony-pair. So by the time the head leaves, there will be at least one manifest colony; in fact the exit will happen during a transfer process from a manifest colony (possibly slowed down by healing).

   In general, whenever the exit happens after a long stay then it either happens this way or during the marking stage of a rebuilding process.

Suppose there were manifest colonies at entry time, and the head enters, say, on the left. Let $C_0$ be the leftmost manifest colony of $I(t)$. The head can pass to $C_0$ only as a consequence of a transfer process from some colony $C_{-1}$. So the long stay either adds $C_{-1}$ as a new manifest colony, or joins $I(t)$ with another subinterval of $K(t)$ on its left, which contains a manifest colony $C_{-1}$.

2. Suppose now that a rebuilding process starts before a manifest colony could have been added as described above. This process could be restarted repeatedly, but we don't count the time spent on the rebuilding processes interrupted by a restart. Since the stay is long, one of them has to succeed; on termination, it creates a left result and a right result.

□

**Corollary 7.8** *The number of long stays is at most* $3\gamma$.

*Proof.* There are at most $\gamma$ manifest colonies in $K(t)$, so there can be at most $\gamma$ creation events. There are at most $\gamma - 1$ events of joining two disjoint subintervals of $K(t)$ of size $\geq QB$. Hence the total number of long stays of kind 1 is at most $2\gamma - 1$, and the total number of long stays of kind 2 is also at most $\gamma$. □

The following lemma is the scale-up of the Escape condition.

**Lemma 7.9** (Escape) *Let $q$ be as introduced in* (2.5). *In the absence of Noise$^*$, the head will leave any interval $G$ of size $\gamma QB$, within time $q^*T^*$.*

*Proof.* Consider a time interval of length $q^*T^*$ that the head spends in $G$ in the absence of *Noise*$^*$. Because of the absence of *Noise*$^*$, at most one burst can happen during it. If it does then we will consider the larger part $J$ of the time interval before or after the burst (or the whole interval if there is none).
Let us apply Lemma 6.9 as well as its notation to the current situation, with $t_0$ our starting time. The interval $G$ is covered by

$$n = \gamma Q/\gamma' = 2Q$$

blocks of size $\gamma'$. Assume that our noise-free path $P$ starts at some time $t_0$ at a point $b_0$. Then there is a sequence of times $t_0 < t_1 < \ldots$ with $t_{i+1} - t_i \leq qT$, such that during the skips $(t_i, t_{i+1}]$ the head passes over some block $H_i$ either leftwards or rightwards, further except for $2n\pi$ skips, the interval $H_i' = \mathrm{Int}(H_i, (c_{\mathrm{marg}} + c_{\mathrm{spill}})B)$ is clean.

1. The number of skips during which the head touches disorder is at most $n(\pi + 4(c_{\mathrm{marg}} + c_{\mathrm{spill}}))$, and this is also an upper bound on the number of (short or long) stays in $K(t)$.

*Proof.* We already estimated the number of skips $i$ for which $H'_i$ is not clean. The remaining skips pass over a clean $H'_i$, but may touch disorder before or after it. If they do this then they will have to either enter a clean $H'_i$ from disorder, or leave it. By the Attack Cleaning property, each leaving skip increases by $\geq B$ the clean interval it leaves. it follows that disorder in $H$ will be eliminated after $2(c_{\text{marg}} + c_{\text{spill}})$ leaving skips over some block $H$ (ignoring integer parts). The entering skips will have to be balanced by leaving skips, so the total number of skips touching disorder in $H$ is $\leq 4(c_{\text{marg}} + c_{\text{spill}})$.

Since each (long or short) stay ends with a skip that touches disorder, this is also the bound on the number of stays.

2. Let us add up all the estimates, using the notation $\pi' = \pi + 4(c_{\text{marg}} + c_{\text{spill}})$.
Part 1 shows that the number of skips that are not clean is at most $n\pi'$, with $n = 2Q$, for a total time of $2QqT\pi'$.

Corollary 7.8 bounds the number of long stays by $3\gamma$, and Lemma 7.4 bounds the length of each long stay except for the time spent on interrupted rebuildings by $2\gamma UT$, so this gives a total time at most $6\gamma^2 UT$.

Corollary 7.3 bounds the number of restart events by $4\gamma Q/Z$. Each rebuilding interrupted by restart has at most $12\gamma QZ$ steps for the first sweeps over the rebuilding area, so $48\gamma^2 Q^2 T$ bounds the total time spent on rebuilding procedures interrupted by restarts.

Part 1 bounds then number of stays, so the total time spent in short stays is at most $2nUT\pi' = 4QUT\pi'$.

This gives the bound on the total time as $T$ multiplied with

$$2Qq\pi' + 6\gamma^2 U + 48\gamma^2 Q^2 + 4QU\pi'.$$

For $U$ as in Definition 2.14, the last term dominates the previous ones, so for large $Q$ this will be bounded by $c_{\text{esc}} QU\pi$ for an appropriate constant $c_{\text{esc}}$. As by definition $q^* = c_{\text{esc}} Q\pi$, this completes the proof of the lemma.

□

## 7.2 Weak attack cleaning

This section will scale up the Attack Cleaning property of trajectories (Definition 2.13) to machine $M^*$, but first only in a weaker version, restricting the number of bursts in the relevant interval.

The Attack Cleaning property says the following for the present case. Let $P$ be a path that is free of *Noise*$^*$. For current colony-pair $(x, x')$ (where $x' < x + 2QB$),

suppose that the interval $I = \left[x - (c_{\text{spill}} + 1)QB, x' + QB\right)$ is clean for $M^*$. Suppose further that $t$ is at the end of a work period in which the transition function, applied to $\eta^*(x,t)$, directs the head right. Then by the time the head returns to $x' - c_{\text{spill}}QB$, the right end of the interval clean in $M^*$ containing $x$ advances to the right by at least $QB$.

We will use the constant

$$E = 16\Delta B \tag{7.1}$$

which bounds the size of the whole range in which a call to healing operates. The constant $\Delta$ was defined in (5.2).

**Definition 7.10** Recall $\pi^* = \pi + 5$ from Definition 2.14. A path $P$ is called *tame* over the interval $I$ if during every time interval that it spends in $I$ has at most one burst, with at most

$$s = 2\pi^*(\pi^* + 2^{\gamma/5 + c_{\text{marg}} + 2}) \tag{7.2}$$

bursts altogether (this is less than $3\pi^2$ for large $\pi$). ⌟

**Lemma 7.11** (Weak attack cleaning) a) *In addition of the above condition of attack cleaning, assume that the trajectory $(\eta, Noise)$ is tame over the interval $I$. Then the conclusion holds. The analogous statement is also true when switching left and right.*

b) *Assume that the annotation game has been played to the beginning of the attack. Then the Range Extender player can extend the range in $I$ to the end of the attack, satisfying the conditions of the game.*

*Proof.* When the transfer phase of the simulation on the colony-pair $(x, x')$ begins, it may enter disorder to the right of $x' + QB$.

1. Assume an attack to the right. In the transfer process of the simulation, or if a rebuilding process is triggered later, the frontier zone is moving right. When head enters and later exits the disorder then it may create some new inconsistency. Moreover, every time the head exits disorder, since this may happen after a long-time absence, a burst may occur. Since the path is tame, the total number of bursts is limited to $s \ll Z$. The length of the frontier zone at the end of $J$ is $Z/2$, so, with $E$ defined in (7.1), there are at least $Z/2 - 2sE$ cells of the frontier zone that are at a distance at least $E$ from all bursts. No healing will change the sweep or the *BigDigression* field in any of these cells. So they can be overwritten only in normal or rebuilding mode. If rebuilding is started then it will move right. The only way that the head can move much left if the frontier zone itself turns back. This will only happen either at the end of transfer or at the end of a first sweep of rebuilding.

In the first case the simulation creates a new colony. Its *Info* track may be unusable, being damaged by many bursts, but during the rest of the simulation the head does not exit the colony pair, (the transfer sweep went out to the end of the turn region), so at most one new burst occurs, and the `ComplianceCheck` part of the simulation forces a compliant codeword by the end of the work period.

In the rebuilding case, the content of the new colonies is created from scratch anyway.

2. Assume that the attack is to the left. Then for the same reason as above, new bursts cannot turn back the leftward moving front. It can be replaced repeatedly by a rebuilding front started on its left, which may also be overridden similarly. But the only way to arrive to $x' + c_{\text{spill}}QB$ is to finish a started rebuilding, creating a new colony on the left, and also to erase the rebuilding marks in every cell marked for rebuilding in this process, making the whole passed-over area healthy.
3. In order to satisfy (b), let the Range Extender remove from $R$ at the beginning the whole area in which rebuilding happened. Then at the end, add back the whole area including the newly created colony; however, if a burst would occur in the last sweep then do this before the last sweep. This way, the condition will be satisfied that the addition to the range is clean. As the added area has been passed over several times in normal mode, the other conditions of cleanness and turn safety are also satisfied.

□

The following lemma draws a consequence of repeated applications of weak attack cleaning.

**Lemma 7.12** *Let $I_0$ be an interval of size $\geq (2c_{\text{spill}} + 1)QB$ and $J$ an adjacent interval of size $nQB$ on its right. Consider a path $P$ at whose beginning the interval $I_0$ is clean for $M^*$, and that is tame over $I_0 \cup J$. Assume that $P$ passes $I_0$ at least $2^{n+1}$ times from left to right and back. Then at some time during $P$, the whole interval $J$ becomes clean for $M^*$. The analogous statement holds if we switch left and right.*

*The statement analogous to part* (b) *of Lemma 7.11 also holds.*

*Proof.* Let $I_j = \left[a_j, b_j\right)$ be the largest interval containing $I_0$ after $j$ pairs of (left-right, right-left) passes. We will concentrate on the growth of $b_j$, though a similar analysis can show a simultaneous decrease of $a_j$. We know from Lemma 7.11 that after every pair of passes over $I_0$ (not necessarily on the larger interval $I_j$), the interval $I_0 = [a_0, b_0)$ will be clean for $M^*$. Also $b_j - b_0 \geq QB$, and $b_j$ is nondecreasing.

Suppose now that $b_j - b_0 \leq iQB$; we claim that then $b_{j+2^i} - b_j \geq QB$. Indeed, applying Lemma 4.13 to the machine $M^*$, after some $j' \leq 2^i$ left-right passes, the head

must make an attack from the rightmost colony of $I_{j+j'}$ allowing to apply Lemma 7.11, and as a result, increasing $b_{j'} \geq b_j$ by at least $QB$.

Repeating the argument, we get $b_{2^{n+1}} - b_0 \geq nQB$. □

## 7.3 Pass cleaning

The scaled-up version of the Pass Cleaning property considers a path $P$ with no $Noise^*$, as it makes $\pi^*$ pairs of passes over an the interval $I$, and claims that they make $\mathrm{Int}(I, c_{\mathrm{marg}}QB)$ clean for $\eta^*$. From now on, until further notice, consider a tame path $P$ over an interval $I$. Then $I$ will be made up of subintervals of size $\geq 4ZB$ that never gets bursts, separated from each other and the ends by distances $\leq 4sZB$. We will call these *basic holes*.

The pass cleaning property of $(\eta, Noise)$ cleans the basic holes (except for margins of size $\leq c_{\mathrm{marg}}B$) in the first $\pi$ pairs of passes. The Spill Bound property allows them to erode on the edges by the amount $c_{\mathrm{spill}}B$. We will call these somewhat smaller intervals still basic holes. One more pair of passes will make the basic holes, according to Lemma 6.2, safe for turns. The following two lemmas will show how some order will be established on them in a constant number of more passes. Recall that the maximum number of cells in a healing area, $E = O(\beta)$ in (7.1) is a constant, much smaller than the zigging distance (in cell widths) defined in (4.2).

**Definition 7.13** An interval will be called *almost clean* if it is clean except for a single island of size $\leq \beta' B$ where $\beta'$ was defined in (4.5). Let us call this island the *blemish*.

An interval $J$ of size $> 3ZB$ in the left direction from the head is called *right-directed* if

- it is almost clean;
- it is safe for turns;
- outside the blemish, its cells are all right-directed as seen by their sweep values (of simulation or rebuilding);
- its right end contains a frontier zone (of normal or rebuilding mode).
- this is the only frontier zone in $J$.

Left-directedness is defined similarly. ⌟

**Lemma 7.14** *Consider an almost clean interval $J = [a, b)$ of size $\geq 4ZB$ that is safe for turns. If a path with at most one burst passes it from left to right then it will leave a right-directed interval $J' = [a, b')$ with $b' \geq b - c_{\mathrm{spill}}B$. The same is true when interchanging left and right.*

*Proof.* If rebuilding never starts then every possible healing that is triggered succeeds just as in Lemma 6.3, extending the healthy area. The disorder in the blemish will be corrected as the head passes it, but a new burst may leave a new blemish behind (if it happens at the bottom of a zig). At the time of exit, $J'$ naturally becomes directed.

Suppose that rebuilding gets triggered. If it exits on the right then it leaves $J'$ directed. Here as well as in similar later situations, we use part (i1) of the definition of rebuilding, achieving that if a rebuilding is triggered within $(1/4)ZB$ on the right of a frontier zone of normal mode then it moves back to start immediately from this frontier zone. A blemish or burst or a restart (by some frontier on the left) may trigger healing, but eventually a rebuilding will either exit or succeed. If it succeeds then a normal mode starts with a direction to the right, and/or possibly new rebuilding with a center at least $QB$ to the right of the old one. Eventually the head will exit leaving $J'$ directed. □

Recall the definition of the feathering parameter $F$ in (4.2).

**Lemma 7.15** *Consider a tame path starting on the right of a right-directed interval $J = [a, b)$, and eventually crossing it to the left. Then we end up with left-directed interval $J' = [a', b')$, $a' \le a + c_{\rm spill}B$ and $b' \ge b$ having within $2Z$ cells of the right end a footprint of a big left turn (as defined in Section 4.4.2).*

*If $J$ already had such a footprint at position $x$ then $b' \ge x + FB$.*

*Proof.* The path can enter and exit $J$ repeatedly, and is allowed at most one new burst every time, with a bound on the total number of bursts given in Definition 7.10. Now the same reasoning applies to the front as in the proof of Lemma 7.11. If the head leaves $J$ on the right then we end up with an interval $J' \supseteq J$ that differs from a right-directed one only in possibly an area of size $\le 2EsB$ due to uncorrected islands caused by bursts, at most one in each entrance. The head can only leave on the left end if the frontier zone, either in normal or rebuilding mode, turns left on the right end of $J'$, leaving the footprint of a big left turn. In all inconsistencies trigger healing or rebuilding; but as the head leaves on the left, these all must succeed with the exception of one possible blemish caused by a burst that occurred during the last right-to-left pass; we end up with left-directed interval.

If rebuilding started then the frontier can move left only after it moved right by $> Q$ cells. In case of a footprint of a big left turn at the end of $J$, if rebuilding does not start then the feathering property will force the front to move by at least $F$ cells to the right before passing $J$ to the left. □

**Lemma 7.16** (Weak pass cleaning) a) *Suppose that a tame path $P$ makes $\pi^*$ pairs of passes over the interval $I$ starting from the left. Then before the end of the $\pi^*$th pair of passes the interior* $\mathrm{Int}(I, c_{\mathrm{marg}}QB)$ *becomes clean for $\eta^*$.*

b) *Assume that the annotation game has been played to the beginning of $P$. Then the Range Extender player can extend the range in $I$ to the end, satisfying the conditions of the game.*

*Proof.* We will number the right and left passes after the first $\pi$ pairs of passes as r1, l1, r2, l2, ....

1. r1, l1 make the basic holes safe for turns, except for margins of size $\leq (c_{\mathrm{marg}} + c_{\mathrm{spill}})B$.

   *Proof.* As shown above, the basic holes, of size $\geq 4ZB$ and separated from each other and the ends by distances $\leq 4sZB$, become and stay clean for $\eta$ in the first $\pi$ passes, except for margins of size $\leq (c_{\mathrm{marg}} + c_{\mathrm{spill}})B$. By Lemma 6.2, passes r1, l1 will make the basic holes safe for turns.

Let us call at any time a subinterval of $I$ an *essentially right-directed hole* if it can be turned into a right-directed one by changing it in $s$ islands of size $\leq EB$, and an *essentially foot-printed hole* if it has a footprint of a big left turn within $2ZB$ of its right end, again except for these islands.

2. Lemma 7.14 shows that pass r2 turns each basic hole $J$ into a right-directed hole, with possibly a small decrease on the left.

   Lemma 7.15 shows that pass l2 turns each of these holes into a foot-printed one, and that possible subsequent intrusions from the left will leave it essentially foot-printed.

   The first time the head passes right over any one of these footprints, this will be conserved. During all the later parts of the path, feathering requires that the head can pass left over it only by first shifting it to the right by at least $FB$. This will happen no later than during pass l3, thus by this time the right end of each hole will move by at least this much to the right of the right end of the basic hole it originates from. Since basic holes are separated by distances of $\leq 4sZB \ll F$, by the end of pass l3 all holes will overlap, leaving the whole interval $I$ left-directed.

   The intrusions between pass l3 and r4 may introduce some isolated islands and decrease $I$ by the non-isolated ones, but pass r4 make it right-directed again, and also safe for turns. Pass l4 makes it left-directed, and leaves it still safe for turns.

3. Pass r5 will clean $\mathrm{Int}(I, c_{\mathrm{Rebuild}}QB)$ for $\eta^*$.

   *Proof.* The intrusions between pass l4 and r5 may again introduce some isolated islands and decrease $I$. But during pass r5 all islands must be healed, except a single

blemish left behind due to a new burst. As long as healing succeeds then, just as in Lemma 6.3 after it the head returns to the front, and simulation continues. Suppose new rebuilding begins. This being a rightward pass, this does not happen so close to the left boundary that the left marking stage of the process would leave $I$. Hence it could have been triggered only on the right side of a healthy left segment $I'$ of $I$, and after completion, it extends this segment to the right. It may encounter more islands just ahead of the rightward marking front. Then as specified in part (i1) of the description of rebuilding, after any triggered healing, whether successful or not, the rebuilding process will just continue. Eventually, the only unfinished rebuilding process can be one that exits on the right before finishing, so its area is outside $\mathrm{Int}(I, c_{\mathrm{Rebuild}}QB)$.

4. Part (b) of the lemma can also be satisfied.
   The proof is similar to the corresponding part of Lemma 7.11. At the beginning, remove the whole interval $I$ from the range $R$. Then in the last pass $r5$, add back $[a, b) = \mathrm{Int}(I, c_{\mathrm{Rebuild}}QB)$ to the range, but do it possibly in two steps. It there is no burst in this pass that creates an uncorrected island then add it all back at the end. Suppose there is such a burst, then it must be at the left end of a zig that started at a position $x$ some $Z$ cells to the right of the burst. Say, the pass r5 is between times $t_1, t_3$, and the burst happens just after time $t_2$ where $t_1 < t_2 < t_3$. Then at time $t_2$ (when it is still healthy), add back interval $[a, x)$ to the range $R$, and at time $t_3$ add back the rest, $[x, b)$.

□

**Lemma 7.17** *Consider the statement of Lemma 7.12 with the interval $I_0$ having the same length $nQB$ with $n = \gamma/5$ as the interval $J$. The conclusion holds also for non-tame paths.*

*Proof.* Let $J_1 = I_0$, $J_0 = J$. We will show that if the conclusion does not hold then the path passes over all the infinite sequence of consecutive adjacent intervals $J_2, J_3, \ldots$ on the left of $J_1$, of size $|J_1|$. Since the path is finite, this leads to a contradiction. Let $i = 1$.

1. Suppose the conclusion of Lemma 7.12 does not hold. Then there are more than $s$ bursts over
   $$[a, b) = J_{i+1} \cup J_i$$
   during this time, consequently at least $\pi^* + 2^{n+2} < s/2^{n+2}$ bursts happened during some consecutive pair of the $2^{k+2}$ rightward passes over $J_{i+1}$.
2. By the Escape property, the path cannot stay long in an interval of size $\gamma QB >$ $3nQB$, so each burst is contained in a segment of the path covering an interval >

$3nQB$ with no bursts in it. Since these segments don't pass over $J_{i+1}$, each contains a fault-free pass over

$$I = \left[a - \gamma/5 - c_{\text{marg}}, b\right).$$

Suppose that $\text{Int}(I, c_{\text{marg}}QB) \supseteq J_{i+2}$ becomes clean at some time during the first $\pi^*$ of these passes. There are still $2^{n+2}$ fault-free passes over $J_{i+2}$, so we are back at the situation of part 1 with $i \leftarrow i+1$.

3. Suppose that $J_{i+2}$ does not become clean during the first $\pi^*$ passes. Then by Lemma 7.16, the number of bursts in $J_{i+2}$ exceeds $s$. Then at least $2(\pi^* + 2^{n+2})$ bursts happen over $J_{i+2}$ between some consecutive pair of the left-right passes over $J_{i+2}$. Using the Escape property similarly to the above, each burst belongs to a segment containing a fault-free pass over $J_{i+3}$. This brings us back to the situation of part 2 with $i \leftarrow i+1$.

□

Let us remove the bound on the number of bursts in the Pass Cleaning property of $\eta^*$

**Lemma 7.18** (Pass cleaning) *Let $P$ be a space-time path without Noise$^*$ that makes at least $\pi^*$ passes over an interval $I$. Then there is a time during $P$ when* $\text{Int}(I, c_{\text{marg}}QB)$ *becomes clean.*

*Proof.* We will prove the statement for $|I| = (\gamma/5)QB$. If $|I|$ is larger we can cover it by intervals of size $(\gamma/5)QB$ overlapping by $c_{\text{marg}}QB$: applying the statement simultaneously to each, it follows for $I$.

So assume $|I| = (\gamma/5)QB$ and let $J_1 = I$. We will show that if the conclusion does not hold then the path passes over all the infinite sequence of consecutive adjacent intervals $J_2, J_3, \ldots$ on the left of $J_1$, of size $|J_1|$. Since the path is finite, this leads to a contradiction. Let $i = 1$, $n = \gamma/5 + c_{\text{marg}}$.

1. By weak pass cleaning (Lemma 7.16), if $\text{Int}(J_i, c_{\text{marg}}QB)$ did not become clean for $\eta^*$, the number of bursts in $J_i$ is more than $s$ as in (7.2). Then there is a time interval between two consecutive left-right passes over $J_i$ with at least $2(\pi^* + 2^{n+2})$ bursts over $J_i$.

   Consider one of the bursts and an interval of size $\gamma QB$ containing it in the middle. Using the Escape property similarly to the proof of Lemma 7.17, we conclude that the head will escape it without other bursts. So the path contains a burst-free segment of size

$$(\gamma/2 - \gamma/5)QB > (\gamma/5 + c_{\text{marg}})QB$$

either on the left or on the right of $J_i = [a, b)$. Without loss of generality we can assume that at least half of them are on the left, giving $\pi^* + 2^{k+2}$ noise-free passes over $\left[a - (\gamma/5 - c_{\text{marg}})QB, b\right)$ during this time. Let $J_{i+1} = [a - (\gamma/5)QB, a)$.

2. If $J_{i+1}$ does not become clean during the first $\pi^*$ of these passes then restart the reasoning, going back to part 1, setting $i \leftarrow i + 1$. Otherwise by Lemma 7.17, interval $J_i$ becomes clean during the next $2^{n+2}$ noise-free passes over $J_{i+1}$, contrary to the assumption.

□

## 7.4 Attack cleaning and spill bound

Let us remove the bound on the number of bursts in the scale-up of the Attack Cleaning property.

**Lemma 7.19** (Attack cleaning) *Consider the situation of Lemma 7.11. The conclusion holds also if the path is not tame.*

*Proof.* Consider the path $P' \subseteq P$ containing the first $\pi^* + 2^{c_{\text{marg}}+4}$ bursts. The Escape property, used similarly to the proof of Lemma 7.17 implies that $P'$ passes the interval $J$ of length $(\gamma/2)QB$ on the right of $I$ this many times. The Pass Cleaning property then implies that $\text{Int}(J, c_{\text{marg}}QB)$ becomes clean for $\eta^*$ during the first $\pi^*$ passes of $P'$. Then Lemma 7.17 (applied in the left direction) implies that within the next $2^{c_{\text{marg}}+4}$ right-left passes, the disorder of $\eta^*$ of length $\leq (1 + c_{\text{marg}})QB$ between the old clean interval ending at $x' + QB$ and the new one beginning at $x' + (c_{\text{marg}} + 1)QB$ will be erased. □

Here is the scaled-up version of the spill bound property.

**Lemma 7.20** (Spill bound) *Suppose that an interval $I$ of size $> 2c_{\text{spill}}QB$ is clean for $\eta^*$, and let $P$ be a path with no faults of $\eta^*$. Then $\text{Int}(I, c_{\text{spill}}QB)$ stays clean for $\eta^*$.*

*Proof.* Without loss of generality, consider exits and entries of the path on the left of $I$. Let $C_0, C_1$ be the two leftmost colonies in $I$, where by definition $C_0$ is at the very end of $I$. The Spill Bound property of $(\eta, \mathit{Noise})$ allows a spill of size $c_{\text{spill}}B$ into $I$.

1. Assume first that the path is tame according to Definition 7.10. Let $J$ be the largest interval on the left end of $I$ such that every subinterval of size $EB$ contains a burst; then $|J| \leq sEB \ll ZB$, where $s$ is defined in (7.2). Let $I' = I \setminus J$.

   As long as no rebuilding is triggered the islands created by bursts in $I'$ do not affect admissibility. Indeed, without rebuilding the path just continues the simulation in $I'$. During every entrance of the path in $I$ at most one burst can happen within

$\gamma QB$ of the end. If the island left by a burst is not corrected during the present intrusion then it will be corrected at a next one if the path passes over it. If the next intrusion just deposits an island next to it without correcting then this has to be at a big leftward turn. Then due to feathering, if the head ever gets next time, it will pass over these islands by at least $FB$, and thus correct them.

2. Assume now that rebuilding is triggered: this can only happen in $J$. It may be interrupted by exit on the left or a burst. The interruption by a burst would be only temporary, since the heal rebuilding procedure of Section 5.5 deals with it.

   Still, a rebuilding may leave an island at its right end, due to a burst, and exit on the left end without finishing. Other rebuildings may leave other islands due to new bursts, so later rebuildings may encounter more, but total number of bursts is bounded by $s$, not enough to prevent the frontier zone of some rebuilding from eventually continuing and returning. As all rebuilding processes to be considered here must be triggered in $J$ or to the left of it, they can affect the health of an area of size at most $c_{\text{Rebuild}}QB$ on the left.

3. If the path is not tame, then we can finish just as in the proof of Lemma 7.19.

□

Part (b) of Lemma 7.11 and (b) of Lemma 7.16 hold, of course, also for the corresponding lemmas 7.19 and 7.18. One can conclude from them and the other lemmas of this section the following.

**Lemma 7.21** *Given a trajectory $(\eta, Noise)$ of machine $M$, the scaled-up history $(\eta^*, Noise^*)$ is a trajectory of machine $M^*$. Moreover, as the annotation game of $(\eta, Noise)$ is played, whenever the Attack Cleaning or Pass Cleaning property is applied to the scaled-up trajectory $(\eta^*, Noise^*)$, the annotation can be extended to the range cleaned up by these properties.*

# 8 Proof of the theorem

Above, we constructed a sequence of generalized Turing machines $M_1, M_2 \ldots$ with cell sizes $B_1, B_2, \ldots$ where $M_k$ simulates $M_{k+1}$. The sequences and dwell periods were also specified in Definition 2.14. Here, we will use this construction to prove Theorem 1.

## 8.1 Fault estimation

The theorem says that there is a Turing machine $M_1$ that can reliably (in the defined sense) simulate any other Turing machine $G$. Before the simulation starts, the input

$x$ of $G$ must be encoded by a code depending on its length $|x|$. We will choose a code that represents the input $x$ as the information content of a pair of cells of $M_r$ for an appropriate $r = r(x)$, and set their kind to Booting. The code does not depend on the length of the computation to be performed, only on the input length. At any stage of the computation there will be a highest level $K$ such that a generalized Turing machine $M_K$ will be simulated, with its cells of the Booting kind. We will denote the history of the computation by $(\eta^1, Noise^1) = (\eta, Noise)$, and its decodings by the recursion, as defined in Section 6.2 by $(\eta^k, Noise^k)$ where $(\eta^{k+1}, Noise^{k+1}) = ((\eta^k)^*, (Noise^k)^*)$.

Recall the values of $Q_k, U_k, B_k, T_k, S_k$ in Definition 2.14. Let $\mathcal{H}_k$ be the event that no burst of level $k$ occurs in the space-time region

$$W_k = \mathbf{B}((0,0), \gamma(B_{k+1}, S_{k+1})).$$

Lemma 2.6 bounds the probability of burst of level $k$ in any rectangle of type $\mathbf{B}(\mathbf{x}, (B_k, S_k))$ by $p_k = \varepsilon \cdot 2^{-1.5^{k-1}}$, giving

$$\mathsf{P}(\neg\mathcal{H}_k) = O(U_k Q_k p_k),$$

with $Q_k, U_k$ in Definition 2.14. This shows $\sum_k \mathsf{P}(\neg\mathcal{H}_k) = O(\varepsilon)$. From now on we assume that the event $\bigcap_k \mathcal{H}_k$ holds, since it holds with probability $1 - O(\varepsilon)$.

As the computation continues (and the probability of some fault occurring over the longer time increases), the encoding level will be raised again and again, by the lifting mechanism of Section 4.8. The configurations $\eta^k(\cdot, 0)$ are clean by definition for all levels $k$. Let $\sigma_k$ be the (random) time when lifting to level $k$ succeeded. By definition $\sigma_r = 0$. All trajectory properties are lifted by the lemmas in the preceding sections. Since $\mathcal{H}_{r+1}$ holds, the Transition Function property of trajectories applies to $\eta^r$ over the rectangle $W_{r+1}$. The booting and lifting steps of $\eta^r$ will leave the head within the space-time rectangle $W_{r+1}$, so $\sigma_{r+1} < S_{r+1}$. Also the lifted configuration $\eta^{r+1}(\cdot, \sigma_{r+1})$ is clean and healthy as no $r$-level noise disturbed its creation. By the same argument we get that the booting and lifting steps of $\eta^{r+1}$ will leave the head within $W_{r+2}$, with $\sigma_{r+2} < S_{r+2}$, the lifted configuration $\eta^{r+2}(\cdot, \sigma_{r+2})$ is clean and healthy. And so on, this holds for all $k$.

Suppose that the original simulated Turing machine $G$ produces output $y$ at its step $t$ (there is no halting, but the output in cell 0 will not change further). There will be a smallest level $s = s(t)$, depending only on the structure of the simulation, such that in our history $\eta$, for all $k > s$ there is a time $u$ between $\sigma_k$ and $\sigma_{k+1}$ with $\eta^k(0,u).Output = y$, that is the $k$th level simulation also outputs $y$. The times $\sigma_k$ are random, but we will compute below an upper bound $f(t)$ on $\sigma_{s(t)}$ that follows from the earlier assumptions. Take an arbitrary $t' > f(t)$. For each $k$, let $\mathcal{H}'_k(t')$ be the

event that no burst of level $k$ appears in

$$W'_k = \mathbf{B}((0,t'), \gamma(B_{k+1}, S_{k+1})).$$

Just as above for $\mathcal{H}_k$, we can assume that the event $\bigcap_k \mathcal{H}'_k$ holds, since it holds with probability $1 - O(\varepsilon)$. For each $k$ let $\sigma'_k$ be the (random) last time before $t'$ when the head of the simulated machine $M_k$ reaches position 0. Let $s'$ be the largest $k \geq s$ with $\sigma_k < \sigma'_k$.

Then $\mathcal{H}'_{s'}$ implies $\eta^{s'}(0, \sigma'_{s'}).Output = y$, that is the output of the simulated computation at time $\sigma'_{s'}$ on level $s'$ is $y$. Now we will use Lemma 7.21, saying that the areas known to be clean can also be annotated. Then by part (b) of Lemma 6.6 and by part 6 (the trickle-down) of the simulation procedure as described in Section 4.5, the absence of faults of level $s'-1$ while this procedure operates, as implied by condition $\mathcal{H}_{s'-1}$, implies $\eta^{s'-1}(0, \sigma'_{s'-1}).Output = y$. Repeating the argument for all $k < s'$ we find $\eta^k(0, \sigma'_k).Output = y$, so finally $\eta(0,t').Output = y$, with probability $1 - O(\varepsilon)$.

## 8.2 Space- and time-redundancy

Even with the simple tripling error-correcting code, there is a constant $\lambda > 1$ such that a colony of level $k$ uses at most $\lambda$ times more space than the amount of information contained in the cell of level $k+1$ that it simulates. Therefore if $k$ is the level that needs to be simulated before an output of $G$ can be reached then the space used at that time is at most $\lambda^k$ times the space needed to just store the information. If $G$ produces output at time $t$ then its space need is bounded by $t$, so the space need of the reliable simulation is at most $\lambda^k t$. Suppose this is within a pair of $k$-level cells just created by booting. The size of cells of level $k$ is, according to Definition 2.14,

$$Q_1 Q_2 \cdots Q_{k-1} = c_Q^k 2^{1+1.2+\cdots+1.2^{k-1}} = c_Q^k 2^{5 \cdot 1.2^k},$$

so they can simulate $t$ steps of $G$ if $\lambda^k t = c_Q^k 2^{5 \cdot 1.2^k}$. So $k$ is about $d \log\log t$ with $d \approx 1/\log 1.2$. This gives a bound

$$\lambda^k \approx \lambda^{d \log\log t} = (\log t)^\alpha$$

on the space redundancy factor, for some $\alpha > 0$.

The time redundancy can be estimated using the conclusions of Section 4.6.4. It shows that the simulation on a given level of the Turing machine $G$ incurs a redundancy that is a multiplier

$$O(FZ^2) = O(\pi^{8+4\rho}) = O(\pi^9)$$

if $\rho$ is small. Recall $\pi = 5k + O(1)$, Multiplying these on all levels we get, for some $\mu$, that the time redundancy on level $k$ is

$$\mu^k (k!)^9 = 2^{9k \log k + O(k)}.$$

Again, if $k = d \log\log t$ then this is less than

$$(\log t)^{10 \log\log\log t}.$$

**Remark 8.1** There is a mechanism more economical on storage, used in [6], with narrow *Work* and *Hold*$[j]$ tracks but with some added time complexity. This allows a space redundancy factor $1 + \delta_k$ with $\prod_k (1 + \delta_k) < \infty$, yielding a constant space redundancy factor for the whole hierarchy. ⌟

# 9 Discussion

**A weaker but much simpler solution** If our Turing machine could just simulate a 1-dimensional fault-tolerant cellular automaton, it would become fault-tolerant, though compared to a fault-free Turing machine computation of length $t$, the slow-down could be quadratic. (Such a solution would be only *relatively* simpler, being a reduction to a complex, existing one.) We did not find an easy reduction by just having the simulating Turing machine sweep larger and larger areas of the tape, due to the possibility of the head being trapped too long in some large disorder created by the group of faults. Trapping can be avoided, however, *provided that the length $t$ of the computation is known in advance*. The cellular automaton $C$ can have length $t$ , and we could define a "kind of" Turing machine $T$ with a *circular tape* of size $t$ simulating $C$. The transition function of $T$ would move the head to the right in every step (with any backward movement just due to faults).

**Decreasing the space redundancy** We don't know how to reduce the time redundancy significantly, but the space redundancy can be apparently reduced to a multiplicative constant. Following Example 4.3, it is possible to choose an error-correcting code with redundancy that is only a factor $\delta_k$ with $\prod_{k=1}^{\infty}(1 - \delta_k) > 1/2$. This also requires a more elaborate organization of the computation phase described in Section 4.5 since the total width of all other tracks must be only some $\delta_k$ times the width of the *Info* track. For cellular automata, such a mechanism was described in [6].

**Other models** There is probably a number of models worth exploring with more parallelism than Turing machines, but less than cellular automata: for example having

some kind of restriction on the number of active units. On the other hand, a one-tape Turing machine seems to be the simplest computation model for which a reasonable reliability question can be posed, in the framework of transient, non-conspiring faults of constant-bounded probability.

How about a two-dimensional tape? It is not clear such a machine can deal with noise if it is trying to simulate—efficiently—an arbitrary fault-free machine having a 2D tape. At least, all our techniques (attack cleaning, pass cleaning) exploit the one-dimensional nature of the tape in an essential way. ("Efficiently" is important here, as even with a 2D tape, the machine could fall back to using its tape in a 1D way, giving up any advantages of 2D.)

A simpler, universal computation model is the so-called *counter machine*. This has some constant number of nonnegative integer counters (at least two for universality), and an internal state. Each transition can change each counter by $\pm 1$, depends on both the internal state and on the set of those counters with zero value. A fault can change the state and can change the value of any counter by $\pm 1$. It does not seem possible to perform reliable computation on such a machine in any reasonable sense. The statement of such a result cannot be too simple-minded, since there is *some* nontrivial task that such a machine can do: with $2n$ counters, it can remember almost $2n$ bits of information with large probability forever. Indeed, let us start the machine with $n$ counters having the value 0, and the other $n$ having some large value (depending on the fault probability $\varepsilon$). The machine will remember forever (with large probability) which set of counters was 0. It works as follows (in the absence of a fault): at any one time, if exactly $n$ values have value 0, then increase each nonzero counter by 1. Otherwise decrease each nonzero counter by 1.

This sort of computation seems close to the limit of what counter machines can do reliably, but how to express and prove this?

## 10 References


[1] Eugene Asarin and Pieter Collins. Noisy turing machines. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming*, pages 1031–1042, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. 1

[2] Charles H. Bennett. The thermodynamics of computation – a review. *Intern. J. of Theor. Physics*, 21:905–940, 1981. 1

[3] Ilir Çapuni and Peter Gács. A Turing machine resisting isolated bursts of faults. *Chicago Journal of Theoretical Computer Science*, 2013. See also in arXiv:1203.1335. Extended abstract appeared in SOFSEM 2012. 2.1

[4] Bruno Durand, Andrei E. Romashchenko, and Alexander Kh. Shen. Fixed-point tile sets and their applications. *Journal of Computer and System Sciences*, 78:731–764, 2012. 1

[5] Peter Gács. Reliable computation with cellular automata. *Journal of Computer System Science*, 32(1):15–78, February 1986. Conference version at STOC' 83. 1, 1

[6] Peter Gács. Reliable cellular automata with self-organization. *Journal of Statistical Physics*, 103(1/2):45–267, April 2001. See also arXiv:math/0003117 [math.PR] and the proceedings of STOC '97. 1, 4.3, 4.6, 8.1, 9

[7] Peter Gács and John Reif. A simple three-dimensional real-time reliable cellular array. *Journal of Computer and System Sciences*, 36(2):125–147, April 1988. Short version in STOC '85. 1

[8] G. L. Kurdyumov. An example of a nonergodic homogenous one-dimensional random medium with positive transition probabilities. *Soviet Mathematics Doklady*, 19(1):211–214, 1978. 1

[9] Lulu Qian, David Soloveichik, and Erik Winfree. Efficient turing-universal computation with dna polymers. In Yasubumi Sakakibara and Yongli Mi, editors, *DNA Computing and Molecular Programming*, pages 123–140, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. 1

[10] Andrei L. Toom. Stable and attractive trajectories in multicomponent systems. In R. L. Dobrushin, editor, *Multicomponent Systems*, volume 6 of *Advances in Probability*, pages 549–575. Dekker, New York, 1980. Translation from Russian. 1

[11] John von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In C. Shannon and McCarthy, editors, *Automata Studies*. Princeton University Press, Princeton, NJ., 1956. 1

# 11 Appendix

The examples below serve to motivate some complexities of the construction.

**Example 11.1** (Need for feathering) Some big noise can create a number of intervals $I_1, I_2, \ldots, I_n$ consisting of colonies of machine $M_1$, each interval with its own simulated

head, where the neighboring intervals are in no relation to each other. When the head is about to return from the end of $I_k$ (never even to zig beyond it), a burst can carry it over to $I_{k+1}$ where the situation may be symmetric: it will continue the simulation that $I_{k+1}$ is performing. (The rightmost colony of $I_k$ and the leftmost colony of $I_{k+1}$ need not be complete: what matters is only that the simulation in $I_k$ would not bring the head beyond its right end, and the simulation in $I_{k+1}$ would not bring the head beyond its left end.)

The head can be similarly captured to $I_{k+2}$, then much later back from $I_{k+1}$ to $I_k$, and so on. This way the restoration of structure in $M_2$ may be delayed too long. ⌟

**Example 11.2** (Two slides over disorder) This example shows the possibility for the head to slide twice over disorder without cleaning it.

Consider two levels of simulation as outlined in Section 2.2: machine $M_1$ simulates $M_2$ which simulates $M_3$. The tape of $M_1$ is subdivided into colonies of size $Q_1$. A burst on level 1 has size $O(1)$, while a burst on level 2 has size $O(Q_1)$.

Suppose that $M_1$ is performing a simulation in colony $C_0$. An earlier higher-level burst may have created a large interval $D$ of disorder on the right of $C_0$, even reaching into $C_0$. For the moment, let $C_0$ be called a *victim* colony. Assume that the left edge of $D$ represents the last stage of a transfer operation to the right neighbor colony $C_0+Q_1$. When the head, while performing its work in $C_0$, moves close to its right end, a lower-level burst may carry it over into $D$. There it will be "captured", and continue the (unintended) right transfer operation. This can carry the head, over several successful colony simulations in $D$, to some victim colony $C_1$ on the right from which it will be captured to the right similarly. This can continue over new and new victim colonies $C_i$ (with enough space between them to allow for new faults to occur), all the way inside the disorder $D$. So the $M_2$ cells in $D$ will fail to simulate $M_3$.

After a while the head may return to the left in $D$ (performing the simulations in its colonies). When it gets at the right end of a victim colony $C_i$, a burst might move it back there. There is a case when $C_i$ now can just continue its simulation and then send the head further left: when before the head was captured on its right, it was in the last stage of simulating a left turn of the head of machine $M_2$.

In summary, a high-level burst can create a disordered area $D$ which can capture the head and on which the head can slide forward and back without recreating any level of organization beyond the second one. ⌟

The following example extends the above, showing the possibility of many levels of malicious (dis-)organization.

**Example 11.3** (Many slides over disorder) Let us describe a certain "organization" of a disordered area in which an unbounded number of passes may be required to

restore order. For some $n < 0$, let the cells of $M_1$ at positions $x_{-Q_1}, \ldots, x_n$, where $x_{i+1} = x_i + B_1$, represent part of a healthy colony $C(x_{-Q_1})$ starting at $x_{-Q_1}$, where $x_n$ is the rightmost cell of $C(x_{-Q_1})$ to which the head would come in the last sweep before the simulation will move to the *left* neighbor colony $C(x_{-2Q_1})$. Let them be followed by cells $x_{n+1}, \ldots, x_{Q_1-1}, \ldots$ which represent the last sweep of a transfer operation to the *right* neighbor colony $C(x_0)$. If the head is in cell $x_n$, a burst can transfer it to $x_{n+1}$. The cell state of $M_2$ simulated by $C(x_{-Q_1})$ need to be in *no relation* to the cell state of $M_2$ simulated by $C(x_0)$. This was a capture of the head by a burst of $M_1$ across the point 0, to the right.

We can repeat the capture scenario, say around points $iQ_1Q_2$ for $i = 1, 2, \ldots$, and this way cells of $M_3$ simulated by $M_2$ (simulated by $M_1$) can be defined arbitrarily, with no consistency needed between any two neighbors. (We did not write $iQ_1$ just in case bursts are not allowed in neighboring colonies.) In particular, we can define them to implement a *leftward* capture scenario via level 3 bursts at points $iQ_1Q_2Q_3Q_4$, allowing to simulate arbitrary cells of $M_5$ with no consistency requirement between neighbors. So $M_5$ could again implement a rightward capture scenario, and so on. In summary, a malicious arrangement of disorder and noise allows $k$ passes after which the level of organization is still limited to level $2k + 1$. ⌟

**Example 11.4** (Three islands) Suppose that the head has arrived at some colony-pair $C_0, C_1$ from the left, goes through a work period and then passes to the right. In this case, if no new noise occurs then we expect that all islands found in $C_0, C_1$ will be eliminated by the healing procedure. A new island $I_1$ can be deposited in the last sweep.

Consider the next time (possibly much later), when the head arrives (from the right). If it later continues to the left, then the situation is similar to the above. Island $I_1$ will be eliminated, but a new one may be deposited. But what if the head arrived to the colony-pair $C_1, C_2$ and turns back right at the end of the work period? If $I_1$ is not near the right end of $C_0$, then the head may never reach it to eliminate it; moreover, by the feathering way of making turns, it may add a new island $I_2$ near on the right end of $C_0$.

When the head returns a third time (possibly much later), from the right, feathering on the level of the simulated machine will cause it to leave on the left. Islands $I_1, I_2$ will be eliminated but a new island $I_3$ may be created by a new burst before, after or during the elimination. So the healing procedure must count with possibly three islands possibly in close vicinity to each other. But at least one of these, namely $I_2$, is near the end of $C_0$, not in the extended interior $\mathrm{Int}(C_0, PadLen - FB)$. ⌟

**Example 11.5** (No healing in rebuilding) This example shows the need for some heal-

ing of the rebuilding process itself. In it, restarting a rebuilding process on the occasion of every alarm prevents the scale-up of the Spill Bound property from $\eta$ to $\eta^*$. This property supposes that an interval $I = [a, b)$ of size $> 2c_{\text{spill}}QB$ is clean for $\eta^*$ and considers a path $P$ be a path that has no faults of $\eta^*$. It concludes that $\text{Int}(I, c_{\text{spill}}QB)$ stays clean for $\eta^*$. We can exploit the fact that $P$ has no faults of $\eta^*$ only by the implication that during every time interval that the path spends in $I$, it can have at most one burst. Let $l = c_{\text{Rebuild}}QB$.

Suppose that the path enters $I$ on the right during a rebuilding process that (seems) started just on the outside of $I$. The process marks the interval $[a_1, b)$ were $b - a_1 \approx l$. (We don't see what it does outside $I$, on the right of $b$). Somewhere near $a_1$, a burst causes alarm, restarting a rebuilding process which marks the interval $[a_2, b)$ where $a_1 - a_2 \approx l$, but the head leaves on the right of $I$ before rebuilding finishes.

Later, the head returns to continue the rebuild process, but a burst at position $a_1' = a_1 + l/2$ causes alarm and triggers a new rebuilding process. This finishes, making the interval $\left[a_2', b\right)$ healthy, where $a_2' \approx a_2 + l/2$. The interval $\left[a_2, a_2'\right)$, of size $\approx l/2$, is still marked for rebuilding.

Now an iterative process starts, creating marked intervals $\left[a_i, a_i'\right)$, of size $\approx l/2$, where $a_i' \approx a_{i-1}$. This way the disorder of $\eta^*$ in $I$ will not be confined to a subinterval at the right end of $I$ as the Spill Bound requires.

Suppose that we have a marked interval $\left[a_i, a_i'\right)$ of size $\approx l/2$ such that $\left[a_i', b\right)$ is healthy. The head enters in normal mode, continuing a simulation until it reaches $a_i'$. Then the marked cells it encounters trigger new rebuilding, which marks an interval $[a_{i+1}, b')$ where $a_{i+1} \approx a_i - l/2$, $b' \approx a_{i+1} + 2l$. The new rebuilding process is interrupted by a burst at $a_i' + l/2$, starting a new rebuilding. This rebuilding finishes, leaving the interval $\left[a_{i+1}, a_{i+1}'\right)$ marked where $a_{i+1}' - a_i \approx l/2$, and the the interval $[a_{i+1}, b)$ healthy. ⌟