CHICAGO JOURNAL OF THEORETICAL COMPUTER SCIENCE 2013, Article 03, pages 1–39
http://cjtcs.cs.uchicago.edu/

A Turing Machine Resisting Isolated Bursts Of Faults

Peter Gács Ilir Çapuni

Received: April 7, 2012; revised: March 2, 2013; published: March 27, 2013.

Abstract: We consider computations of a Turing machine under noise that causes violations of the transition function. Given an upper bound β on the size of bursts of faults, we construct a Turing machine $M(\beta)$ subject to faults that can simulate any fault-free machine under the condition that the time between bursts is lowerbounded by $O(\beta^2)$.

1 Introduction

1.1 The problem

The most natural question of reliable computation, in every computation model and noise model, is whether given a certain level of noise, a machine of that model exists that can perform arbitrarily complex computations under noise of that level. This question has positive answers for circuits and cellular automata, but has not yet been studied for Turing machines. Here, we construct a Turing machine that—with a slowdown by a multiplicative constant—can simulate any other Turing machine even if the simulator is subjected to constant size bursts of faults separated by a certain minimum number of steps from each other.

The problem of constructing fault-proof machines from components that can fail was first considered by von Neumann in [12], who addressed the problem in the Boolean circuits model. New advances along this path were made in [9, 10]. The question has been considered in uniform models of computation as well. A simple rule for two-dimensional cellular automata that keeps one bit forever even though each cell can fail with some small probability was given in [11]. A 3-dimensional reliably computing cellular automaton using Toom's rule was constructed in [4]. Alas, all simple one-dimensional cellular automata

Key words and phrases: Turing machine, reliability, fault-tolerance

DOI: 10.4086/cjtcs.2013.003

appear to be "ergodic" (forgetting everything about their initial configuration in time independent of the size). The first, complex, nonergodic cellular automaton was constructed in [2], and improved upon in [3]. It supports a hierarchical organization, based on an idea given in [6]. Cells are organized in units that perform fault-tolerant simulation of another automaton (of the same kind). The latter simulates even more reliably a third automaton of a similar kind, and so on.

The question of reliable computation with Turing machines (where arbitrarily large bursts may occur with correspondingly small probability) is raised in [3]. As in the case of one-dimensional cellular automata, no simple solution to this problem appears to exist. The present paper's machine is intended as a building block towards the eventual (hierarchical) construction of such a machine. This follows the paradigm of the proof in [2], where each member of the hierarchy of simulations is a similar building block, coping with distant bursts. To the best of our knowledge, this is the first construction of a sequential machine, reliable in a similar sense.

Turing machines may be a less natural model than cellular automata, given that the tape is not allowed to be directly affected by the faults. But it is quite meaningful to treat faults affecting the processing unit(s) separately from those affecting the memory. Our Turing machine model is the extreme case where the memory is simply not subjected to faults, and there is just one processing unit.

Models like this have been treated, in other contexts, seriously in the literature. See for example [5] (and papers it refers to), where leakage is restricted to the processing steps—the memory is assumed leakage-free.

The title of [1] suggests some connection, but that paper's interest is completely different: it examines the expressional ability, in terms of the arithmetical hierarchy, of Turing machines whose storage tapes are exposed to stochastic noise that tends to zero.

Many more works could be cited that deal with reliable computation—however, we do not think that they are relevant to the construction we are to offer here.

1.2 Simulating cellular automata

It is natural to try to derive fault-tolerant Turing machines from the existing results on fault-tolerant cellular automata. Cannot one simply define a Turing machine that simulates a fault-tolerant cellular automaton? In some sense, the answer is yes. Suppose that we know in advance the memory requirement $m = S(x)$ of a computation on a fault-tolerant cellular automaton M, on input x. The we can define a special Turing machine $T(m)$, working on a ***circular tape*** of size m, with the head moving always in the right direction (in other words, the "oblivious" property is hardwired), where each pass of T over the tape simulates one step of M. This machine will clearly have the same fault-tolerance properties that M has. Our efforts on fault-tolerant Turing machines can be seen as just aiming at the removal of the limitation of circular tape (input size-dependent hardware).

Thus, it would suffice to define fault-tolerant sweeping behavior on a regular tape (once the head can change direction, the sweeping movement can be disturbed by faults): the rest can be done by simulating a cellular automaton. We were, however, not able to do this without recreating the hierarchical constructions used in cellular automata—with all the necessary changes for Turing machines (and significant extra complications)

1.3 Turing machines

Our contribution uses one of the standard definitions of a Turing machine.

Definition 1.1. A Turing machine M is defined by a tuple

$$(\Gamma, \Sigma, \delta, q_{\text{start}}, F).$$

Here, Γ is a finite set of ***states***, Σ is a finite alphabet used in cells of the tape, and

$$\delta : \Sigma \times \Gamma \to \Sigma \times \Gamma \times \{-1, 0, +1\}$$

is a transition function. The tape alphabet Σ contains at least the distinguished symbols $\sqcup, 0, 1$ where $\sqcup$ is called the ***blank symbol***. The distinguished state q_{start} is called the ***starting state***. It is convenient to have not one halting state, but a set F of ***final states***: with the property that whenever M enters a state in F, it can only continue from there to another state in F, without changing the tape.

A ***configuration*** is a tuple

$$(q, h, x),$$

where $q \in \Gamma$, $h \in \mathbb{Z}$ and $x \in \Sigma^{\mathbb{Z}}$. Here, $x[p]$ is the content of the tape cell at position p. The tape is blank at all but finitely many positions. The work of the machine can be described as a sequence of configurations $C_0, C_1, C_2, \ldots$, where C_t is the configuration at time t. If $C = (q, h, x)$ is a configuration then we will write

$$C.state = q, \quad C.pos = h, \quad C.tape = x.$$

Here, x is also called the ***tape configuration***.

Though the tape alphabet may contain non-binary symbols, we will restrict input and output to binary.

Definition 1.2. For an arbitrary binary string x, let

$$M(x, t) \tag{1.1}$$

denote the configuration at time t, when started from a binary input string x written on the tape starting from position 0, with head position 0 and the starting state. Thus, the symbol at tape position p at time t can be written

$$M(x, t).tape[p].$$

The transition function δ tells us how to compute the next configuration from the present one. When the machine is in a state q, at tape position h, and observes tape cell with content a, then denoting

$$(a', q', j) = \delta(a, q),$$

it will change the state to q', change the tape cell content to a' and move to tape position to $h + j$. For $q \in F$ we have $a' = a$, $q' \in F$.

We say that a ***fault*** occurs at time t if the output (a', q', j) of the transition function at this time is replaced with some other value (which is then used to compute the next configuration). For the sake of a clean definition of simulations, we will be more formal in defining fault-free histories.

Definition 1.3 (Trajectory)**.** Let

$$\text{Configs}_M$$

denote the set of all possible configurations of a Turing machine M. Consider a sequence $\eta = (\eta(0), \eta(1), \dots)$ of configurations of $M = (\Gamma, \Sigma, \delta)$ with $\eta(t) = (q(t), h(t), x(t))$. This sequence will be called a ***history*** of M if the following conditions hold:

- $q(0) = q_{\text{start}}$.
- $x(t+1)[n] = x(t)[n]$ for all $n \neq h(t)$.
- $h(t+1) - h(t) \in \{-1, 0, 1\}$.

Let

$$\text{Histories}_M$$

denote the set of all possible histories of M. A history η with $\eta(t) = (q(t), h(t), x(t))$ of M is called a ***trajectory*** of M if for all t we have

$$(x(t+1)[h(t)], q(t+1), h(t+1) - h(t)) = \delta(x(t), q(t)). \tag{1.2}$$

We say that a history has a ***fault*** at time t if (1.2) is violated at time t. (Thus, if a history has any one fault, it is not a trajectory.) A ***burst of faults*** of a history is a sequence of times containing some faults.

With the earlier notation (1.1), if $x \in \Sigma^*$ is a string of nonblank tape symbols, then the history defined by

$$\eta(t) = M(x, t)$$

for all t is a trajectory in which $\eta(0)$ is a starting tape configuration obtained by surrounding x with blanks.

1.4 Codes and universal machines

To define simulation of a noise-free machine M_2 by a noisy machine M_1, we will specify a correspondence between configurations of these two machines. After a burst, the state of machine M_1—as well as the state of cells where the head was during the burst, could have been changed in an arbitrary way. The simulating machine must recover the information lost. Redundant storage will help. Section 3, we will show how one step of M_2 is simulated by a bounded number of steps of M_1.

We formalize redundant storage with the help of codes.

Definition 1.4 (Code)**.** Let Σ_1, Σ_2 be two finite alphabets. A ***block code*** is given by a positive integer Q, an ***encoding function*** $\varphi_* : \Sigma_2 \to \Sigma_1^Q$ and a ***decoding function*** $\varphi^* : \Sigma_1^Q \to \Sigma_2$ with the property $\varphi^*(\varphi_*(x)) = x$.

A ***tape configuration code*** is a pair of functions

$$\varphi_* : \Sigma_2^{\mathbb{Z}} \to \Sigma_1^{\mathbb{Z}}, \quad \varphi^* : \Sigma_1^{\mathbb{Z}} \to \Sigma_2^{\mathbb{Z}}$$

that encodes infinite strings of Σ_2 into infinite strings of Σ_1. Each block code (φ_*, φ^*) gives rise to a natural tape configuration code which we will also denote by (φ_*, φ^*). If $\xi = \cdots \xi(-1)\xi(0)\xi(1)\cdots$ is an infinite string of letters of Σ_2 then $\varphi_*(x)$ is the string

$$\cdots \varphi_*(\xi(-1))\varphi_*(\xi(0))\varphi_*(\xi(1))\cdots,$$

while for decoding an infinite tape configuration ξ' we subdivide it first into blocks of size Q (starting with $\xi'(0)\cdots\xi'(Q-1)$), decode each block separately, and concatenate the results.

Let us standardize the encoding of tuples of symbols and binary strings into binary strings (the particular choice here is not important):

Definition 1.5 (Standard pairing)**.** For a (possibly empty) binary string $x = x(1)\cdots x(n)$ let us introduce the map

$$x^o = x(1)x(1)x(2)x(2)\cdots x(n)x(n)01.$$

Now we encode pairs, triples, and so on, of binary strings as follows:

$$\langle s,t\rangle = s^o t,$$
$$\langle s,t,u\rangle = \langle\langle s,t\rangle,u\rangle,$$

and so on.

From now on, we will assume that our alphabets Σ, Γ are of the form $\Sigma = \{0,1\}^s$, $\Gamma = \{0,1\}^g$, that is our tape symbols and machine states are viewed as binary strings of a certain length. Also, if we write $\langle i,u\rangle$ where i is some number, it is understood that the number i is represented in a standard way by a binary string.

Definition 1.6 (Computation result, universal machine)**.** Assume that a Turing machine M starting on binary x, at some time t arrives at the first time at some final state. Then we look at the longest (possibly empty) binary string to be found starting at position 0 on the tape, and call it the ***computation result*** $M(x)$. We will write

$$M(x,y) = M(\langle x,y\rangle), \quad M(x,y,z) = M(\langle x,y,z\rangle),$$

and so on.

A Turing machine U is called ***universal*** among Turing machines with binary inputs and outputs, if for every Turing machine M, there is a binary string p_M such that for all x we have $U(p_M,x) = M(x)$. (This equality also means that the computation denoted on the left-hand side reaches a final state if and only if the computation on the right-hand side does.)

Let us introduce a special kind of universal Turing machines, to be used in expressing the transition functions of other Turing machines. These are just the Turing machines for which the so-called s_{mn} theorem of recursion theory holds with $s(x,y) = \langle x,y\rangle$.

Definition 1.7 (Flexible universal Turing machine)**.** A universal Turing machine will be called ***flexible*** if whenever p has the form $p = \langle p', p''\rangle$ then

$$U(p,x) = U(p', \langle p'', x\rangle).$$

(Even if x has the form $x = \langle x', x''\rangle$, this definition chooses $U(p', \langle p'', x\rangle)$ over $U(\langle p, x'\rangle, x'')$, that is starts with parsing the first argument.)

It is easy to see that there are flexible universal Turing machines. On input $\langle p,x\rangle$, a flexible machine first checks whether its "program" p has the form $p = \langle p', p''\rangle$. If yes, then it applies p' to the pair $\langle p'', x\rangle$. (Otherwise it just applies p to x.)

Definition 1.8 (Transition program)**.** Consider an arbitrary Turing machine M with state set Γ, alphabet Σ, and transition function δ. A binary string π will be called a ***transition program*** of M if whenever $\delta(a,q) = (a', q', j)$ we have

$$U(\pi, a, q) = \langle a', q', j\rangle.$$

We will also require that the computation induced by the program makes $O(|p| + |a| + |q|)$ left-right turns, over a length tape $O(|p| + |a| + |q|)$.

The transition program just provides a way to compute the (local) transition function of M by the universal machine, it does not organize the rest of the simulation.

Remark 1.9. In the construction of universal Turing machines provided by the textbooks (though not in the original one given by Turing), the program is generally a string encoding a table for the transition function δ of the simulated machine M. Other types of program are imaginable: some simple transition functions can have much simpler programs. However, our fixed machine is good enough (similarly to the optimal machine for Kolmogorov complexity). If some machine U' simulates M via a very simple program q, then

$$M(x) = U'(q,x) = U(p_{U'}, \langle q,x\rangle) = U(\langle p_{U'}, q\rangle, x),$$

so U simulates this computation via the program $\langle p_{U'}, q\rangle$.

1.5 The result

The main result could be stated as just saying that for every Turing machine M_2 and every burst constraint there is a Turing machine M_1 simulating M_1 under those burst constraints. But we want to bound the complexity of M_1 in terms of the complexity of M_2 in a somewhat subtle way that facilitates building a hierarchy of simulations later. Therefore we formulate a result in terms of universal Turing machines;

then it will be possible for M_1 to refer to a "program" of M_2, in a sense more general than just a transition table.

For simplicity, we will consider only computations whose result is a single symbol, at tape position 0:

$$M(x,t).tape[0]$$

at any time t in which $M(x,t).state$ is a final state. This frees us of the problem of having to decode before announcing the final result of the fault-tolerant computation. We will prove:

Theorem 1.10 (Main)**.** *For a given Turing machine M_2 with transition program p_2, and positive integer β, following items can be constructed, with $s_2 = |p_2| + \log|\Sigma_2| + \log|\Gamma_2|$:*

- *Integers $Q = O(s_2 + \beta)$, $V = O(s_2 Q)$;*
- *A block code (φ_*, φ^*) of blocksize Q giving rise to a corresponding tape configuration code;*
- *A machine M_1 whose number of states and alphabet size depend polynomially on Q, with some function f defined on its alphabet;*

such that the following holds.

Suppose that on input x, the fault-free machine M_2 enters a final state at time T. Assume that η is a history of machine M_1 on starting configuration $\varphi_*(x)$ such that bursts of faults have size at most β, and are separated by at most V steps from each other. Let t be any time $\geq VT$ such that no fault occurred in the last V steps before and including t, then

$$f(\eta(t).tape[0]) = M_2(x,T).tape[0]. \tag{1.3}$$

Section 2 specifies the layout of the tape and the structure of the states, and introduces the notion of rules. The parts of the transition function of M_1 dealing with redundant simulation are defined in Section 3. Section 4 introduces the parts allowing to restore the structure of the simulation after locally garbled by faults. The main theorem is proved in Section 5.

2 Program Structure

2.1 Overview

It is useful to view the task of error correction broken up into several problems to be solved. The solution of one problem gives rise to another problem one, but fortunately the process converges.

Redundant information The tape information of the simulated Turing machine will be stored in a redundant form, more precisely in the form of a block code.

Redundant processing The block code will be decoded, the retrieved information will be processed, and the result recorded. To carry out all this in a way that limits the propagation of faults, the tape will be split into tracks that can be handled separately, and the major processing steps will be carried out three times within one work period.

Local repair All the above process must be able to recover from a local burst of faults. For this, it is organized into a rigid, locally checkable structure with the help of local addresses, and some other tools like sweeps and zigging. The major tool of local correction, the local recovery procedure, turns out to be the most complex part of the construction.

Disturbed local repair A careful organization of the recovery procedure makes sure that even if a new burst interrupts it (or jumps into its middle), soon one or two new invocations of it will finish the job (whenever needed).

Here is some more detail.

Each tape cell of the simulated machine M_2 will be represented by a block of size Q of the simulating machine M_1 called a ***colony***. 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 makes a number of sweeps over the current colony, decodes the represented cell symbol and state, computes the new state, and transfers the necessary information to the neighbor colony that corresponds to the new position of the head of M_2.

In order to protect information from the propagation of errors, the tape of M_1 is subdivided into ***tracks***: each track corresponds to a ***field*** of a cell symbol of M_1 viewed as a data record. Each stage of computation will be repeated three times. The results will be stored in separate tracks, and a final cell-by-cell majority vote will recover the result of the work period from them.

All this organization is controlled by a few key fields, for example a field called *c.Addr* showing the position of each cell in the colony, and a field *c.Sw* showing the last sweep of the computation (along with its direction) that has been performed already. The technically most challenging part of the construction is the protection of this control information from bursts.

For example, a burst can reverse the head in the middle of a sweep. Our goal is that such structural disruptions be discovered locally, so we cannot allow the head to go far from the place where it was turned back. Therefore the head's movement will not be straight even during a single sweep: it will make frequent short switchbacks (zigzags). This will trigger alarm, and the start of a recovery procedure if for example a turn-back is detected.

It is a significant challenge that the recovery procedure itself can be interrupted (or started) by a burst.

2.2 Error-correcting codes

The redundant code used in the implementation needs some error-correcting property.

Definition 2.1 (Error-correcting code)**.** A block code is (β,t)-***burst-error-correcting***, if for all $x \in \Sigma_2$, $y \in \Sigma_1^Q$ we have $\varphi^*(y) = x$ whenever y differs from $\varphi_*(x)$ in at most t intervals of size $\leq \beta$.

Example 2.2 (Tripling)**.** Suppose that $Q \geq 3\beta$ is divisible by 3, $\Sigma_2 = \Sigma_1^{Q/3}$, $\varphi_*(x) = xxx$. Let $\varphi^*(y)$ be obtained as follows. If $y = y(1)\dots y(Q)$, then $x = \varphi^*(y)$ is defined as follows: $x(i) = \mathrm{maj}(y(i), y(i+Q/3), y+2Q/3)$. (We treat $y(i)$ as binary strings, so a bitwise majority always exists.) For all $\beta \leq Q/3$, this is a $(\beta,1)$-burst-error-correcting code.

If we repeat 5 times instead of 3, we get a $(\beta,2)$-burst-error-correcting code (there are also much more efficient such codes than just repetition).

2.3 Fields

Each state of the simulating machine M_1 will be a tuple $q = (q_1, q_2, \ldots, q_k)$, where the individual elements of the tuple will be called ***fields***, and will have symbolic names. For example, we will have fields *Info* and *Drift*, and may write q_1 as $q.Info$ or just *Info*, q_2 as $q.Drift$ or *Drift*, and so on.

We will call the current direction of the simulated machine M_2 the ***drift*** (-1 for left, 0 for none, and $+1$ for right).

A properly formatted configuration of M_1 splits the tape into blocks of Q consecutive cells called ***colonies***. One colony of the tape of the simulating machine represents one cell of the simulated machine. The colony that corresponds to the active cell of the simulated machine (that is the cell that the simulated machine is scanning) is called the ***base colony*** (later we will give a precise definition of this notion based on the actual history of the work of M_1). Once the drift is known, the union of the base colony with the neighbor colony in the direction of the drift is called the ***extended base colony*** (more precisely, see Definition 4.4).

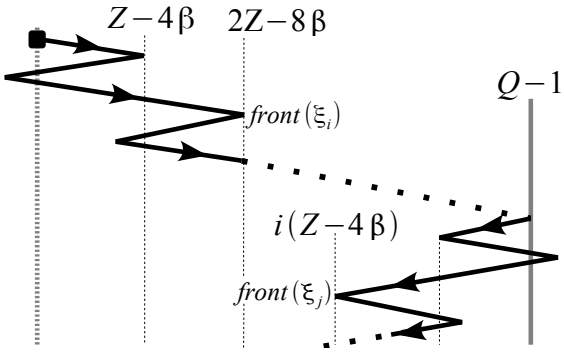


Figure 1: A sweep of a base colony with zigging.

The states of machine M_1 will have a field called

$$Mode \in \{\text{Normal}, \text{Recovering}\}.$$

It shows whether the machine is engaged in the regular business of simulation or in the repair of its own structure. The normal mode corresponds to the states where M_1 is performing the simulation of M_2. The recovery mode tries to correct some perceived fault.

Similarly, each cell of the tape of M_1 consists of several fields. Some of these have names identical to fields of the state. In describing the transition rule of M_1 we will write, for example, $q.Info$ simply as *Info*, and for the corresponding field $a.Info$ of the observed cell symbol a we will write $c.Info$. The array of values of the same field of the cells will be called a ***track***. Thus, we will talk about the $c.Hold$ track of the tape, corresponding to the $c.Hold$ field of cells.

The basic fields of the state and of cells are listed in Section 7.1 with some hints of their function (this does not replace our later definition of the transition function). Each field of a cell has also a possible value $\emptyset$ corresponding to the case when the state is blank.

Some fields and parameters are important enough to introduce them right away. The

$$c.Info, c.State$$

track of a colony of M_1 contain the strings that encode the content of the simulated cell of M_2 and its simulated state respectively.

The field

$$c.Addr$$

of the cell shows the position of the cell in its colony. There is a corresponding *Addr* field of the state. The

$$Sw$$

field counts the sweeps that the head makes during the work period. There is a corresponding *c.Sw* field in the cell. The direction $d \in \{-1,0,1\}$ in which the simulated head moves will be denoted by

$$Drift.$$

There is a corresponding field *c.Drift*. The number of the last sweep of the work period will depend on the drift d, and will be denoted by

$$\mathrm{Last}(d). \tag{2.1}$$

2.4 Rules

Instead of writing a single huge transition table, we present the transition function as a set of ***rules***. Each rule consists of some conditional statements, similar to the ones seen in an ordinary program: "**if** *condition* **then**...", where the condition is testing values of some fields of the state and the observed cell. Even though rules are written like procedures of a program, they describe a single transition. When several consecutive statements are given, then they (almost always) change different fields of the state or cell symbol, so they can be executed simultaneously. Otherwise and in general, even if a field is updated in some previous statement, in all following statements that use this field, its old value is considered.

Rules can call other rules, but these calls will never form a cycle. We will also use some conventions introduced by the C language: namely, $x \leftarrow x+1$ and $x \leftarrow x-1$ are abbreviated to $x{+}{+}$ and $x{-}{-}$ respectively.

Rules can also have parameters, like $\texttt{Swing}(a,b,u,v)$ (see below). Since each rule is called only a constant number of times in the whole program, the parametrized rule can be simply seen as a shorthand.

Most our rules will be described only informally; only a couple of examples will show how the formal version would look.

3 The Simulation

Machine M_1 never halts: even when the simulated computation ends the simulation continues to defend the end result from faults.

One computational step of the machine M_2 is simulated by many steps of M_1 that make one unit called the ***work period***. This is broken up into a ***computation phase***, when the transition function of M_2 is computed, and a ***transfer phase***, when the necessary information is transferred into the appropriate neighbor colony.

3.1 Coding

Let us give here some details of the encoding of the state and cell content of machine M_2 into a colony of M_1.

We will frequently make use of the following parameter:

Definition 3.1. Let $E = 30\beta$.

For simplicity, let us assume that the set of states Γ_2, and the alphabet Σ_2 are subsets of the set of binary strings $\{0,1\}^\ell$ for some $\ell < Q$ (we can always ignore some states or tape symbols, if we want). We will then use the same code υ for both the states of machine M_2 and its alphabet. Let (υ_*, υ^*) be a $(\beta,2)$-burst-error-correcting code

$$\upsilon_* : \{0,1\}^\ell \to \{0,1\}^{Q-2.2E}.$$

(The length of the codeword only $Q-2.2E$, in order to distance it by $1.1E$ from both colony ends.) We could use, for example, the repetition code of Example 2.2. Other codes are also appropriate, but we require that they have some fixed, constant programs p_{encode}, p_{decode} on the universal machine U, in the following sense:

$$\upsilon_*(x) = U(p_{\text{encode}}, x), \quad \upsilon^*(y) = U(p_{\text{decode}}, y).$$

Also, these programs must work in quadratic time and linear space on a one-tape Turing machine (as tripling certainly does, as 5-fold repetition, too, which is a $(\beta,2)$ error-correcting code).

Let a be the tape configuration of M_2 at time 0, and s the starting state of M_2. The initial tape configuration $a' = \varphi_*(a)$ of M_1 is defined as follows:

$$a'[h \cdot Q + 1.1E, \ldots, (h+1)Q - 1.1E - 1].\mathit{Info} = \upsilon_*(a[h]), \tag{3.1}$$

$$a'[1.1E, \ldots, Q - 1.1E - 1].\mathit{State} = \upsilon_*(s). \tag{3.2}$$

In cells of the base colony and its left neighbor colony, the *c.Sw* and *c.Drift* fields are set to Last$(+1)$ and 1 respectively. In the right neighbor colony, these values are Last(-1) and -1 respectively. In all other cells, these values are empty.

The head is initially located at the first cell of the base colony. We assume that the *Addr* fields of each colony are filled properly, that is

$$a'[i].\mathit{Addr} = i \bmod Q.$$

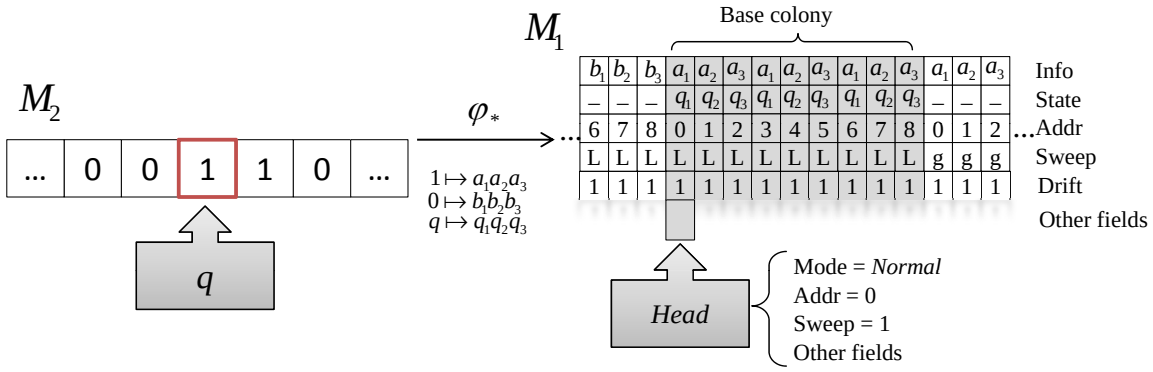


Figure 2: The initial configuration of machine M_2 is encoded into the initial configuration of M_1, where $L = \mathrm{Last}(1)$, $g = \mathrm{Last}(-1)$.

Machine M_1 starts in normal mode, $Drift = 1$, $Sw = 1$. All other fields have also their initial (or empty) values (see Figure 2).

The corresponding block decoding function φ^* is obtained by applying the decoding function υ^* to just the *c.Info* track of M_1 (actually to just the part between addresses $1.1E$ and $Q - 1.1E$ of each colony) to obtain the tape of the simulated machine M_2, and to just the *c.State* track of the base colony to obtain its state.

This definition of decoding will be refined for configurations different from the initial one, since the location of the base colony must also be found using decoding.

3.2 Sweep counter and direction

The global sweeping movement of the head will be controlled by the parametrized rule

$$\texttt{Swing}(a,b,u,v).$$

This rule makes the head swing between two extreme points a, b, while the counter Sw increases from value u to value v. The Sw value is incremented at the "turns" a, b (and is also recorded on the track *c.Sw*).

The sweep direction δ of the simulating head is derived from Sw, $Addr$ and the current value Dir in the following way. On arrival of the head to an endpoint (that is when $Dir \neq 0$ and $Addr \in \{a,b\}$), the values Sw and *c.Sw* are incremented and Dir is set to 0. In all other cases, the sweep direction is determined by the formula

$$\mathrm{dir}(s) = (-1)^{s+1}. \tag{3.3}$$

Let

$$\delta = \begin{cases} 0 & \text{if } Addr \in \{a,b\} \text{ and } Dir \neq 0, \\ \mathrm{dir}(Sw) & \text{otherwise.} \end{cases} \tag{3.4}$$

As we mentioned, each sweep will be broken up into zigzags to allow the detection of premature turn-back. At each non-zigging step, $Addr \leftarrow Addr + \delta$.

We will make frequent use of the following parameter:

Definition 3.2. Let $Z = 22\beta$.

As an example of rules, we present the details of the zigging rule in Rule 7.2, which itself uses the rule `Move`(d). A characteristic of this rule is that it changes nothing on the tape: all its control information is in the state. The rule repeats the following cycle: first it moves forward $Z - 4\beta$, moving ahead the ***front***, and performing all necessary operations of the computation. Then it moves backward and forward by Z steps, not changing anything on the tape (but as we will see, some consistencies will be checked). The field *ZigDepth* of the state measures the distance from the front during the switchback. From now on, we will assume that Q is a multiple of $Z - 4\beta$.

3.3 The computation phase

The aim of this phase is to obtain new values for *c.State*, *c.Drift* and *c.Info*. It essentially repeats three times the following ***stages***: decoding, applying the transition, encoding. In more detail:

1. For every $j = 1, 2, 3$ do

   a. Calling by g the string found on the *c.State* track of the base colony between addresses $1.1E$ and $Q - 1.1E$, decode it into string $\tilde{g} = \upsilon^*(g)$ (this should be the current state of the simulated machine M_2), and store it on some auxiliary track in the base colony. Do this by simulating the universal machine on some work track: $\tilde{g} = U(p_{\text{decode}}, g)$.

   Proceed similarly with the string a found on the *c.Info* track of the base colony, to get $\tilde{a} = \upsilon^*(a)$ (this should be the observed tape symbol of the simulated machine M_2).

   b. Compute the value

   $$(a', g', d) = \delta_2(\tilde{a}, \tilde{g})$$

   similarly, simulating the universal machine U with program p_2. The string p_2 (as well as the constant-size programs $p_{\text{decode}}, p_{\text{decode}}$) is "hardwired" into the transition function of M_1, that is the program we are writing. More precisely, before performing the computation of $U(p_2, \tilde{a}, \tilde{q})$ of Definition 1.7, machine M_1 writes the program p_2 onto some work track: for this, the string p_2 is a *literal* part of the program of M_1.

   c. Write the encoded new state $\upsilon_*(g')$ onto the *c.Hold*$[j]$*.State* track of the base colony between positions $1.1E$ and $Q - 1.1E$.

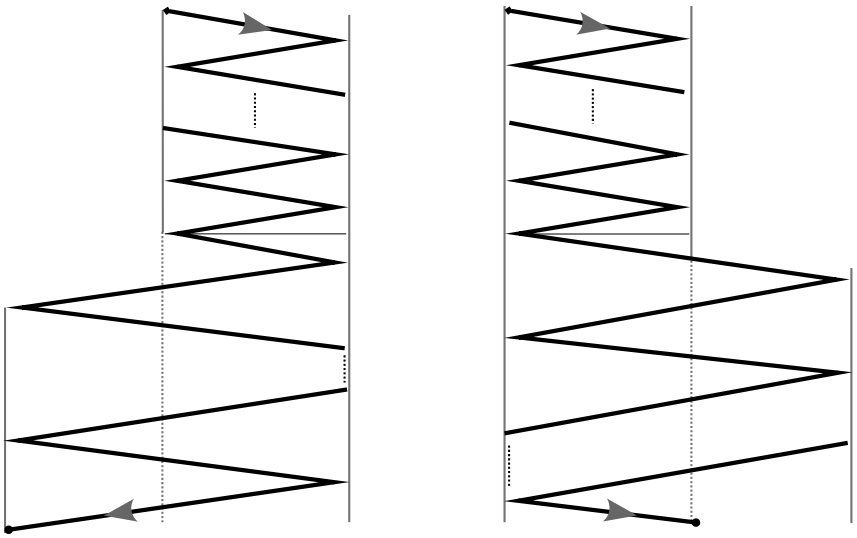
   Similarly, write the encoded new observed cell content $\upsilon_*(a')$ onto the *c.Hold*$[j]$*.Info* track of the base colony. Write also the first symbol of a' into position 0 of the same track (just because the Main Theorem 1.10 expects the result of the whole computation at tape position 0, not position $1.1E$).

   Write d into the *c.Hold*$[j]$*.Drift* field of *each cell* of the base colony.

2. Repeat the following twice:

   Sweeping through the base colony, at each cell compute the majority of $c.Hold[j].Info$, $j = 1,2,3$, and write into the field $c.Info$. Proceed similarly, and simultaneously, with *State* and *Drift*.

   It can be arranged—and we assume so—that the total number of sweeps of this phase, and thus the starting sweep number of the next phase, depends only on Q.



(a) (b)

Figure 3: A work period of the machine M_1 (zigging and many sweeps are not shown, for clarity). In (a) the machine drifts left, while in (b) it drifts right.

3.4 Transfer phase

The aim of this phase, present only if $Drift \neq 0$, is to transfer the new *State* of M_2 into the neighbor colony in the direction of $\delta = Drift$ (which was computed in the previous phase), and to move there. $\mathrm{TfSw}(\delta)$ is the ***transfer sweep***, the sweep in which we start transferring in direction δ. We have $\mathrm{TfSw}(-1) = \mathrm{TfSw}(1) + 1$.

The phase consists of the following actions.

1. Spread the value δ found in the cells of the $c.Drift$ track (they should all be the same) onto the neighbor colony in direction δ.

2. For $i = 1,2,3$:

   Copy the content of $c.State$ track of the base colony to the $c.Hold[i].State$ track of the neighbor colony.

3. Repeat the following twice:

   Assign the field majority: $c.State \leftarrow \mathrm{maj}(c.Hold[1\ldots3].State)$ in all cells of the neighbor colony.

4. If $Drift = 1$, then move right to cell Q (else stay where you are).

The work period in case of both non-zero drift values is illustrated in Figure 3.

3.5 Simulation with no faults

The proof of Theorem 1.10 uses a simulation of machine M_2 by machine M_1. Though the theorem only speaks about the end result, for the sake of the proof we give a formal definition of simulation. For the moment, we concentrate on the fault-free case.

Definition 3.3. Let M_1, M_2 be two machines, further let

$$\varphi_* : \mathrm{Configs}_{M_2} \to \mathrm{Configs}_{M_1}$$

be a mapping from configurations of M_2 to those of M_1, such that it maps starting configurations into starting configurations. We will call such a map a ***configuration encoding***. Let

$$\Phi^* : \mathrm{Histories}_{M_1} \to \mathrm{Histories}_{M_2}$$

be a mapping. The pair (φ_*, Φ^*) of mappings is called a ***simulation*** (of M_2 by M_1) if whenever ξ is an initial configuration of M_2 and η is a trajectory of machine M_1 with initial configuration $\varphi_*(\xi)$, the history $\Phi^*(\eta)$ is a trajectory of machine M_2.

We say that M_1 ***simulates*** M_2 if there is a simulation (φ_*, Φ^*) of M_2 by M_1.

We summarize the construction of the previous section in the following statement.

Lemma 3.4. *Machine M_1 simulates machine M_2.*

Proof. Since there are no faults interfering with the operation of M_1, the history of M_1 is a trajectory, easy to break up into ***work periods***: intervals in which the counter *Sw* is growing. Let τ_t be the end of the t-th work period. (Though τ_t is roughly proportional to t, we did not make it an exact multiple. Such a relation would be lost in the faulty case, anyway.)

The code (φ_*, φ^*) is given in Section 3.1. The history decoding function Φ^* for the noise-free case is

$$\Phi^*(\eta)(t) = \varphi^*(\eta(\tau_t)),$$

where φ^* is the tape configuration decoding function obtained from the block code (φ_*, φ^*).

If t reaches a final state of M_2, then starting from step τ_t, machine M_1 will not change the state represented on the *c.Info* track anymore. □

4 Faults

A fault is a violation of the transition function. A burst of faults can change the state to an arbitrary one, and change an interval of cells over which the head passes, arbitrarily. We will call such an interval an "island of bad cells" here informally, later formally.

Faults cause two kinds of change. One is that they change the information about the represented machine M_2. This problem will be corrected with the help of redundancy (encoding of the information and repetition of computation).

Remark 4.1. We could spell out a lemma similar to Lemma 3.4, in which faults are allowed, but are not allowed to change the triple $Core = (Addr, Sw, Drift)$ or the corresponding triple $c.Core$ on the tape. Our simulation program has enough redundancy that its proof would be straighforward: we leave the checking to the reader.

The second kind of change affects the very structure of the simulation (the triples *Core* and $c.Core$). These changes will be detected and corrected locally, by the recovery rule.

As mentioned earlier, a special field *Mode* of the state can have the values Normal and Recovering. All the simulation rules described above apply in the normal mode. When a ***coordination check*** (see later) fails, we will switch to recovering mode. The recovery rule is governed by several fields of the state and the cell. Recovery will start with trying to identify a small interval containing the damage. A particularly important field is $c.Rec.Addr$ which lays out temporary coordinates in the cells of this interval. This is followed by restoring the "structure" (addresses, $c.Sw$ and $c.Drift$ values) in the interval.

Definition 4.2 (Markedness)**.** A cell is called ***marked*** if $c.Rec.Addr \neq \emptyset$.

We will define formally later in Definition 4.7 what it means for the state to be coordinated with the observed cell. This is always the case in the noise-free simulation, so let us display the "main" rule of machine M_1 in Rule 4.1.

Rule 4.1: Main rule

if $Mode = \text{Normal}$ **then**
  if not Coordinated **or** the cell is marked for recovery **then** `Alarm`
  else if $1 \leq Sw < \text{TfSw}(1)$ **then** `Compute`
  else if $\text{TfSw}(1) \leq Sw < \text{Last}$ **then** `Transfer`
  else if $\text{Last} \leq Sw$ **then** move the head to the new base.

4.1 Integrity

Let us specify the kind of structural integrity we expect a configuration to have. We spell out the details to make it clear what is it that needs (and can) be locally checked.

Informally, in the absence of faults, "outer" cells are those outside the base colony, and even outside the area (to be called workspace) in which the program extends it in the transfer phase.

Definition 4.3 (Outer cells)**.** Recall the definition of the sweep value $\mathrm{Last}(\delta)$ from (2.1). For $\delta \in \{-1,1\}$, if a cell is nonempty and has $0 \le c.Addr < Q$, $c.Drift = \delta$, $c.Sw = \mathrm{Last}(\delta)$ then it will be called a ***right outer cell*** if $\delta = -1$. It is a ***left outer cell*** if $\delta = 1$. If it is empty then it will be considered both a left outer cell and a right outer cell.

Definition 4.4 (Healthy configuration, base colony, extended base colony, workspace)**.** A configuration ξ is ***healthy*** if the mode is normal, further the following holds.

Let d denote the direction of sweep, as determined by (3.4). Recall that $\xi.pos$ is the head position. We define the position $f = \mathrm{front}(\xi)$, called the ***front***, by

$$\mathrm{front}(\xi) = \xi.pos + ZigDepth \cdot d.$$

This is the farthest position to which the head has advanced before starting a new backward zig.

Let $\delta = Drift$. Recall the definition of the ***transfer sweep*** $\mathrm{TfSw}(\delta)$ in Section 3.4 if $\delta \neq 0$. There is no transfer sweep if $\delta = 0$. We require:

Colonies The non-blank cells of the tape form a single segment, subdivided into colonies, starting from the ***base*** defined by counting back from $Addr$ (this is not necessarily the origin of the tape). The leftmost colony and rightmost colony of the tape may be only partially filled.

The colony starting at the base is called the ***base colony***. The ***extended base colony*** X is obtained by extending the base colony in the direction δ, provided $Sw \ge \mathrm{TfSw}(1)$.

The front $\mathrm{front}(\xi)$ is always in the extended base colony. The drift of nonempty outer cells points towards the base colony.

Workspace The non-outer cells form a single interval called ***workspace***, with the following properties:

- For $Sw < \mathrm{TfSw}(\delta)$, it is equal to the base colony.
- In case of $Sw = \mathrm{TfSw}(\delta)$, it is the smallest interval including the base colony and the cell adjacent to $\mathrm{front}(\xi)$ on the side of the base colony.
- If $\mathrm{TfSw}(\delta) < Sw < \mathrm{Last}(\delta)$, then it is the extended base colony.
- When $Sw = \mathrm{Last}(\delta)$, it is the smallest interval including the future base colony and $\mathrm{front}(\xi)$ (it is shrinking onto the future base colony).

The field $c.Addr$ varies continuously over the workspace in all these cases, except possibly $Sw = 1$.

Sweep For $1 \le c.Sw \le \mathrm{Last}(\delta)$, we have $c.Sw(x) = Sw$ in all cells x behind $\mathrm{front}(\xi)$ in the workspace. For $1 < c.Sw$, we have $c.Sw(x) = Sw - 1$ in all cells x ahead of $\mathrm{front}(\xi)$ (inclusive) in the workspace.

Addresses Consider addresses $c.Addr$ in the workspace. Except for $Sw = 1$, they increase continuously.

In the first sweep, the address track $c.Addr$ is either $[-Q, 0)$ or $[Q, 2Q)$, but reduced modulo Q on the segment $[0, \mathrm{front}(\xi))$.

Drift If $Sw \ge \mathrm{TfSw}(1)$ or $Sw = 1$ then $c.Drift$ is constant on the workspace.

Simulated content The *Info* and *State* tracks contain valid codewords as defined in Section 3.1.

Normality All cells are unmarked, that is $c.Rec.Addr = \emptyset$ throughout.

The following observation comes directly from the definition of health.

Lemma 4.5. *In a healthy configuration, a cell is either under the head, or is in the workspace, or is an outer cell.*

Definition 4.6 (Local configuration, replacement)**.** A ***local configuration on*** a (finite or infinite) interval I is given by values assigned to the cells of I, along with the following information: whether the head is to the left of, to the right of or inside I, and if it is inside, on which cell, and what is the state.

If I' is a subinterval of I, then a local configuration ξ on I clearly gives rise to a local configuration $\xi(I')$ on I' as well, called its ***subconfiguration***: If the head of ξ was in I and it was for example to the left of I', then now $\xi(I')$ just says that it is to the left, without specifying position and state.

Let ξ be a configuration and $\zeta(I)$ a local configuration that contains the head if and only if $\xi(I)$ contains the head. Then the configuration $\xi|\zeta(I)$ is obtained by replacing ξ with ζ over the interval I, further if ξ contains the head then also replacing $\xi.pos$ with $\zeta.pos$ and $\xi.state$ with $\zeta.state$.

Definition 4.7 (Coordination)**.** The state is called ***coordinated*** with the content of the observed cell if it is possible for them to be in some healthy configuration.

Of course, it would be possible to give a finite table describing the coordination conditions. But we just point out some consequences of coordination we will use later:

Lemma 4.8 (Coordination)**.** *Each $Core = (Addr, Sw, Drift)$ value determines uniquely the c.Core value of the cell it is coordinated with, with one exception: when the cell is empty, $Addr \notin [0, Q)$, and Sw and Drift shows we are stepping onto a cell the first time in a transfer sweep.*

In the reverse direction, the relation is less strict: each $(c.Addr, c.Sw)$ pair determines uniquely the Addr that can be coordinated with it, and requires $Sw \in \{c.Sw, c.Sw+1\}$, with the following exception: when c.Addr is within 4β of a colony end.

Proof. The exception comes from the fact that there are two ways for the head to step onto cells of a neighbor colony: either during the transfer sweep, or at times when the head makes a turn at the end of a sweep, and after moving forward $Z - 4\beta$ steps, zigs back Z steps, thereby reaching 4β steps into the neighbor. □

To describe the self-correction process, we need to characterize the kind of configurations that can be found during it. We cannot hope to restore health in all islands created by faults, in a very short time after the faults occurred. Indeed, as seen from Figure 4, it may happen that a burst creates an island, but leaves it with a state of the head that will not require it to zig back anymore. Moreover, this may happen in the last sweep of a work period, while moving the base, say, to the left: so the island created this way will be seen next, if ever, only if the simulated computation transfers the base right again.

The following definition classifies the kinds of alteration that noise can bring to a healthy configuration. Informally, in islands, the structure may have been damaged, while in stains, only the *c.Info* and *c.State* tracks could be. The distress area is where structure is currently being restored.

Recall that the $Core = (Addr, Sw, Drift)$, the *c.Core* and *c.Rec.Core* tracks are analogous.

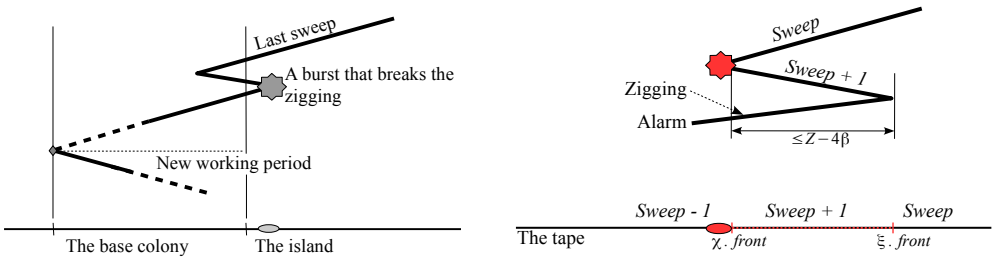


Figure 4: (a) A burst during the last visit of the colony, at the bottom of a zig. It puts the state into normal mode, with appropriate values of *ZigDepth* and *ZigDir*. This leaves the created island "undetected" until the head returns to the colony (b) A burst switches the sweep value causing the head to move forward and leaving an island and a part of the tape without incremented sweep number.

Definition 4.9 (Annotated configuration)**.** An ***annotated configuration*** is a quadruple

$$(\xi, \chi, \mathcal{I}, \mathcal{S}, D),$$

with the following meaning: ξ is a configuration, χ is a healthy configuration, $\mathcal{I}$ is a set of intervals of cells called ***islands***, further $\mathcal{S} \supset \mathcal{I}$ is a set of intervals of cells called ***stains***, D is an interval containing the head called the ***distress area***.

The distress area contains any island containing the head.

Islands and stains are of size $\leq \beta$. The distress area has size $\leq 3E$.

We can obtain χ from ξ by changing

- the *c.Core* track in the islands and $< 2Z$ additional cells within D;
- the *c.Info* and *c.State* track in the stains;
- the state, the *c.Rec.Addr* track in D, and the head position inside D.

We say that an interval W is the ***workspace*** of the annotated configuration if it is the workspace of χ. The following additional properties are required:

Islands At most one island intersects the workspace. There are at most 2 islands in each colony that do not intersect the workspace. If there is more than one, then one is within distance $E + 5\beta$ from the colony boundary towards the base colony.

Stains In the base colony, either all stains but one are within a distance $E + \beta$ to the left colony boundary, or all but one are within a distance $E + \beta$ to the right colony boundary. In all other colonies, all stains but one are within distance $E + \beta$ of the boundary towards the base colony.

Distress If D is empty then the mode is normal.

We say that a cell is ***free*** in an annotated configuration when it is not in any island or D. The head is ***free*** when D is empty.

Definition 4.10 (Admissible configuration)**.** A configuration ξ is ***admissible*** if there is an annotated configuration $(\xi, \chi, \mathcal{I}, \mathcal{S}, D)$. In this case, we say that χ is a healthy configuration ***satisfying*** ξ. Any change to an admissible configuration is called ***admissible***, if the resulting configuration is also admissible.

The following key lemma shows that an admissible configuration can be locally corrected. Recall outer cells from Definition 4.3. For convenience, we introduce another parameter:

Definition 4.11. Let $\Delta = 3Z$.

Though all the parameters β, Z, Δ, E, Q are within constant factor of each other, it is convenient to think of them as

$$\beta \ll Z \ll \Delta \ll E \ll Q,$$

since we can increase the quotients between them as it suits our analysis.

Lemma 4.12 (Patching)**.** *Consider an annotated configuration*

$$(\xi, \chi, \mathcal{I}, \mathcal{S}, D),$$

and an interval $R = [a, b)$ of length $2E$.

Assume that either in the left half or the right half of R, at least $E - 3\beta$ cells of $\xi(R)$ are nonempty. Then it is possible to compute from $\xi.c.Core(R)$ an interval $\hat{R} = [\hat{a}, \hat{b})$, a local configuration $\zeta = \zeta(\hat{R})$ with no empty cells, such that $\chi|\zeta(\hat{R})$ is healthy, and the following holds.

(a) *The states of nonempty cells of ξ can differ from the corresponding cells of ζ only in the islands, and in the interval D.*

(b) *The computation of ζ can be carried out by the machine M_1 (relying only on ξ and R), using a constant number (independent of β, Q) of passes over R, and a constant number of fields containing values of size $\leq Q$.*

(c) *If $\chi.pos < a + 2\Delta$ then $\hat{a} = a + 4\Delta$, else $\hat{a} = a$. The same conditions define $\hat{b}$, interchanging left and right.*

Before proving the lemma, we need a notion of plurality.

Definition 4.13 (Plurality)**.** The ***plurality*** of some field $c.F$ over some interval I, is the value that appears the most, but at least $1/3$ of the times. If there is no such value then any value of the field appearing in the interval will be accepted as a result of this operation.

For a rule to compute the plurality, we use a version of an algorithm from [8]: Running in a single sweep over I, the rule maintains pairs of (v_i, c_i), $i = 1, 2$ that store two candidate plurality values of a track $c.F$ and their current weight. In each step, looking at the current value of $c.F$, if $c.F = v_i$ or $c_i = 0$ for some $i \in \{1, 2\}$, then c_i is incremented and $v_i \leftarrow c.F$. Else both c_1 and c_2 are decremented.

Proof of Lemma 4.12. For any interval I, let $\alpha(I)$ denote the plurality value of $\xi.c.Addr(x)-(x-a)$ over I, further $\sigma(I)$ and $\delta(I)$ the plurality value of $\xi.c.Sw(x)$, and $\xi.c.Drift(x)$ over I. Let $m_\alpha(I)$, and so on, denote the multiplicity of $\alpha(I)$, and so on, over interval I. Empty cells are not counted in the total, and do not contribute to the counts.

We now outline a procedure that finds $\hat{R}$ and ζ. Even if the reasoning below refers to the healthy configuration χ occasionally, the computation only relies on the configuration ξ.

1. Assume that at least $2E-9Z$ cells of R are left outer cells of ξ, or at least $2E-9Z$ are right outer cells.

   Without loss of generality, assume that at least $2E-9Z$ cells in R are left outer cells: set $\hat{R} \leftarrow [a,b-12Z)\big(=[a,b-4\Delta)\big)$. The value $\sigma[a,b-E/2)$ is necessarily Last(1). Let $\alpha=\alpha[a,b-E/2)$. Setting $\zeta.c.Addr(x)=\alpha+x-a$, $\zeta.c.Sw=\text{Last}(1)$, and $\zeta.c.Drift=-1$ defines $\zeta.c.Core(x)$ accordingly for all x in $\hat{R}$ (not leaving empty cells).

Assume that the above test fails: then given that R intersects at most 3 islands, we can assume that at least $9Z-3\beta$ cells of R belong to the workspace of the healthy configuration χ.

2. Suppose that ξ has at most 3β outer cells in R.

   Then the non-workspace cells of

$$R^- = [a^+,b^-) = [a+3\beta, b-3\beta)$$

   are all island cells, since the non-island non-workspace cells of R must all be at the ends.

   Compute $\sigma(R^-)$, and assume without loss of generality $\text{dir}(\sigma)=1$ (the right sweep).

   We claim

$$\text{front}(\chi)-Z \le a^+ + m_\sigma \le \text{front}(\chi)+Z.$$

   Indeed, in the healthy configuration χ, the right-sweeping cells inside R^- form an interval on the left of front(χ). By the definition of annotation, m_σ could differ from the size of this interval only due to island cells, and possibly in the interval D containing front(χ), whose size is $<2Z$ by Definition 4.9.

   2.1. Compute the addresses and sweep values.

   Recall that we assumed $\text{dir}(\sigma)=1$. First we compute the candidate address and sweep values in $[a,a^+ + m_\sigma)$.

   Admissibility implies that $\xi.c.Addr(x)-(x-a)$ should be constant as x runs on all non-island workspace cells of R^- with the above plurality value of σ. Therefore it has some plurality value α.

   For cells x in $[a,a^+ + m_\sigma)$, let $\zeta.c.Sw(x) \leftarrow \sigma$, $\zeta.c.Addr(x) \leftarrow \alpha+x-a$. This can change only island cells or shift the front to the left by a number of cells equal to the number of island cells encountered in this interval.

   If $a^+ + m_\sigma \ge b-9Z$ then set $\hat{R} \leftarrow [a,b-12Z)\big(=[a,b-4\Delta)\big)$.

   Assume now $a^+ + m_\sigma < b-9Z$, and set $\hat{R} \leftarrow R$.

Now we compute the candidate address and sweep values in $[a^+ + m_\sigma, b^-)$. Due to admissibility, the values $\xi.c.Sw(x)$ and $\xi.c.Addr(x) - (x-a)$ should be constant for almost all x in $[a^+ + m_\sigma, b^-)$; let σ', α' be the respective pluralities. For x in $[a^+ + m_\sigma, b)$, set $\zeta.c.Sw(x) \leftarrow \sigma'$, $\zeta.c.Addr(x) \leftarrow \alpha' + x - a$. This again can only change island cells or possibly some cells due to the left shift of the front: the total number of these changes is still at most 3β.

2.2. Compute the remainder of ζ.

Assume first $|\sigma' - \sigma| = 1$, that is they are two consecutive sweep values within a work period.

If $\sigma' < \sigma$, then set $\mathrm{front}(\zeta) \leftarrow a^+ + m_\sigma$, $\zeta.Sw \leftarrow \sigma$; otherwise $\mathrm{front}(\zeta) \leftarrow a^+ + m_\sigma - 1$, $\zeta.Sw \leftarrow \sigma'$. If $\min(\sigma, \sigma') \geq \mathrm{TfSw}(1)$ then all over $\hat{R}$, set the $\zeta.c.Drift$ values to the plurality of the $\xi.c.Drift$ values over R.

Assume now that σ, σ' are the two values corresponding to the transition to a new work period.

By assumption $\sigma = 1$; we set $\mathrm{front}(\zeta) \leftarrow a^+ + m_\sigma$.

The value $\chi.c.Drift$ has a constant value δ on R. We can determine it using plurality of $\xi.c.Drift$ over R^-, and replace it all over $\hat{R}$.

Assume that the test 2 also fails: then R intersects the workspace without being essentially contained in it, or essentially disjoint from it.

3. We have the following, not mutually exclusive possibilities, that can be checked:

   - All but 3β cells of the left/right half of R are outer cells.
   - All but 3β cells of the left/right half of R are workspace cells.

Proof. This follows from the fact that in the healthy configuration χ, the workspace is surrounded by outer cells.

From the four possibilities listed in part 3 above, now only these remained: one half of R is essentially covered by workspace cells and the other one is not, or one half of R is essentially covered by outer cells and the other one is not. Given the left-right symmetry, we can therefore make the following assumption:

4. Assume that the left half of R is not covered essentially (that is to within 3β) by outer cells. Also, either the left half is covered essentially by workspace cells, or the right half is covered essentially by outer cells.

   Let m be the number of workspace cells of ξ in R. Given that assumption 1 fails, we have $m \geq 9Z - 3\beta$. The intersection of the workspace with R must agree within 3β with $[a, a+m)$, just as above in part 2.1. Now we can carry out the computations of part 2 in the intervals $[a, a+m)$ and $[a+m, b)$ in place of R^-, since an interval of size $9Z - 3\beta$ is still sufficient for the correct computation of the pluralities.

   In case it is found that the front is exactly at the end of the workspace then let Sw still set to the value found inside the workspace (and not 1 larger).

It is straightforward to check that the conditions of the lemma guarantee that the construction of ζ has the properties claimed in the lemma.

□

Assuming that the conditions of Lemma 4.12 hold, it is clearly possible to compute a constant upper bound on the number of sweeps of the domain R needed for the machine M_1 to perform the calculations, resulting in a bound $O(\beta)$ on the total number of steps used.

Definition 4.14 (Patching data)**.** The following information

$$\text{Pdata} = (s, \alpha, \alpha', \sigma, \sigma', \delta, f)$$

incorporates all the data defining the patched healthy local configuration $\zeta(\hat{R})$, provided R is given:

- $s \in \{1,2,3,4,5.6\}$ says which of the possibilities of part (c) of Lemma 4.12 (Patching) holds.
- $\alpha, \alpha', \sigma, \sigma'$ help computing the address and sweep values as seen in the proof of that lemma.
- δ is the *c.Drift* value shared by all elements of the workspace inside $\hat{R}$ in case $\sigma \geq \text{TfSw}(1)$ or $\sigma = 1$.
- $f = \text{front}(\zeta) - a$ in case $\hat{R} = R$.

4.2 Recovery procedure

Starting from a point z_1, the recovery procedure opens an interval

$$R = z_1 + [-E, E) = [a, b),$$

to which it applies the algorithm of the proof of the Patching Lemma 4.12. To guard against bursts, the algorithm is computed twice (in stages Planning_i, $i = 1,2$) before committing to the result (which also happens twice).

We will make sure that the interval R is positioned in a special way.

Definition 4.15 (Aligned intervals)**.** We require Q to be divisible by E. An interval is called ***aligned*** if its endpoints are divisible by E.

The above requirement guarantees that colonies are aligned. The recovery procedure will make the interval R aligned. The examples below motivate this.

Examples 4.16. 1. Consider the following scenario. During the leftward transferring sweep towards the left colony endcell e, while at e, the head hits an island, calling alarm. This starts a recovery procedure, which opens a recovery interval R. While the head is on the left boundary of this interval, a burst occurs. As a result of this burst, nothing changes inside the recovery interval, or in the head position or the state, but an island I is created at $e - E$. Assume that the computation from now on continues to the right of e. In some much later work period, at the last sweep before moving right from colony e, a burst leaves an island at $e + \beta$. Then, in some much later work period, during the transferring sweep to colony with left endcell e, the head hits this new island and the recovery starts. Now we repeat the same scenario as above, creating an island $I + \beta$ which will stay there. If we continue with this adversarial way of putting islands, almost the entire interval $[e - E, e)$ can be covered by islands.

Then, much later, in a transfer to colony with base $e-Q$, this may defeat the algorithm of the Patching Lemma 4.12.

With the alignment requirement, if the left end of R contains the left colony endcell e, then the left end of the interval R will always extend to the same point $e-E$, and hence the interval $[e-E,e)$ cannot become littered with islands.

2. Consider the following scenario. The head hits an island in the transfer stage, say, to the right, and a recovery starts. During recovery, in the right half of the recovery interval $[a,b)$, a new burst triggers a new alarm and new recovery, with a recovery interval $[a+d,b+d)$, for some $d>0$. The new recovery finishes, the head moves forward, leaving behind the marked cells in the interval $[a,a+d)$. This is undesirable, as we want the recovery to succeed locally, even when it it disturbed. The creation of aligned recovery intervals will eliminate this possiblity, too.

Given the importance to guarantee that new bursts cannot prevent the eventual success of recovery, we give a rather detailed description of the recovery procedure.

There is a point in which the Patching Lemma did not specify all the changes: when for example $\hat{R}=[a,b-4\Delta)$ then it only said that the head should go to the right of $\hat{R}$, not the exact place where it should go. In this case, the procedure will put the head at b, that is all the way to the right edge of R. Similarly, if $\hat{R}=[a+4\Delta,b)$ then the head goes to a, that is on the left edge of R. In both cases we set $ZigDepth=0$.

For controlling the details, the procedure uses the field $Rec.Addr$ to measure the (signed) distance from point z_1, and a field $Rec.Sw$ to measure the progress, just as in the main program. There are corresponding $c.Rec.Addr$ and $c.Rec.Sw$ fields in the cells. According to the values of $Rec.Sw$, we distinguish ***stages***, and introduce the ***pseudofield*** $Stage$ (it is just a function of $Rec.Sw$), with values

$$Stage \in \{\text{Marking}, \text{Planning}_i\ (i=1,2), \text{Committing}_i (i=1,2)\}.$$

The process makes use of a number of rules: `Alarm`, `Mark`, `Plan`(i), `Commit`(i) for $i=1,2$. Whenever we say that a rule "checks" something, it is understood that if the check fails, alarm is called. In all rules but in `Commit`(i), wherever the head steps, it walks on marked cells or it marks them, that is it sets $c.Rec.Core\neq 0$. The rule `Commit`(i) is devoted to unmarking. Zigging will be performed using the fields $Rec.ZigDepth, Rec.ZigDir$. It will keep the head within the recovery interval (in contrast to zigging in normal mode, which takes the head occasionally outside the workspace). It needs the following parameter:

Definition 4.17. Let $Rec.Z=Z/2$.

The following rule is going to run simultaneously through all the rest of the recovery procedure.

- Check if $c.Rec.Addr=Rec.Addr+d$, where $d=\pm 1$ is the direction of the sweep.
- If not zigging, check if $c.Rec.Sw=Rec.Sw-1$. If zigging, check if $c.Rec.Sw=Rec.Sw$.
- Update the field $Rec.Addr$ in every move, increasing or decreasing it as we move left or right.

Recovery goes through the following steps.

1. The rule `Alarm` sets *Mode* ← Recovering, *Stage* ← Marking.

2. Rule `Mark` locates and marks the recovery area by filling in *c.Rec.Addr* centered on z_1, and moves to $z_1 - E$. It alarms if any of the cells along the way that it expects to be already marked is not.

   In order to mark R, the head moves in a zigging way, similarly to what is done in the main simulation described in point 7.2 of Section 2, except that we do not go outside the interval R. Zigging makes sure not to mark too many cells in one sweep or without checking that they are marked consistently with what was marked before.

   After determining the interval R from examining a segment of $2Z$ cells, the rule marks one half of this interval, then passes over the marked half to mark the other half. Here are the details.

   The ***locating*** part (not marking yet) starts from a cell x (where the alarm was called), and ends on cell $x+Z$. In its sweep 1, moving left, it remembers the plurality of $c.Addr(y) - (y-x) \bmod Q$ for $y \in x + [-Z, 0)$ as a candidate modulo Q address λ_{-1} for x. If there is no such plurality, the value is undefined. It also computes a plurality sweep value σ_{-1} if a plurality exists. Now, the machine turns right and while passing over $[x, x+Z)$ it computes λ_1 and σ_1 similarly. Admissibility implies that if both λ_j are defined then they are equal. Moreover, if both are defined then at least one of the σ_j is defined.

   From these values, we will compute a candidate mod Q address λ and a candidate direction δ as follows.

   (1) If one of the pairs (λ_j, σ_j) is defined and the other one is not, then $\lambda \leftarrow \lambda_j$, $\delta \leftarrow -j$ (direction is towards the undefined pair). Otherwise let λ be the common value of the λ_j.

   (2) If $\sigma_{j'} \le \sigma_j \le \sigma_{j'} + 1$ or $\sigma_j < \sigma_{j'}$ then $\delta \leftarrow (-1)^{\sigma_j + 1}$, that is δ is the direction of the current sweep as defined in (3.3).

   From λ we can compute $x_0 = x - (x \bmod E)$ and $x_1 = x_0 + E$. Now we determine z_1 as follows: If $|x - x_j| < 2\Delta$ for some j then let $z_1 = x_j$. Otherwise, let $z_1 = x_j$ for the x_j with $\operatorname{sign}(x_j - x) = \delta$, so x will found in the rear half of R.

   The locating part achieves the following conditions.

   (a) The point x is in R, at least 2Δ away from its boundary. Moreover, if for example the sweep direction is to the right then x is in $[z_1 - E + 2\Delta, z_1 + 2\Delta)$.

   (b) If $|x - \text{front}(\chi)| < \Delta$, then R reaches less than $E + 3\Delta$ backwards from $\text{front}(\chi)$.

   Indeed, without loss of generality assume that the direction of the sweep is 1. Then $x - z_1 < 2\Delta$, hence $x - (z_1 - E) < E + 2\Delta$. The assumption and Definition 4.11 gives $x > \text{front}(\chi) - \Delta$, which in combination with the above yields $\text{front}(\chi) - (z_1 - E) < E + 3\Delta$.

   At the end of the locating part, being in a cell y, we set

$$Rec.Addr = y - (z_1 - E).$$

3. The following rule is going to run simultaneously through all the rest of the recovery procedure:

   Check the alignedness of the interval R every time you pass over an interval of size Z containing nonempty cells: for all but 3β of the cells the relation $Rec.Addr \equiv c.Addr \pmod{E}$ must hold. If it does not, go back the last Z steps and call alarm.

4. Rule `Calculate` carries out, over interval R, the algorithm of the Patching Lemma 4.12 to determine the interval $\hat{R}$ and the local configuration $\zeta(\hat{R})$. If none of the cases apply in the algorithm described in the proof, the rule calls alarm. Otherwise it remembers the computation result in a field Pdata as given in Definition 4.14.

5. Stages Planning_1 and Planning_2 follow each other. Stage Planning_i calls rule $\texttt{Plan}(i)$.

   $\texttt{Plan}(i)$ calls `Calculate`. In case $i = 1$, it writes the resulting $\zeta.c.Core$ values on the $c.Rec.Core$ track of $\hat{R}$. In case $i = 2$, it just checks whether the result is equal to the existing values of $c.Rec.Core$.

6. Stages Committing_1 and Committing_2 also follow each other.

   Rule $\texttt{Commit}(1)$ unmarks the cells over R, setting $c.Core \leftarrow c.Rec.Core$ at the same time, if $c.Rec.Core \notin \emptyset$. It relies on the field $Rec.Addr$, and also on the data Pdata introduced in Definition 4.14, (and computed in each stage Planning_i). In more detail, it works as follows.

   If $\hat{R} = [a, b-4\Delta)$ then it moves from the left end of R to the right end while unmarking, and stays there. If $\hat{R} = [a+4\Delta, b)$ then it moves from the right end to the left end while unmarking. In these cases, recovery ends here.

   Otherwise, it first moves to the end of R in direction $-\text{dir}(\zeta.Sw)$ (that is backward from the sweep direction of ζ).

   Finally, it erases the marks up to position $\text{front}(\zeta)$.

   Rule $\texttt{Commit}(2)$ moves to the other end, and moves towards the front, erasing marks and ending up at $\text{front}(\zeta)$ with no marked cells left.

   Zigging is used during the committing stage just as during the marking stage.

   The commitment part, along with the marking part, achieve the following condition:

   (a) The marked cells always form an interval containing the head.

It is easy to check that the recovery procedure uses only a constant number of sweeps, for a total number of steps

$$K_R = O(\beta). \tag{4.1}$$

5 Proof of the Main Theorem

It is useful to spell out the kind of simulation that machine M_1 performs.

Definition 5.1. A computation history in the sense of Definition 1.3 is a (β, V)***-noisy trajectory***, if faults in it are confined to bursts of size $\leq \beta$ separated by time intervals of size $\geq V$.

A pair of mappings (φ_*, Φ^*) in the sense of Definition 3.3 is a (β, V)***-tolerant simulation*** of Turing machine M_2 by Turing machine M_1 if for every string $x \in \Sigma_2^*$, every (β, V)-noisy trajectory η of M_1 whose initial configuration is $\varphi_*(x)$, the history $\Phi^*(\eta)$ is a trajectory of M_2.

The proof of the main theorem will show, as a side result, that our simulation is a (β, V)-tolerant simulation of M_2 by M_1. We assume that the output of M_2 is 0 or 1 written in cell 0. It is time to define more precisely the concepts connected with recovery.

5.1 Annotated history

Let us analyze the kind of histories that are possible with sparse bursts of faults. Recall the definition of annotated configuration in Definition 4.9.

Definition 5.2 (Annotated history)**.** An ***annotated history*** is a sequence of annotated configurations if its sequence of underlying configurations is a (β, V)-trajectory, and it satisfies some additional requirements given below.

If the head is in a free cell, in normal mode, then the time (and the configuration) will be called ***distress-free***. A time that is not distress-free and was preceded by a distress-free time will be called a ***distress event***. Consider a time interval $[t, t+u)$ starting with a distress event and ending with the head becoming free again. It is called a ***relief event*** of ***duration*** u if the only possible island that remains from the distress area is due to some burst that occurred at a time intersecting $[t, t+u)$. Moreover, if such an island exists, then the sweep direction from before the distress event is preserved, except when the island is outside the extended base colony—then it will be reversed. The ***extent*** of a relief event is the maximum size interval covering the distress area during the distress.

Recall the definition of the parameter K_R in (4.1). The additional requirements for annotated history are:

(a) Islands are only created by noise. Stains and the distress area start out as islands.

(b) Each distress event is followed immediately by a relief event, of duration $\leq 4K_R$ and extent $\leq 4E$.

(c) If a distress-free configuration has proceeded beyond the computation phase of the work period, that is $Sw \geq \mathrm{TfSw}(1)$, then the base colony contains no stains from earlier work periods.

Lemma 5.6 (Recovery), showing that a distress event is followed by a relief event, will be a crucial step towards the proof of the main theorem. Before spelling it out and proving it, we provide some preparatory lemmas.

5.2 Undisturbed recovery

The idea of the proof of relief from damage is the following. If alarm is called and the recovery process is allowed to complete, then it carries out the needed correction, as guaranteed by the Patching Lemma 4.12. Most complications are due to the fact that the state after a burst is arbitrary.

When the mode is normal then zigging will make sure that the effect is limited to near the island where the burst happened: for example, the direction of a sweep cannot be changed in the middle of the workspace, since then zigging would notice this and call alarm.

But the mode after the burst can be the recovery mode, with arbitrary values for all fields. Moreover, a new burst may occur after an alarm, at an arbitrary stage of the recovery.

In this section, we address the cases when two bad effects do not combine: either an alarm is called and completes without a new burst intervening, or a burst occurs at a distress-free time. Recall the definitions of the constants K_R in (4.1) and Z in Definition 3.2.

Lemma 5.3 (Undisturbed alarm)**.** *Suppose that in an annotated history, alarm sounds at a time when the front of the healthy configuration χ is at a distance $< 2Z$ from the head, and the size of the distress area is $< 3Z$. Suppose also that no burst occurs in the next K_R steps. Then the annotation of the history can be extended so that relief comes in fewer than K_R steps, and the distress areas during this are contained in a common interval of size $2E$.*

Proof. Assume that the conditions of the lemma hold. Let x denote the position of the head at the moment when alarm is called. Let us follow the recovery procedure, to show how the relief is achieved.

After the alarm, in the first two sweeps of the recovery procedure, interval $[x-Z, x+Z)$ is examined, and then, an interval $R = [a,b)$ of size $2E$ is opened that extends the existing distress area of size $< 2Z$. For the procedure to succeed, the additional condition of the Patching Lemma 4.12 must hold that in one half of R no more than 3β cells are empty. This is trivially true even when the alarm is called on the very first few steps of the simulation—since we have assumed that the address fields of the base colony and its two neighbors are nonempty.

Let us see that front(χ) and front(ζ) are in $\hat{R}$, and therefore the patching algorithm indeed restores a correct annotation. In the proof of the Patching Lemma, we used m_σ to denote the multiplicity of the plurality sweep σ within the interval $[a+3\beta, b-3\beta)$. Without loss of generality, assume that the direction of σ is 1.

The assumptions of the lemma along with definitions of $Z, E, Rec.Z, \Delta$ in Definition 3.2, Definitions 3.1, 4.17 and 4.11 imply that x is not further than Δ from front(χ). Now property (2b) of the recovery procedure implies front$(\chi) \le a + E + 3\Delta \le b - 4\Delta$. Furthermore, at least m_σ right sweeping cells in R will be on the left of front(χ). Since this is a plurality among at least $E - 3\beta$ cells, front$(\chi) > a + 4\Delta$. It follows from these two estimates that $\hat{R} = R$. The recovery procedure erases the marks, and rewrites all island cells in R, allowing us to erase the distress area and the islands in the final annotation. □

Lemma 5.4 (Burst)**.** *Assume that the history has been annotated up to a time when a burst, creating an island J_0, occurs at a distress-free time. Then the burst is followed by a relief event of duration $\le K_R + Z$ and extent $\le 3E$.*

Proof. We consider various situations after the burst. Recall that we called an interval R of length $2E$ ***aligned*** if in a healthy configuration satisfying the present one, its ends have addresses divisible by E (equivalently, if its end positions as absolute integers are divisible by E).

Let χ denote the healthy configuration that is part of the annotation at the time of the burst. Since the burst occurs at a distress-free time, the head is within Z from front(χ) when it happens. In what follows, we will sometimes refer to front(χ) of this moment as just the ***front***.

1. Assume first that the mode immediately after the burst is normal.

   Without loss of generality, assume that the sweep was to the right. We start at some position x that is either in island J_0 or next to it. Now the head zigs backward and forward Z steps (see Definition 3.2), with respect to the sweep direction, between any two sequences of $Z-4\beta$ forward moving steps. In any of these, it may discover an incoordination and call alarm, in which case Lemma 5.3 becomes applicable.

   1.1. Assume first that the burst does not change the sweep.

   In this case, the head will continue its forward sweep, with just possibly changed zigging. It may sense incoordination and call alarm. If this happens then Lemma 5.3 applies, since we are at most $Z+3\beta$ steps behind front, and at most 3β steps ahead it.

   Before the head manages to traverse J_0, it may hit another island causing an alarm. The point where this alarm can be called may be at most $Z-3\beta$ steps ahead of the front, so Lemma 5.3 applies again. In case of alarm (during the burst or later), the recovery area will include the island J_0, and if it was triggered by another island then that one, too. Suppose now that no alarm occurs.

   Any points of island J_0 traversed during the progress and zigging can be erased from the island, and after a complete cycle of zigging occurs the untraversed parts of J_0 may stay as an island.

   How can it happen that not the whole $J_0 =: [a, a+\beta)$ is traversed? In this case, the next backward zig does not cross J_0, so it starts from $\geq a+Z$. To get there we need front(χ) $\geq a+Z-(Z-4\beta) = a+4\beta$ when we start.

   1.2. Assume now that the burst changes the sweep *Sw*.

   Lemma 4.8 says that unless *c.Addr* is in a certain interval of length 4β, the pair $(c.Addr, c.Sw)$ determines uniquely the *Addr* value coordinated with it. If the burst changes *Addr* then therefore this will be noticed as soon as the head leaves the island and possibly this interval, causing an alarm, so Lemma 5.3 applies.

   Similarly, if *Sw* changes by more than 1 then it will be noticed, as soon as the head leaves the island or the area of size 4β mentioned. If it just changes by 1 then the head reverses direction, and the incoordination may not be immediately noticed when stepping off the island. But zigging will take us all the way across J_0 and therefore if alarm does not sound, J_0 can be erased (this can only happen if J_0 is at the end of the colony where the sweep would have changed anyway). Indeed, just as above, we can see that the only possibility that the next backward zig does not cross J_0 would be that the front is to the left of $a+\beta$ by at least 4β. But this is impossible, since as the original sweep is to the right, the head was not right of the front when the burst occurred.

2. Suppose that the mode after the burst is Recovering.

   In the recovery rule, as defined in Section 4.2, the head moves around in a marked area containing R,

of size $\leq 3E$. This area is extended in stage Marking, and shrunk in stages Committing$_i$. If the stage after the burst is Planning$_i$, then alarm is called almost immediately (possibly passing through some island cells first), since we assumed a start from a distress-free configuration, in which by definition no non-island cells are marked. Then Lemma 5.3 applies.

2.1. Suppose that the stage after the burst is Marking.

By its design, the marking rule marks new cells while also using a rule similar to $\texttt{Zigzag}(d)$, but marking at most $Rec.Z - 4\beta$ cells while moving in one direction. Alarm is only called when an alignment problem is found, or non-marked cells are found where marked ones are expected. Therefore alarm can only occur within the first $2Rec.Z$ steps after a burst. Indeed, zigging checks alignment with the cells marked earlier. If alignment inconsistency is not found then it will not be found later either. It follows that in case of new alarm, Lemma 5.3 applies, and the new recovery reprocesses all cells marked after the burst.

2.2. Assume that after the burst a committing stage is entered.

The committing stages only erase marks, and apply $c.Core \leftarrow c.Rec.Core$. Since we started in a distress-free configuration, we had unmarked cells everywhere but in the islands. It follows that within at most as many steps as the total length of islands possibly encountered, there is either an alarm due to not seeing marks, or return to normal mode. From there on, the analysis of part 1 applies.

□

5.3 Disturbed recovery

We would say that recovery is disturbed when a burst occurs during a recovery process started by an alarm. Since bursts are rare, the alarm in question must have happened then without a burst, that is on encountering some island J_1. Given no burst within the last V steps, this encounter could have occurred only while extending the base colony in the first sweep of the transfer phase.

Lemma 5.5 (Disturbed recovery)**.** *Assume that the history has been annotated up to a time when the head steps on an island J_1, in a transfer sweep, or in the first zigging into the neighbor colony immediately after this sweep.*

Then the annotation can be extended such that a relief event of duration $\leq 4K_R$ and extent $\leq 3E$ occurs.

Proof. Without loss of generality, assume that is the direction of the transferring sweep is to the right. Let χ denote the healthy configuration that is part of the annotation at the time when an alarm occurs at some cell

$$u.$$

In what follows, we will sometimes refer to front(χ) of this moment as just the ***front***.

In the transferring phase all structural (that is *c.Core*) information in non-empty non-island cells we pass is computable from the field *Core* of the state (see Lemma 4.8). Therefore if no alarm or burst occurs while the head is on J_1, then the part of J_1 that was passed can be deleted from the island. Alarm may still occur after a possible zigging, followed by an attempt to pass through J_1 completely, that is within $2Z$ steps. Suppose that no burst occurs that is either within next K_R steps after this, or within distance $2E$ of u. Then Lemma 5.3 is applicable, giving undisturbed recovery. Alarm then cannot occur until the next burst, at which time Lemma 5.4 becomes applicable.

From now on, we assume that alarm occurs at some time while the head is on J_1 (or over a cell next to it), and a burst occurs only at some later time t_1. If the head is still on island J_1, then Lemma 5.4 is applicable, so assume the burst occurs later, but still within K_R steps of the alarm, creating an island J_0. Let M denote the interval of marked cells at time t_1, created by the recovery process started by the alarm.

1. Suppose that new alarm will be called within $2Z$ steps after the burst at some cell u'.

   After the burst, we are within distance β from M. Due to zigging, the alarm occurs at distance $< Z$ of M. If the recovery before the burst did not yet determine alignment, that is z_1, then $|M| \le 2Z$. Since after the burst we are burst-free for a while, Lemma 5.3 guarantees the relief. Suppose now that z_1 has been determined already. By property (2a) of the locating part of recovery,

   $$u \in [z_1 - E + 2\Delta, z_1 + 2\Delta). \tag{5.1}$$

   After the burst, a new recovery area $R' = [a', b') = z'_1 + [-E, E)$ will be created. Because of alignment we have $z'_1 - z_1 \in \{-E, 0, E\}$. If $z'_1 = z_1$, then all cells of M will be reprocessed, and the recovery succeeds. Now, if alarm after the burst is called in the same interval (5.1) as the initial alarm then the same recovery interval will be opened again, hence $z'_1 = z_1$. Hence if $z'_1 = z_1 - E$ then $u' < z_1 - E + 2\Delta$, and if $z'_1 = z_1 + E$ then $u' \ge z_1 + 2\Delta$.

   1.1. Assume $z'_1 = z_1 - E$.

   If $\mathrm{front}(\chi) < z'_1 + E - 2\Delta$ then by the Patching Lemma we have $\widehat{R'} = R'$. After the new recovery finishes, marked cells in interval $M \setminus R'$, of length $\le E$, may still be there. However, the mode after the recovery is normal, and the sweep direction, assumed to be to the right, clearly does not change. Therefore, these marked cells will be reached, alarm will be triggered. Indeed, even if the front is at the colony boundary, and z_1 is the colony boundary (in which case the head is turning left), within $Z - 4\beta$ steps the zigging will start, and then the head will pass over z_1. By property (6a) of the recovery procedure, marked cells of the first recovery formed a contiguous interval containing the head before the new burst. It follows that if there are still some then z_1 is marked. As the head moves forvard over z_1, (or reaching it in zigging, in case z_1 is the right colony end),[1] the mark triggers alarm. Then a new recovery (the third in this sequence of events), with recovery interval equal to R again, will be undisturbed, and will eliminate remaining marks.

   If $\mathrm{front}(\chi) \ge z'_1 + E - 2\Delta = z_1 - 2\Delta$, then $\widehat{R'} = [a', b' - 4\Delta)$. Once the recovery over R' finishes, the head will be left on its right end, where alarm will be called just as above, with undisturbed recovery erasing the the remaining marks.

[1] In normal mode, we allow the head to zig into the neighbor colony in order to definitely reach all remaining marked cells.

1.2. Consider the case $z'_1 = z_1 + E$.

Now the new recovery interval does not contain the front 2Δ deep inside. Indeed, alarm at u was called on an island either when moving right, or while zigging at most 4β deep into a right neighor colony. Therefore, $u \leq \text{front}(\chi) + Z - 4\beta$. Since $u \in [z_1 - E + 2\Delta, z_1 + 2\Delta)$, the front cannot be in $[a' + 2\Delta, b' - 2\Delta)$ as $a' = z'_1$, hence the Patching Lemma 4.12 yields $\widehat{R'} = [a' + 4\Delta, b')$.

Once the recovery completes, the head is put into $a' = z_1$. Just as noted above, if there are any marks left from the first recovery then $a' - 1$ is marked. Therefore new alarm will be called when the mark is discovered during zigging. The new recovery area after this alarm is R again, and the process eliminates the remaining marks.

2. Suppose that alarm will not be called within $2Z$ steps after the burst.

2.1. Suppose that the burst ends in normal mode.

If $J_0 \supseteq M$ then the proof of Lemma 5.4 is applicable. Otherwise, as zigging meets the marked cells in M within $2Z$ steps, a new alarm will be called, and part 1 is applicable.

2.2. Suppose that the stage after the burst is Marking.

If the recovery process continues the old one seamlessly, then it terminates with success. Otherwise, since the marking stage employs zigging, alarm occurs within $2Rec.Z < 2Z$ steps. An analysis identical to the one in the proof of Lemma 5.4 shows that the cells marked between the burst and the new alarm will be contained in the new recovery area. From here on, the analysis of part 1 is applicable.

2.3. Suppose that the stage after the burst is Planning$_i$ or Committing.

Since these stages expect to walk over a recovery area, they must seamlessly continue what went on before, except for changing the state and the content of some cells in an island—otherwise alarm occurs immediately.

If the burst occurs during Planning$_1$, and it changes what is computed, then Planning$_2$ will notice this and trigger alarm. Since this alarm occurs in the existing marked area M, the analysis of part 1 still applies.

If the burst occurs during Planning$_2$ or Committing then it either triggers alarm, in which case the above analysis applies, or it allows the recovery process to end, with the lasting effect of the burst restricted just to the island J_0.

Lemma 4.12 guarantees that whatever assignments $c.Core \leftarrow c.RecCore$ were made in the committing stage, they are admissible—even if committing will be interrupted by a burst (and then continued as committing).

To bound the duration of relief, note that there were at most 3 recovery cycles, and alarm started within $2E$ steps between them. To bound its extent, note $|R \cup R'| \leq 3E$.

□

5.4 Finishing the proof

The following lemma implies the main theorem.

Lemma 5.6 (Recovery)**.** *Assume that machine M_1 starts working on a tape configuration of the form $\varphi_*(x)$. Every (β, V)-**noisy trajectory** of M_1 can be annotated.*

Proof. Assume that the history has been annotated in an admissible way up to a certain time. First we show that in case a distress event occurs, the annotation can be extended to keep property 5.2(b) of an annotated history. Then using this, we will show that in case of no distress, the annotation can also be extended in an admissible way while keeping the other properties.

1. Consider property 5.2(b).

   If a distress event occurs due to a burst then Lemma 5.4 applies. Assume now that a distress event occurs due to stepping onto an island J_1.

   Assume first that no burst occurs in the following $3K_R$ steps. Now, if no alarm sounds within $2Z$ steps, then zigging guarantees that the part of the island passed over can be replaced with a stain. (The only way not to pass some part is when the island is in a neighbor colony at distance $\approx 4\beta$ from the boundary: zigging may reach just a part of it.) If alarm sounds, then Lemma 5.3 applies.

   If there will be a burst within the following $3K_R$ steps, then no burst occurred within V steps before it. There could not have been any distress in the last sweep: indeed, any earlier island on which the head could have stepped would have been eliminated (at least its part in the path of the head) without or with alarm, as seen in the previous paragraph. But then the only way to step on an island is under the conditions of Lemma 5.5.

2. Consider property 5.2(a).

   Assume an admissible annotated history until a distress-free time t. We will show that by just keeping the islands constant, the annotation is extendable in an admissible way to $t+1$. In particular, there will still be a satisfying healthy configuration.

   Looking at Definition 4.4 of healthy configurations, most properties are obviously preserved in each step by just the form of the transition rule. The exceptions are the property which requires that the *c.Drift* track holds constant values in certain intervals at certain times, and the property which requires that *c.Info* and *c.State* tracks hold valid codewords of the code (φ_*, φ^*) defined in Section 3.1.

   So we are only concerned with the recomputation of the values of *c.State*, *c.Info*, *c.Drift* in the base colony, during the computation phase, and then the transfer of *c.State* during the transfer phase. (The value of *c.Drift* in the neighbor colonies is inherited from earlier, and its spreading from *Drift* is watched over by the coordination requirement: a change would trigger alarm at zigging.)

   Recall the structure and the tasks of the computation phase given in Section 3.3. By the properties of annotated configurations in Definition 4.9, in the base colony, besides a possible island, there is at most 1 more stain of size β, and possibly more stains, all contained in a single interval of size $E+\beta$. (The bound comes from the length of possible penetration of the head in a neighboring colony while faults could occur.) These last stains can be ignored, since our code is defined in such a way that it places a

codeword of the $(\beta,2)$ burst-error-correcting code (υ_*,υ^*) at a distance $1.1E$ away from the colony boundaries.

The recovery rules do not change the *c.Hold*, *c.State* and *Info* tracks, they do not change *c.Drift* track either before the transfer sweeps. Therefore, since there are at most 2 stains at distance $\geq 1.1E$ from the boundaries, and our code is $(\beta,2)$ burst-error-correcting, the result of decoding from the *Info* and *State* tracks during the computation phase will be the same as if the configurations had been stainless all along.

Any distress event will directly affect at most one of the three repetitions of the computation phase: the workspace is free during the others. Consequently, the correct values will will be stored in track $c.Hold[i]$, $i \in \{1,2,3\}$ for all but one i. If the sweep of the field majority computation during the encoding stage of the computation phase is distress-free, then every cell will receive the correct value $\mathrm{maj}(c.Hold[1\ldots3])$. But even if distress occurs in this sweep, relief guarantees that all cells but the ones in the island of the burst that caused the distress will hold the correct value.

The same argument proves the property that the newly computed *c.State* will be correctly transferred to the extended base colony in the transfer phase.

3. Consider property 5.2(c).

From the above argument it is clear that the only possible stain remaining in the base colony is the one created by a burst in the current work period. On the other hand, we can add stains and islands to neighbor colonies.

Let us see what is the farthest distance to which we can intrude into a neighbor colony and leave islands. With zigging, the head can penetrate at most 4β cells into the neighbor colony, where it can find an island causing alarm. A burst ocurring anywhere in the recovery interval created by this alarm may leave a stain anywhere within distance of $E+\beta$ from the colony boundary (where the recovery interval is centered).

□

Lemma 5.7 (Simulation)**.** *Under the conditions of Lemma 5.6, via some simulation function Φ^* (to be defined in the proof of the present lemma), the movement of the base colony corresponds to the head movement of the simulated machine M_2 (scaled up by a factor of Q). Whenever the sweep in the free cells of the base colony is not one of switching to a new work period, the array of c.State values there decodes into the state of M_2, and the array of c.Info values decodes into the current tape cell symbol of M_2.*

Proof. Lemma 5.6 gives us an admissible history. At all distress-free times, it also defines uniquely a base colony. For distressed times, let the base colony be equal to that of the last distress-free time. Once a base colony is given for each configuration, the simulation function is also uniquely defined: we decode the simulated cell content of each cell of M_2 from the corresponding colony, and the simulated state from the *c.State* array of the base colony. Part 2 of the proof of Lemma 5.6 shows that the decoding indeed defines a trajectory of M_2. □

Proof of Theorem 1.10. The statement follows essentially from Lemma 5.7, adding only the following. Let f be a projection from the alphabet of M_1 to the alphabet of M_2, defined by $f(s) = s.c.Info$. Consider

now the cell at the origin of the tape. Then, relation (1.3) holds due step 1c of the computation procedure in section 3.3.

How can we bound Q? Since the program p_2 of machine M_2 must fit in a colony, we need $Q = \Omega(|p_2|)$. Definition 3.2 and 3.1 show $E = O(\beta)$. The code (ϕ_*, ϕ^*) in Section 3.1 must fit into the part of the colony away by $1.1E$ from the boundary: this is satisfied if it has size All these requirements are satisfied with some $Q = O(s_2 + \beta)$, where $s_2 = |p_2| + \log|\Sigma_2| + \log|\Gamma_2|$. The computation phase lasts a constant number of sweeps, since we can use a very simple $(\beta, 2)$-burst-error correcting code, (see 5-fold repetition). The transferring of the *State* into the neighboring colony will need $O(s_2 Q)$ sweeps, as information of size $O(s_2)$ will be transferred to distance Q. Therefore the constant V bounding the time overhead of machine M_1 is $V = O(s_2 Q)$. □

6 Conclusions and future work

In this paper we have shown that for any Turing machine there is one that can simulate it while correcting occasional violations of its own transition function. The procedure recovering the simulation structure is based on an organization in which any group of cells affected by the faults is surrounded by cells that conserve some valid traces of the computation.

We hope to use this construction, similarly to [3], as a building block in a more complex construction of a Turing machine that can resist faults occurring independently with small probability.

To the best of our knowledge, this is the first construction of a reliable sequential machine. An interesting question is if the Turing machines are the simplest machines that can perform universal computation under isolated bursts of noise. It seems that simpler models, like the counter machines of [7], are insufficient, but there are some interesting questions open concerning the nature of their insufficiency.

References

[1] E. ASARIN AND P. COLLINS: Noisy turing machines. In *ICALP'05*, Lecture Notes in Computer Science 3580, pp. 1031–1041, 2005. 2

[2] PETER GÁCS: Reliable computation with cellular automata. *Journal of Computer System Science*, 32(1):15–78, February 1986. Conference version at STOC' 83. 2

[3] 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. 2, 35

[4] 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

[5] SHAFI GOLDWASSER AND GUY N. ROTHBLUM: How to compute in the presence of leakage. In *Proc. of the 53-th IEEE FOCS Symposium*, pp. 31 –40, October 2012. [doi:10.1109/FOCS.2012.34] 2

[6] GEORGII L. KURDYUMOV: An example of a nonergodic homogenous one-dimensional random medium with positive transition probabilities. *Soviet Mathematical Doklady*, 19(1):211–214, 1978. 2

[7] MARVIN MINSKY: *Computation: Finite and Infinite Machines*. Englewood Cliffs, New Jersey, 1967. 35

[8] J. MISRA AND D. GRIES: Finding repeated elements. *Science of Computer Programming*, 2:143–152, 1982. 20

[9] NICHOLAS PIPPENGER: On networks of noisy gates. In *Proc. of the 26-th IEEE FOCS Symposium*, pp. 30–38, 1985. 1

[10] DANIEL A. SPIELMAN: Highly fault-tolerant parallel computation. In *Proc. of the 37th IEEE FOCS Symposium*, pp. 154–163, 1996. 1

[11] ANDREI L. TOOM: Stable and attractive trajectories in multicomponent systems. In R. L. DOBRUSHIN, editor, *Multicomponent Systems*, volume 6 of *Advances in Probability*, pp. 549–575. Dekker, New York, 1980. Translation from Russian. 1

[12] 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

7 Glossary

For reference we list here the fields and rules in the transition function of the Turing machine constructed, and mentioned in the paper.

7.1 Fields

Addr and the corresponding cell field $c.Addr$ range from $-Q$ to $2Q-1$. The values $-Q$ to -1 are taken during the left drift, while the values Q to $2Q-1$ during a right drift. During the first sweep of the work period, $c.Addr$ is reduced modulo Q.

There are corresponding fields $Rec.Addr, c.Rec.Addr$ serving the recovery rule.

Core The triple of fields $(c.Addr, c.Sw, c.Drift)$ will determine the role played by a cell in the colony work period: for notational convenience, we introduce the names

$$Core = (Addr, Sw, Drift), \quad c.Core = (c.Addr, c.Sw, c.Drift). \tag{7.1}$$

The field $c.Rec.Core$ serving the recovery rule is the planned new value of the *Core* field.

Dir stores the direction of the previous step.

Drift stores the direction of the simulated machine M_2. It may have values $\emptyset, -1, 0, 1$. The value $\emptyset$ corresponds to the case when the new *Drift* is still not computed, and will also be the default value (for example in empty cells). The *c.Drift* field of the cells of the extended colony correspond to *Drift* in the state.

c.Hold Even though we store information with redundancy, faults can disturb the coding and decoding operations and the simulating computation itself. Therefore these procedures will be repeated several times, and their results, serving as candidates of the final values of $(Info, Drift, State)$, will be stored in the *c.Hold* track. The different candidates will be stored in the different parts of the *c.Hold* field, which is actually a small array *c.Hold*[1], *c.Hold*[2], *c.Hold*[3]. The subfield *c.Hold*[i].*Info* holds the value of the i-th candidate for the new *Info* value of the current cell.

c.Info, *c.State* represent the tape symbols and the state of the simulated machine M_2.

Mode shows whether the computation is in normal or recovering mode.

Rec, *c.Rec* The fields used in recovery mode are all collected as subfields of the field *Rec* of the state, and the field *c.Rec* of the cell state.

Sw numbers the sweeps through the colony. The first sweep of a work period has number 1 and is to the right, and this way each right sweep is odd, each left sweep is even. Thus the sweep direction of the head is completely determined by the parity of *Sw*, unless the head is at a "turning" point. At turning points, *Sw* is incremented.

Field *c.Sw* holds the number of the most recent sweep. The simulation consists of a ***computation phase*** and a ***transferring phase***, each corresponding to a certain interval of sweep values to be specified below (these intervals depend somewhat on the value *Drift*).

There are field *Rec.Sw* and *c.Rec.Sw* with a similar role in the recovery rule.

Stage is a function of *Rec.Sw*, dividing recovery into stages.

ZigDepth, *ZigDir* control the zigging rule. There are fields *Rec.ZigDepth* and *Rec.ZigDir* for the recovery rule, with a similar role.

7.2 Main rules

`Alarm` is the rule initializing the recovery rule.

`Calculate` is a subrule used in subrules `Plan`(i) of the recovery rule.

`Compute` performs the main computation of the computing phase.

`Mark` is the subrule of the recovery rule that marks the recovery area.

`Move`(d) is the rule applied at each step when moving the head in direction d.

`Plan`(i), `Commit`(i) are subrules working in different stages of the recovery rule.

`Swing`(a,b,u,v) is a rule controlling the details of the sweeping movement.

`Zigzag`(d) The sweep-through is interrupted by switchbacks called ***zigging***, described by a rule `Zigzag`(d). Here d is the direction of sweep. The process also depends on a fixed parameter Z given in Definition 3.2, and is controlled by the fields *ZigDepth* and *ZigDir* of the state.

Rule 7.1: `Move`(d)

Dir $\leftarrow d$ // $d \in \{-1,0,1\}$.
if *Mode* = Normal **then** *Addr* $\leftarrow$ *c.Addr* $\leftarrow$ *Addr* $+ d$
else *Rec.Addr* $\leftarrow$ *Rec.Addr* $+ d$
Move in direction d.

Rule 7.2: `Zigzag`(d)

// $d \in \{-1,1\}$ is the direction of progress.
if *ZigDir* $= -1$ **and** ((*ZigDepth* $= 0$ **and** $(Z-4\beta)|$*Addr*)
or $0 <$ *ZigDepth* $< Z$) **then**
    ZigDepth++
    if *ZigDepth* $= Z-1$ **then** *ZigDir* $= 1$
    `Move`$(-d)$
else if *ZigDir* $= 1$ **or** (*ZigDepth* $= 0$ **and** $(Z-4\beta) \not|$*Addr*) **then**
    if *ZigDepth* > 0 **then** *ZigDepth*$--$ **else** *ZigDir* $\leftarrow -1$
    `Move`(d)

7.3 Symbols and parameters

β is the bound on the size of bursts.

Δ This parameter also controls the recovery rule: it is defined in Definition 4.11, and is intermediate in size between Z and E.

D is the distress area that is part of an annotated configuration.

E This parameter controls the recovery rule, and is also used in defining the block code. It is set in Definition 3.1.

χ is the healthy configuration that is part of an annotated configuration.

$\mathcal{I}$, $\mathcal{S}$ are the sets of islands and stains that are part of an annotated configuration.

ϕ The block code of the simulation.

K_R is a bound on the number of steps of the recovery procedure, defined in (4.1).

M_1, M_2 are the simulating and the simulated machine.

Last(δ) is the number of the last sweep when the drift is δ.

Marking, Planning$_i$, Committing$_i$ are stages of the recovery procedure.

p_{encode}, p_{decode} are the programs of encoding and decoding the error-correcting code υ.

Pdata is the information computed by the patching procedure. It is given in Definition 4.14.

$R = [a,b) = [z_1 - E, z_1 + E)$ denotes the interval laid out by the recovery rule.

TfSw(δ) The sweep number at which transfer starts in direction δ.

U A fixed flexible universal machine introduced in Definition 1.7.

V is the lower bound on the time between bursts.

υ The burst-error-correcting code used in defining ϕ.

ξ, ζ denote configurations.

η denotes histories.

Z, *Rec*.Z. These parameters control zigging in normal and recovery mode. They are defined in Definitions 3.2 and 4.17.

AUTHORS

Ilir Çapuni
Computer Engineering Department, Epoka University
Tirana, Albania
icapuni@epok.edu.al

Peter Gács
professor
Computer Science Department, Boston University
gacs@bu.edu
http://www.cs.bu.edu/~gacs