JOURNAL OF COMPUTER AND SYSTEM SCIENCES 32, 15–78 (1986)

Reliable Computation with Cellular Automata

PETER GÁCS*

Boston University, Boston, Massachusetts 02215

Received December 1, 1983; revised July 30, 1985

We construct a one-dimensional array of cellular automata on which arbitrarily large computations can be implemented reliably, even though each automaton at each step makes an error with some constant probability. In statistical physics, this construction leads to the refutation of the "positive probability conjecture," which states that any one-dimensional infinite particle system with positive transition probabilities is ergodic. Our approach takes its origin from Kurdyumov's ideas for this refutation. To compute reliability with unreliable components, von Neumann proposed Boolean circuits whose intricate interconnection pattern (arising from the error-correcting organization) he had to assume to be immune to errors. In a uniform cellular medium, the error-correcting organization exists only in "software," therefore errors threaten to disable it. The real technical novelty of the paper is therefore the construction of a *self-repairing organization.*

1. INTRODUCTION

Can we avoid the accumulation of errors in arbitrarily large computations using unreliable components? The formal statement of this problem is based on the assumption that computing devices of arbitrary size must be built from a few elementary components. Each component makes errors with some frequency independent of the size of the device to be built. What are the architectures enabling us to deal with all combinations of errors likely to arise for devices of a given size?

We will consider the case when a failure does not incapacitate the component permanently, only causes it, in the step when it occurs, to violate its rule of operation. In the following steps, the component obeys its rule of operation again, until the next error. The case of permanent component failure may be of greater practical importance, but it has not been investigated in the same generality. (However, see [10], for some elegant geometrical results in a similar model.) There are reasons to believe that many of the techniques developed for the case of the

* This work was supported in part by the National Science Foundation Grants MCS 8110430, MCS 8104008, MCS 8302874, and DCR 8405270, and in part by DARPA Grant N00014-82-K0193 monitored by the ONR. Part of this research was done while the author was with the University of Rochester. This paper was requested for the Special STOC 83 Issue, but because of scheduling delays could not be included in that issue.

0022-0000/86 $3.00

571/32/1-2

transient failure will be applicable for the case of permanent failure. Another justification of this model is the interest it holds from the point of view of statistical physics.

Reliable computation with unreliable components must use massive parallelism. Information temporarily stored anywhere during computation is subject to decay and therefore must be actively maintained.

In 1953, von Neumann designed some reliable Boolean circuits. In his model, each component had some constant probability of failure. For a circuit consisting of n perfect components, he built a circuit out of $O(n \log n)$ unreliable components, computing the same function. (For an efficient realization of his idea, see [1].) In 1968, Taylor, using Gallagher's low-density parity-check codes, constructed a Boolean circuit out of $O(K)$ unreliable components and memory elements, capable of holding K bits of information for a polynomial number of steps. This construction was improved by Kuznietsov, using an idea of Pinsker, increasing the storage time to an exponential function of n.

All the above constructions suffer from the same deficiency: the circuits use a rather intricate connection pattern which cannot be realized in three-dimensional space with wires of constant length. On the other hand, the natural assumption about a wire is that as its length grows, its probability of failure converges to 1.

A cellular space (medium) is a lattice of automata in, say, three-dimensional space where every automaton takes its input from a few of its closest neighbors. First introduced by von Neumann and Ulam, such devices are now sometimes known as "systolic arrays" or iterative arrays. Typically, all automata are required to have the same transition function and are connected to the same relative neighbors, i.e., the device is translation-invariant. The spatial uniformity suggests the possibility of especially simple physical realization.

Cellular media are desirable computing devices, and it is easy to construct a one-dimensional cellular space that is a universal computer. (Take a one-tape Turing machine.) There is no known nontrivial design for reliable cellular medium made out of unreliable components. Work has been done on fault-tolerant cellular automata, e.g., in [6, 11]. However, these papers make the very strong assumption that two errors do not occur close to each other in space–time.

One of the main questions asked in connection with any medium is whether it is *ergodic*. Without going into the formal definition, it is clear that an ergodic infinite medium can not be used for computation since sooner or later, it forgets almost all information about its initial configuration. Besides being nonergodic, a computing device should be *stable*, in the sense that if its probability distribution is perturbed a little, it will still work reliably, and will certainly stay nonergodic.

Toom was the first one to construct stable nonergodic media (see [14, 15]). The set of sites is the two-dimensional lattice $\mathbf{Z}^2$. Each cell has two states, 0 and 1. The medium works according to a deterministic transition function. In one of the examples, to determine its state in the next moment, each cell computes the majority of the current states of the three cells consisting of its Northern and Eastern neighbor and itself.

In 1984, Reif noticed that a three-dimensional real-time reliable computing medium can be constructed using one of Toom's error-correcting rules in two-dimensional slices and the rule of an arbitrary one-dimensional medium across the slices. The reliability of the infinite version of this construction follows from [15]. In [3], we used the technique of "k-sparse sets of errors" first developed in the present work to give an efficient finite version of Toom's theorem. Using this result, one can now do the following. Given K cells of a one-dimensional medium D working for T steps, one can build a three-dimensional medium M on the set $\{1,..., K\} \times \mathbf{Z}_m^2$. Here $m = \log^{1+\varepsilon}(KT)$ and $\mathbf{Z}_m$ is the group $\{0,..., m-1\}$ of remainders modulo m (with $\mathbf{Z}_\infty = \mathbf{Z}$, the set of integers). When started with the appropriate input (each site in a torus-shaped slice $\{i\} \times \mathbf{Z}_m^2$ receives the same input symbol), the medium M will simulate D reliably step-for-step, without any time delay for T steps.

It has been conjectured that if all local transition probabilities are positive then a *one-dimensional* medium is ergodic. This is the so-called *positive probability conjecture.* If this conjecture held then there would be no stable nonergodic one-dimensional medium.

The *positive probability conjecture* implies that there is no "simple" one-dimensional reliable memory. Here, simplicity means spatial uiformity and local error-correction, i.e., that the memory is a one-dimensional finite medium (with, say, wraparound at the ends). The designer is free to specify the transition rule and an error probability bound independent of the size of the device, but the actual work of each cell at each step will deviate from the rule with some probability within the bound. Suppose that we want to store one bit of information, i.e., there are two possible starting configurations, u_0 and u_1. For a memory of size K, let m_K be the maximum number of steps after which it is still possible to find out which of u_0 and u_1 was the initial content, with a probability of mistake less than 1/3. It follows from the positive probability conjecture that the value m_K is bounded by some number depending only on the error-correcting rule and the error probability but not on K. Thus if the conjecture is true then *no amount of redundancy can make a one-dimensional memory reliable.*

It is instructive to try simply error-correcting rules in a one-dimensional memory and see them fail. For definiteness, let us assume that we started with u_0 and the error probability is ρ. It seems natural to choose the K-fold repetition of 0 for u_0. The first idea for an error-correcting rule is majority voting among the three neighbors of a cell (including itself). However, this rule will not eliminate any island of 1's longer than one cell. As more and more of these islands are brought in by errors, the content of the memory will lose all similarity to u_0 in about ρ^{-2} steps. (Due to the technical complexity of the probability model, the failure of this voting rule is not really proved yet, though the methods of [5] are believed to eventually lead to its proof.)

Thus "local voting" is ruled out. The rules considered in [2] carry information from one end of a long island to the other end and thus seem to be better than local voting. But since no measures are taken to protect this information against new errors, the rules eliminate a finite set of islands only if no new errors occur.

The above reasoning suggests that the simple task of protecting a bit of information requires the capability of carrying information to large distances reliably. It is hard to imagine how to do this without setting up some structure, "division of labor" among cells: assigning roles to cells in a way varying in space and time. Tsirel'son did this (see [17]) but he sacrificed homogeneity: components of three different kinds are present, and the component kind changes in both space and time according to a grand plan not subject to errors.

Thus, if we are not willing to give up uniformity in space–time, the task of protecting one bit of information leads us to the task of setting up a non-local (e.g., hierarchical) organization and a rule that will continuously restore this organization from the damages caused by errors.

Using the above insights, Kurdyumov made in [7] some valuable suggestions for the construction of a one-dimensional stable medium. The presentation was too tentative to be taken seriously by most researchers in the field. Using Kurdyumov's ideas as a starting point, in the present paper we will show the construction of a one-dimensional stable medium that is also a reliable and (asymptotically) economical computing device. This result is a refutation of the positive probability conjecture.

I hope that after the above discussion, the reader expects a complicated construction and proof. Unfortunately, the complexity of the proof goes beyond these expectations. My main encouragement to the reader is that I think the effort to understand the proof is worthwhile. The problem it solves is simple (preserving information in a noisy environment), and some of its principles seem to be ones of a biological or social organization. If it turns out that no simpler solution exists then this proof contributes to a deeper understanding of the significance of these principles.

The present paper benefited from conversations with Leonid Levin and Georgii Kurdyumov, coauthors of [2], and Charles H. Bennett, whose work on "algorithmic depth" helps to formulate the question whether deep sequences can arise in a noisy nature. We give a table of reference of the notation that is used throughout the paper in the Appendix.

2. Media and their Perturbations

We will generally write the time and space variables as "array indices" in square brackets. To denote intervals of integers, we combine a notation from the programming language Pascal with one from real analysis. Let a, b be two real numbers. Then

$$[a \cdots b] = [a, b] \cap \mathbf{Z} = \{n \in \mathbf{Z}: a \leqslant n \leqslant b\},$$

$$[a \cdots b) = [a, b) \cap \mathbf{Z} = \{n \in \mathbf{Z}: a \leqslant n < b\},$$

etc. For a function $x[n]$ and an interval I of its domain of definition, we denote the sequence $(x[n]: n \in I)$ as $x[I]$. Similarly, for a function $x[t, i]$, we denote the sequence $(x[t, i]: m \leqslant i \leqslant n)$ by $x[t, [m \cdots n]]$.

We imagine our space-time as a plane with a left–right space axis and a downward time axis. For a point $p = (p_0, p_1)$, the time coordinate is p_0 and the space coordinate is p_1. For a space–time rectangle $[k \cdots k+h) \times [m \cdots m+n)$, we call n its *width*, and h its *height*. The set $\{k\} \times [m \cdots m+n)$ is called its *input*, and the set $\{k+h-1\} \times [m \cdots m+n)$ its *output*.

In the definitions, we will adapt [15]. A (one-dimensional homogenous deterministic) medium is a uniform chain of locally interacting automata, working in discrete time $t = 0, 1, \ldots$. We will consider both the finite case when the set of sites is the set $\mathbf{Z}_m = [0 \cdots m)$ with wraparound (0 is the right neighbor of $m-1$) and the infinite case, when the set of sites is the set $\mathbf{Z}$ of integers. We can use the notation $\mathbf{Z}_m$ in both cases with $m = \infty$ in the second one. In case of finite m, all integer operations on the sites are to be understood mod m. We will look at the history of the medium for the time steps between times $-m'$ and m' where m' can be ∞. Let $A = (-m' \cdots m') \times \mathbf{Z}_m$.

The medium is defined by the finite set $S = S_D$ of automata states and the transition function D: $S^3 \to S$ which we will also use to name the medium. Any function $x[t, n]$ over A with values in S is called an *evolution*. We will say that x is a *trajectory* of D if the relation

$$x[t+1, n] = D(x[t, n-1], x[t, n], x[t, n+1]) \tag{2.1}$$

holds for all (t, n). If (2.1) does not hold we will say that an *error* occurred in x at (t, n). Given a fixed trajectory y and an evolution x, we will say that a *deviation* occurred an t, n if $x[t, n] \neq y[t, n]$. Our goal is to limit the probability of deviations despite the fact that errors occur with some frequency.

Let ξ be a system of random variables $\xi[t, n]$ for $(t, n) \in A$ with values in S. We say that ξ is a *ρ-perturbation* of the trajectory y of the medium D if

(a) We have $\xi[t, n] = y[t, n]$ with probability 1 for all $t \leqslant 0$, n in $\mathbf{Z}_m$;

(b) For all subsets E of A the probability that an error occurs at *every* element of E is not greater than $\rho^{|E|}$.

(Here $|E|$ is the number of elements of E.) Instead of saying that errors occur independently with probability $\leqslant \rho$ we will allow a little more, and will deal with any ρ-perturbation of a medium D. The goal is to find interesting media D and trajectories y with the property that the probability of deviations is small for all ρ-perturbations of y. This notion is easy to formalize for infinite trajectories. Let us form the quantity

$$\text{maxdev}(\rho) = \sup \text{Prob}[\xi[t, n] \neq y[t, n]]$$

where the supremum is taken over all ρ-perturbations of y and all t, n. A trajectory is *stable* if we have

$$\lim_{\rho \to 0} \operatorname{maxdev}(\rho) = 0.$$

The first theorem refutes the positive probability conjecture mentioned in the introduction.

THEOREM 1. *There is a medium M with two stable trajectories y_0, y_1 over $\mathbf{Z}^2$ such that $y_0[t, n] \neq y_1[t, n]$ for all t, n.*

The proof will be given in Section II using a lemma proved later.

3. CODING AND SIMULATION

1. It was noted in the introduction that the medium M was satisfying Theorem 1 will have to compare distant pieces of information. For all this, it needs some behavior that is structured in space-time. The structure must be protected from errors. If the medium copes with such a difficult task then it is likely that it has more uses than just the storage of one bit of information. Maybe we can use it to perform an arbitrary computation reliably.

Let us therefore consider what is needed for reliable computation. By the ability for computation we mean the ability to simulate an arbitrary deterministic medium. By reliability we mean the possibility of decoding the simulated computation from the evolution of the simulating medium despite errors. For this, of course, the simulation must be an error-correcting, redundant, code. Otherwise, it loses significant information already in the first or last step. We will generalize the usual notion of a code somewhat. Let S_1 and S_0 be two state sets. A *code* γ is given by a pair P_0, P_1 of positive integers and the pair

$$\gamma = (\gamma_*(\cdot, \cdot), \gamma^*(\cdot)).$$

Here $\gamma_*(u, v)$ is the *encoding function.* The first argument u is the string in $S_1^{P_1}$ to be encoded. The second argument v is an arbitrary parameter that runs, say, over strings in $S_0^{P_1}$. The values are in $S_0^{P_0}$. In the case $P_1 = 1$ the code is called *single-letter.* We require

$$\gamma^*(\gamma_*(u, v)) = u.$$

Without the parameter v, this would be the usual definition of a code. The generalization is needed in order to combine codes conveniently. We will omit the second argument in codes where it is not used.

The quotient P_0/P_1 is the *space factor* (rate) of the code. We can extend a code f to strings whose length is a multiple of P_1 by defining

$$\gamma_*(u_1 \cdots u_m, v_1 \cdots v_m) = \gamma_*(u_1, v_1) \cdots \gamma_*(u_m, v_m).$$

The decoding function is extended correspondingly. The extension does not change the space factor.

2. Now we formulate the notion of simulation used in the present paper. Let us fix media D_0, D_1, the constants P_0, T_0, P_1, T_1. Let $\mathscr{Y}$ be a set of trajectories of D_1 with the property that the state of the middle part $[P_1 \cdots 2P_1)$ at the end of the working period $[0 \cdots T_1)$ depends only on the state of the block $[0 \cdots 3P_1)$ at the beginning of the working period. It does not depend on the border conditions, i.e. the state of the cells -1 and $3P_1$ during the working period. We express this as follows. For any trajectory y_1 in $\mathscr{Y}$ and any other trajectory y of D_1, the relation

$$y_1[0, [0 \cdots 3P_1)] = y[0, [0 \cdots 3P_1)]$$

implies

$$y_1[T_1, [P_1 \cdots 2P_1)] = y[T_1, [P_1 \cdots 2P_1)]. \tag{3.0}$$

Let γ be a code of blocks with parameters P_1, P_0 from D_1 to D_0. This code is a *simulation* of the trajectories $\mathscr{Y}$ of D_1 by D_0 with working periods T_1, T_0, if the following holds. For $i = 0, 1$, let y_i be any trajectory of D_i with $y_1 \in \mathscr{Y}$, and

$$y_0[0, [0 \cdots 3P_0)] = \gamma_*(y_1[0, [0 \cdots 3P_1)]).$$

Then we have

$$y_0[T_0, [P_0 \cdots 2P_0)] = \gamma_*(y_1[T_1, [P_1 \cdots 2P_1)]). \tag{3.1}$$

If $P_1 = T_1 = 1$ then we speak of a *single-letter simulation*, and (3.0) becomes meaningless. In this case, for any elements s_1, s_2, s_3 of S_{D_1} let y_0 be any trajectory of D_0 with

$$y[0, [0 \cdots 3P_0)] = \gamma_*(s_1)\, \gamma_*(s_2)\, \gamma_*(s_3).$$

Then (3.1) requires

$$y_0[T_0, [P_0 \cdots 2P_0)] = \gamma_*(D_1(s_1, s_2, s_3)).$$

A medium U is *universal* if for any other medium D it has a single-letter simulation that simulates all trajectories of D.

To make the reliable medium "universal" it is enough to make sure it can "simulate reliably" (in a suitable sense) a medium U which is universal in the above sense. In Section 8, we will find a universal medium. For the purpose of the following theorems, let U be an arbitrary but fixed universal medium and M a fixed other medium.

3. Let us be more specific about the form of the reliable simulation ψ of U

by M that we will be using. We introduce the notion of *concatenation* of codes. For $i=0, 1$, let ϕ_i be single-letter codes. Then the code $\phi_1 \circ \phi_0$ is defined as follows:

$$(\phi_{0_*} \circ \phi_{1_*})(u) = \phi_{0_*}(\phi_{1_*}(u)).$$

The decoding is applied, of course, in reverse order. Example: if $\phi_*(0)=000$ and $\phi_*(1)=101$ then $\phi_*(\phi_*(1))=101000101$. The code $\phi \circ \gamma$ is called the *concatenation* of ϕ and γ. The kth *iteration* of ϕ is $\phi^k = \phi \circ \cdots \circ \phi$ (k times).

4. From now on, let P and T be some integer parameters greater than 1, to be chosen later appropriately. We can restrict our attention to computations of U over a space $\mathbf{Z}_{P^r}$, over some time period of length T^s. For some arbitrarily chosen error tolerance $\varepsilon > 0$, let us define

$$k = \lceil \log(r + s - \log \varepsilon) \rceil. \tag{3.2}$$

The code ψ depends on two codes ϕ and γ. Here γ is a code mapping S_U into S_M, and ϕ is a code of S_M into S_M^P. The code ϕ is a self-simulation of M with working period T. For a distinguished element single of S_M we define

$$v = \phi_*^r(\text{single}), \qquad \psi_*(u) = \phi_*^k(\gamma_*(u, v)). \tag{3.3}$$

The string v can be viewed as a constant "software" needed for the simulation.

5. The 5-tuple $(U, M, \phi, \gamma, \text{single})$ will be defined in the following sections. We call it a *scheme*. To give a code, we also need the parameters r, s describing the size of the computation, and an error tolerance ε. Given a scheme we call a trajectory y of M over $\mathbf{Z}_m$ *legal* if we have r, k, such that with ψ defined as in (3.3) we have

$$m = P^{r+k}, \qquad m' = T^{s+k}, \qquad y[0, [0 \cdots m)] = \psi_*(u). \tag{3.4}$$

THEOREM 2. *There is a scheme* $(U, M, \phi, \gamma, \text{single})$ *and a bound* ρ *such that for any* r, s, ε, *with* k, ψ, m, m' *given by* (3.3) *and* (3.4), *for all legal trajectories* y *of* M *over* $[-m' \cdots m') \times \mathbf{Z}_m$, *all* ρ*-perturbations* ξ *of* y, *all* $h < T^s$, *the probability of the event*

$$\psi^*(\xi[hT^k, \mathbf{Z}_{P^{r+k}}]) = \psi^*(y[hT^k, \mathbf{Z}_{P^{r+k}}]) \tag{3.5}$$

is at least $1 - \varepsilon$.

The present paper is devoted to the proof of Theorem 2. The proof of Theorem 1 will come as a byproduct.

6. Is it not unnatural to assume that coding is error-free? No, because the process of encoding and decoding is used only to interpret the meaning of the computation for an outside observer. In an unreliable environment, information must

live in encoded form. Moreover, the larger amount of information we have and the more processing steps we plan to perform on it (e.g., the longer we want to keep it) the larger the space factor (redundancy) required in the code. If the output of one computation is not decoded (and the redundancy is large enough), it can be immediately used as the input of another one. It is assumed that the input and output strings include all memory space needed during the computation.

It is clear from the construction of the code ψ that computation is not hidden into the encoding. Indeed, ψ_* is essentially the iteration of a fixed self-code ϕ of M, combined with a fixed code γ of U by M. Decoding is inverse to encoding, and the code is *simple to compute.* It would take only linear time to compute our code on a serial machine, and only logarithmic time on a suitable (not cellular) parallel machine.

7. The stable scheme given in the theorem implements every computation of the ideal error-free medium U in the "physical" medium M is in such a way that the probability of deviations remains under control. The space requirement P^r of the original computation is increased to P^{r+k} in the implementation, where $k \approx \lceil \log(r+s) \rceil$. Hence the *space factor* of the implementation is P^k. Similarly, the *time factor* is T^k. Both the space and time factor and therefore of the form $\log^\beta(P^r T^s)$, for some constant β. Here $P^r T^s$ is the "size" of the original computation. It is not possible to keep even one bit of information in n cells of an unreliable medium longer than exponential time, since the n cells may form an ergodic Markov chain whose state converges this fast to a unique equilibrium state. The product of our time and space factors comes close to von Neumann's factor $\log(P^r T^s)$, which is shown in [1] to be in some sense optimal. However, the present paper answers not only the question what are the optimal time and space factors of reliable computation, but also whether reliable computation, or even just memory, is possible at all in a one-dimensional medium.

4. The Sparsity of Errors

Two constants, P and T, will play a central role in the definition of M. When M is on one of the legal trajectories, the cells will be organized into *blocks* of the self-simulation ϕ: intervals of length P. These blocks will be grouped into 2-blocks: intervals of length P^2, etc. Similarly, the trajectory divides the time axis into *working periods* of length T, and these periods are hierarchically grouped into k-periods of length T^k for all k. Let us denote by $\mathscr{P}^k[n]$ the k-block $[nP^k \cdots (n+1)P^k)$. The time period $\mathscr{T}^k[i]$ is defined similarly. We define

$$V^k[h, i] = \mathscr{T}^k[h] \times \mathscr{P}^k[i].$$

The arguments $[h]$, $[i]$, $[h, i]$ will be omitted if they are 0.

For any subset $B = [t_0 \cdots t_1) \times [n_0 \cdots n_1)$ of $\mathbf{Z}^2$ and numbers a, b, we define

$$(a, b) + B = [a + t_0 \cdots a + t_1) \times [b + n_0 \cdots b + n_1)$$
$$aB = [at_0 \cdots at_1) \times [an_0 \cdots an_1).$$

The first one fo these sets (the translation) will also be called a *copy* of B. A *k-rectangle* is a copy of V^k.

The cells in the kth order block $\mathscr{P}^k[i]$ would, under error-free conditions, perform a coordinated activity over the working period $\mathscr{T}^k[h]$. Of course, they will make errors, but they will be designed to work satisfactory as long as the set of errors in the rectangle $V^k[h, i]$ and a few of its neighbors is *k-sparse*. The notion of k-sparsity is defined recursively. It, as many other definitions later, depends on a parameter w that can be chosen at the end. All conditions on w will be lower bounds, so it only has to be chosen large enough.

First, we define the notion of a *k-window* as a set of the form $(a, b) + wV^k$. A set is 0-sparse if it is empty. A set E is *k-sparse*, if for every k-window I there is a copy J of $3wV^{k-1}$ such that $E \cap I \setminus J$ is $k-1$-sparse.

A 1-sparse set is one whose elements are far enough from each other so only a small cluster of them belongs to the same 1-window. With a two-sparse set, it may happen that more than one cluster occurs in some 1-window but such exceptions are so rare that in every 2-window, they can be covered by one 1-window blown up by a factor of three. The following lemma gives an upper bound on the probability to have a k-sparse set of errors over a certain space-time rectangle. This lemma is our only tool for estimating the error probability. Its proof does not contain any essentially new idea, therefore I recommend to skip it at the first reading.

LEMMA 4.1. *There is a constant ρ such that the following holds. Let ξ be a ρ-perturbation of a trajectory. Let B be a union of N k-windows B_i. Let p be the probability that the set of errors is not k-sparse on B. Then we have*

$$p < N\rho^{2^{k-1}+0.5}.$$

Proof. 1. The probability that we want to estimate is the sum of some probabilities of events of the form: "the set of errors is exactly H" for various sets H. By the definition of ρ-perturbation, this probability is at most $\rho^{|H|}$. Therefore we only increase our upper bound if we assume that errors occur independently and with probability ρ. For the rest of the proof of the present lemma, we make this assumption.

2. Let E be the set of errors. Notice that a set is k-sparse if and only if its intersections with every k-window are k-sparse. We prove the lemma by induction. It holds obviously for $k = 0$ and a small enough ρ (depending on w). We assume that it holds for k and prove it for $k+1$.

3. Let p_k denote the maximum possible probability that the set of errors is not k-sparse on a copy of $2wV_k$. We define four partitions $\mathscr{P}_j$ of B into copies of

$2wV_{k+1}$ as follows. For a pair $j=j_0j_1$ of binary digits, partition $\mathscr{P}_j$ consists of the intersections with B of all copies of $2wV_{k+1}$ whose left corner is a vector of the form

$$((2i_0+j_0)\,wT^{k+1}, (2i_1+j_1)\,wP^{k+1}).$$

Since each B_i is covered by at most 4 elements of $\mathscr{P}_{00}$, the size of $U_j\mathscr{P}_j$ is at most $16N$. Suppose that B contains a k-window I such that E is not $k+1$-sparse on I. Then I is contained in an element K of $U_j\mathscr{P}_j$, hence E is not $k+1$-sparse on K. Thus we have

$$p \leqslant 16Np_{k+1}. \tag{4.1}$$

4. We now estimate p_{k+1}. Let us subdivide K in the manner described above, into four partitions $\mathscr{R}_j$ consisting of copies of $2wV_k$. The number of elements in $\mathscr{R}_j$ is at most $(T+2)(P+2)$. Let U be the event that there are disjoint elements J_0, J_1 of $\mathscr{R}=U_j\mathscr{R}_j$ such that $J_i\cap E$ is not k-sparse. We prove that $p_k\leqslant \text{Prob}[U]$; especially, we prove that if U does not occur then the set of errors is $k+1$-sparse on K.

Suppose that U does not occur. Then there is an element J_1 of $\mathscr{R}$ with the property that if E is not k-sparse on an element J_0 of $\mathscr{R}$ then J_0 intersects with J_1. If J_0 intersects with, say, J_1 then it contains the center of J_1. Therefore J_0 is contained in the copy I_1 of $3wV_k$ which we obtain from J_1 by blowing it up by a factor 1.5 from its center. We prove that E is k-sparse on $K\backslash I_1$ and thus $k+1$-sparse on K. If this is not so then there is a k-window L of V_k in K such that E is not k-sparse on L. This L is contained in an element J_0 of $\mathscr{R}$ which, in its turn, must be contained in I_1.

5. We proved $p_{k+1}\leqslant \text{Prob}[U]$. To estimate $\text{Prob}[U]$ note that for each pair J_0, J_1 of disjoint elements of $\mathscr{R}$, the probability that the set of errors in not k-sparse on either J_0 or J_1 is less than p_k^2. This follows from the assumption made at the beginning of the proof that errors occur independently. The total number of possible pairs J_0, J_1 is less than $|\mathscr{R}|^2/2$. Therefore

$$\begin{aligned} p_{k+1} &\leqslant (4(T+2)(P+2)p_k)^2/2 \\ &= 8(T+2)^2(P+2)^2p_k^2. \end{aligned} \tag{4.2}$$

Using the inductive assumption on $2V_k$, we have

$$p_k \leqslant 2^2\rho^{2^{k-1}+0.5}.$$

Hence

$$p_k^2 \leqslant 2^4\rho^{0.5}\rho^{2^k+0.5}. \tag{4.3}$$

Substituting (4.2) and (4.3) into (4.1) gives

$$p \leqslant N\rho^{2^k+0.5}f(k,\rho)$$

where

$$f(k, \rho) = \rho^{0.5} 2^{11} (T+2)^2 (P+2)^2.$$

It is now obvious that for a ρ small enough we have $f(k, \rho) < 1$ for all k. ∎

5. Correcting the Information Errors

This and the next sections introduce the constructions needed for the correction of a 1-sparse set of errors. The results obtained here are not immediately applicable to the proof of Theorems 1 and 2. Therefore the exposition in these sections is less formal. However, both the constructions and the reasoning serve as building blocks for the final construction and proof.

1. Our task here is to give a simulation of the work of some trajectories of a universal medium U by some medium M_1 whose work will be changed by a 1-sparse set of errors. A working period $\mathscr{T}[h]$ of length T of a block $\mathscr{P}[i]$ of length P in the medium M_1 will simulate the working period $\mathscr{T}_0[h]$ of length T_0 of a block $\mathscr{P}_0[i]$ of length P_0 of the universal medium U. For definiteness, we fix the numbers $P_0, \ldots, T$ as follows:

$$\begin{aligned} P_0 &= 2^w, & T_0 &= 6P_0 = 6 \cdot 2^w, \\ P &= 5P_0 = 5 \cdot 2^w, & T &= (2w^2+1)(2w^3+1)\, 3^w. \end{aligned} \tag{5.1}$$

with the parameter w introduced in Section 4. These definitions set the appropriate relations of magnitude and, in the case of T, they make it divisible by $(2w^2+1)(2w^3+1)$ for a reason that becomes understandable in the next section. The trajectories z of U that we want to imitate will have the property that the outcome of a working period $\mathscr{T}_0[h]$ of a block $\mathscr{P}_0[i]$ depends only on z at time hT_0 in the three blocks $\mathscr{P}_0[i-1]$, $\mathscr{P}_0[i]$, $\mathscr{P}_0[i+1]$. Without danger of confusion, we will call such trajectories of U *legal.*

2. Let us describe, in a nutshell, how the simulation works. To fight errors, block $\mathscr{P}[i]$ holds the information about block $\mathscr{P}_0[i]$ in redundant form. This redundancy will be, in the simplest implemention, just repetition. The simulation involves the cooperation of blocks $\mathscr{P}[i-1]$, $\mathscr{P}[i]$, and $\mathscr{P}[i+1]$. First the original content of blocks $\mathscr{P}_0[i+j]$ $(j = -1, 0, 1)$ is recovered by majority voting from their redundant form in blocks $\mathscr{P}[i+j]$. Then the work of U will be simulated step for step. Then the result will be stored in $\mathscr{P}[i]$ in redundant form. To cancel the possible effect or errors occuring during the computation, the whole process is repeated two more times, and the final result is obtained by majority voting from the result of the three subperiods.

3. The program described above involves the coordinated movement of

large amounts of data through the block and precise timing. Now we give the details of this organization.

To distinguish the different sorts of information present in a cell of M_1, the cell states $x[t, n]$ are determined by a collection of *variables*. They can also be viewed as different "tracks" of a tape. Formally speaking, a variable Z is a function $Z(s)$ on the space S_{M_1}. This space is the Descartes product of the ranges of all variables. The function $Z(s)$ is the projection to the range of variable Z. We will also use the pseudovariables θ and v to indicate the current time and the position of the current cell. These variables are not available as arguments for the transition rule of the cell but make it easier to describe the rule. We will use the shorthand

$$Z[t, n] = Z(x[t, n]).$$

When it does not lead to confusion, we will simply Z^-, Z, Z^+ for $Z[\theta, v-1]$, $Z[\theta, v]$ and $Z[\theta, v+1]$, respectively. More generally, we can write

$$Z^j = Z[\theta, v+j].$$

The variable Z^-, Z^+ are, of course, arguments of the transition rule M_1.

4. In the set of variables we can distinguish two disjoint subsets: those of *information variables* and *control variables*. These notions are useful for the understanding of the principles of the work of M_1 but will not be used formally. Most variables will not belong to either one of these groups: they derive their values from information or control variables, and no special effort will be made to protect them from errors.

The *endcells* of the block $\mathscr{P} = [0 \cdots P)$ are $e^{-1} = 0$, $e^1 = P - 1$. The set $\mathscr{P}$ is divided into five subintervals $\mathscr{P}_0[0],..., \mathscr{P}_0[4]$ of length P_0. We denote

$$\mathscr{K}_i = \mathscr{P}_0[i], \qquad \mathscr{K} = \mathscr{K}_1 \cup \mathscr{K}_2 \cup \mathscr{K}_3.$$

The two control variables are τ and π. They are used by the cell to know what kind of step to perform at a certain stage. The variable $\tau[n]$, with values in $\mathscr{T}$, shows which step of the working period is now being performed by cell n, while $\pi[n]$ shows the place of cell n in its block. The program will explicitly mention only the variables whose value is changed. However, for the variable the default operation is, of course,

$$\tau[t+1, n] := \tau[t, n] + 1 \bmod T. \tag{5.2}$$

5. We will have several variables "of type U":

$$X,\ Y,\ Input_s,\ Output_i,\ Mail_j$$

for $s = 0,..., 4$, $i = 1, 2, 3$, $j = -1, 1$. These take values from the set S_U. The variable X is an *information variable*. It contains the value "represented" by the cell in the imitation. In general, it is changed only in the last step of the working period.

Ideally, all five words $X[\mathscr{K}_s]$ for $s = 0,..., 4$ are equal to the state of the same block $\mathscr{P}_0[i]$ of the medium U. Three copies would be enough but the proofs are a little easier with five copies.

During a working period of M, the working period of U will be performed three times. The result of the ith repetition will be written in the information variable $Output_i$. The variables

$$X, Output_i \quad (i = 1, 2, 3)$$

are all the information variables.

The various information transfer operations are performed with the help of the "mailbox variables" $Mail_-$, $Mail_+$ of type U. Variable Y (of type U) is used to actually imitate the computation of U.

The above organization will be successful if the control variables are able to restore themselves quickly after a local disruption, since it is these variables which show for each cell what to do at the present time with the information entrusted to it. The problem of restoration of the control variables will be addressed in the next section. For the present section, we imagine that the values of τ and π are immune to errors.

6. We describe the function M in terms of *procedures*, which are then combined at the end. To describe our procedures we will use a notation somewhat similar to the one used in programming languages like Algol or Pascal. Not all steps will be described in formal detail. Summary descriptions will occur in the programs in italics type. Comments come in parentheses, in roman type. The construct

repeat n **times**
  S

for some program statement S is a shorthand for

for $i = 1$ **to** n **do**
  S

where i is a variable not occurring in S. In listing the conditions of an "**if**" clause, we will use sometimes commas instead of "**and**" to indicate conjunction.

7. The procedure *Readin'* reads in the information found in the variable X in the current block $\mathscr{P} + v - \pi$ and the two neighbor blocks. Since in these blocks, all information is repeated five times, we write the five supposedly identical subblocks

$$X[\mathscr{K}_s + v - \pi - P] \qquad \text{for} \quad s = 0,..., 4$$

of the left neighbor block into the corresponding variables $\text{Input}_s\ [\mathscr{K}_1 + v - \pi]$ of the $\mathscr{K}_1$ part of the current block, and proceed similarly with the other two blocks.

Finally, a majority vote decides among the five pieces of information read in over each other. We use the apostrophe in the name *Readin'* since the final denifition of *Readin* will be slightly different:

procedure *Readin'*;
begin
  for $s \in [0 \cdots 4]$, $j = -1, 0, 1$ **do**
    $Input_s[\mathscr{K}_{j+2} + v - \pi] := X[\mathscr{K}_s + v - \pi + jP]$;
  $Y :=$ Maj($Input_0$,..., $Input_4$);
  end

The operation $\text{Input}_s\ [\mathscr{K}_{j+2} + v - \pi] := X[\mathscr{K}_s + v - \pi + jP]$ is a global information transfer operation. For block $v - \pi = 8P$, $j = -1$, $s = 3$, it results in writing the content of the X variables of the $\mathscr{K}_3$ part of $\mathscr{P}[7]$ into the $Input_3$ variables of the $\mathscr{K}_1$ part of $\mathscr{P}[8]$. This happens through many individual steps, by the work of cells who, of course, do not know the number $v - \pi$ of the beginning cell of the block. However, they know their relative place π in their block, and know the number τ of the program step being performed. For the transfer operation they use the *mailboxes* $Mail_-$, $Mail_+$. In most of the steps, a cell would set

$$Mail_j := Mail_j^j \tag{5.3}$$

for $j = \pm 1$. In words: "copy the contents of the left mailbox of your left neighbor into your own left mailbox. Proceed similarly with the right mailbox." However, at certain instants determined by the more detailed program of the above transfer operation (which we omit since it is obvious but boring), a cell will set $Mail_j := X$ or $Input_s := Mail_j$ for the appropriate j and s.

8. The next procedure, *Core*, performs the actual imitation. In it, we pretend that the three blocks $Y[\mathscr{K}_s]$ for $s = 1, 2, 3$ are just three consecutive blocks of length $\mathscr{P}_0$ of the medium U and iterate on them, T_0 times, the transition rules of U. The result will be found in $Y[\mathscr{K}_2]$. This procedure seems to need the states of the cells surrounding the imitated interval of length $3P_0$ of U. However, let us remember, from the remark made after (5.1), that we want to imitate only those trajectories of U in which the outcome does not depend on these boundary values. Therefore we agree that whenever *Core* needs a boundary value it uses some distinguished element 0 of S_U.

9. The procedure *Readout*(i) sends the result in five identical copies into $Output_i[\mathscr{K}_s + v - \pi]$ for $s \in [0 \cdots 4]$:

procedure *Readout*(i);
begin
  for $s \in [0 \cdots 4]$ **do**
    $Output_i[\mathscr{K}_s + v - \pi] := Y[\mathscr{K}_2 + v - \pi]$;
  end

Here is the whole program. No confusion arises if the name of the program is the same as the name of the medium whose program it is.

procedure $M_1(T)$;
begin
  for $i = 1, 2, 3$ **do begin**
    idle P **steps**; (To separate the three thirds from each other in time.)
    Readin';
    Core;
    Readout(i);
    end;
  idle steps to make the length of M_1 equal to T
  $X :=$ Maj(*Output*(1), *Output*(2), *Output*(3));
end

10. This ends the construction of medium M_1. It gives rise to a code μ of U by M_1. The definition of the encoding functions μ_{1*} and the decoding function μ_1^* is obvious. For a string x in $S^P_{M_1}$, the string $z = \mu_1^*(x)$ in $S^{P_0}_U$ is defined as follows: For all n in $\mathscr{P}_0$, let

$$z[n] = \text{Maj}(X(x[n + P_0]), \ldots, X(x[n + 4P_0])).$$

In the other direction, the encoding $x = \mu_{1*}(z)$ is defined as follows. For $s \in [0 \cdots 4]$, let

$$X(x[\mathscr{K}_s]) = z.$$

For all n, let $\tau(x[n]) = 0$, $\pi(x[n]) = n$. The values of all other variables of type U are set to the default value 0 for each n in $x[n]$. This defines the string x.

11. The following theorem will not be used itself, but it motivates the construction above. The reasoning in the proof will be used. Notice the strong assumption (5.4). It will be eliminated in the next section.

THEOREM 4. *For an integer R, let z be a legal trajectory of U over $\mathbf{Z} \times \mathbf{Z}_{RP_0}$. Let y be a trajectory of M_1 over $\mathbf{Z} \times \mathbf{Z}_{RP}$ such that the relation*

$$y[hT, \mathbf{Z}_{RP}] = \mu_{1*}(z[hT_0, \mathbf{Z}_{RP_0}])$$

holds for $h = 0$. (Then, of course, it holds for all h). Let x be a 1-trajectory of M_1 with $x = y$ at time 0. Suppose further that we have

$$\begin{aligned} \tau(x[t, n]) &\equiv t \bmod T, \\ \pi(x[t, n]) &\equiv n \bmod P, \end{aligned} \tag{5.4}$$

for all t, n. Then for all $h \geq 0$ and all intervals I of length $0.5wP$ there is an interval J of length $3w$ such that x does not deviate from y on $I \backslash J$ at time hT.

Proof. We will prove the theorem by induction on h. It is true for $h = 0$. Let us suppose that it holds for h, we prove it for $h+1$. We will look at time instant $(h+1)\,T$, and a space segment I of size $0.5wP$. We have to find a space segment J_0 of size $3w$ such that $x = y$ on the set $I \backslash J_0$ at this time. Let I_1 be the space segment that we get by enlarging I symmetrically to size wP. (We can assume that w is divisible by 4.) We define the rectangle

$$C_1 = [hT \cdots (h+1)\,T] \times I_1.$$

Then there is a rectangle $D = D' \times D''$ of size at most $3w \times 3w$ such that there are no errors in $C_1 \backslash D$. We will show that at time $(h+1)\,T$, there are no deviations in $I \backslash D''$. Let n be a point of $I \backslash D''$. We can assume, without loss of generality, that n is in the left half of I. Let I_2 be the left half of I_1. Let I_2 be the half of I_1. $i = \lfloor n/P \rfloor$,

$$C_2 = [hT \cdots (h+1)\,T) \times [(i-1)\,P \cdots (i+1)\,P).$$

Then n is at a distance of at least $0.25wP$ from the ends of I_2. For a large enough w, the interval $[(i-1)\,P \cdots (i+1)\,P)$ is contained in I_2. By the inductive assumption, there is an interval J of length $3w$ such that at time hT, there are no deviations in $I_2 \backslash J$.

Notice that for any time t in $(hT \cdots (h+1)\,T)$, there is no deviation in the X variable outside $J \cup D''$. Indeed, if no error happens then the X variable does not change before the end of the working period.

Let us look at the three identicals parts of the computation happening in the rectangle C_2. In parts $k = 1, 2, 3$, the result of the computation is sent to the variables $Output_k$. Each part begins with P idling steps. The rectangle D can intersect with the non-idling steps of ony one of the three parts. Let us show that in the other two parts k, the variables $Output_k$ will hold the values they are supposed to hold in the trajectory y (we will say that there is no *deviation* in their values).

After idling, each part begins with the *Readin'* procedure which tries to filter out deviations by majority voting. If this operation is successful then there will be no deviations in the Y values computed by *Readin'*, and the rest of the computation of $Output_k$ is error-free. The procedure *Readin'* will filter out the deviations, since they do not affect more than two of the five copies of any of the X values stored in any of the blocks $\mathscr{P}[i-1]$, $\mathscr{P}[i]$, $\mathscr{P}[i+1]$. Indeed, the five copies are at a distance P_0 from each other. The set of deviations is confined to the intervals J and D'' which have length at most $3w$.

We proved that in two of the three parts k of the computation in C_2, the $Output_k$ variables will contain no deviations. The final voting will therefore compute the correct new value for X from them everywhere where the cells do not make an error in the final vote, i.e., everywhere outside D''. ∎

6. Correcting a 1-Sparse Set of Errors

In this section, we want to eliminate assumption (5.4) from Theorem 3. If the values of the variables τ, π can be damaged by the errors then these values must be restored, since the proper functioning of the cells depends on the correctness of these values.

1. For the task of restoration, a new value *Dead* is allowed as the value of the variable X. If $X = Dead$ then we will say that the state x is *dead*. On the other hand, the statement $X \in S_U$ is equivalent to saying that x is *live*. If the state is dead the program will never look at the values of any variables different from X.

Notice that some of the procedures defined in this section have apostrophes in their name. The reason is that we will have to change them later to get the versions suitable for the proof of Theorems 1 and 2.

We introduce the function *consis*'(x_0, x_1) that will be 1 if x_0 and x_1 are consistent as the states of two neighbor cells. It is expressed in terms of the variables $X_0 = X(x_0)$, $\tau_0 = \tau(x_0)$, etc. We have *consis*'$(x_0, x_1) = 1$ if the following holds:

$$X_0, X_1 \in S_U, \quad \tau_0 = \tau_1, \quad \pi_1 \equiv \pi_0 + 1 \bmod P.$$

Otherwise, *consis*'$(x_0, x_1) = 0$. It will be more convenient to work with the function *Cons*'(j) for the evolution x defined as *consis*'(x^-, x) for $j = -1$ and *consis*'(x, x^+) for $j = 1$. To avoid the long word "inconsistency," let us call a pair n, $n+1$ a *break* at time t in the evolution x if

$$consis'(x[t, n], x[t, n+1]) = 0$$

or equivalently, *Cons*'$(-1)[n+1, t] = 0$. By the definition, a dead cell is inconsistent with any live cell. Obviously, at any time t, the set $\mathbf{Z}_m$ will be broken into connected groups of cells consistent with each other.

2. Two additional procedures are needed. The procedure *Purge*' kills all connected groups of consistent cells if they are too short. The procedure *Heal*' resurrects a small group of dead cells. The control variables of the resurrected cells will be set in accordance with their live neighbors. Both of these procedures will be executed many times during the program; *Purge*' more frequently than *Heal*'.

We begin the description of these procedures with the auxiliary procedure *Conform*' which makes a cell consistent with its left or right neighbor. A cell n will use *Conform*'(j) when it is dead. Therefore there is a little uncertainty about when to use it since usually, the cells use their program counter τ to determine the time of various actions. Let us agree that the dead cell uses *Conform*'(j) if

1. The program calls *Conform*'(j), as shown by the program counter variable τ in the *neighbor cell* $n + j$, and

2. Either the other neighbor cell is dead or it has the same τ value.

```
procedure Conform'(j, neut);
begin
  if (
  X = Dead, X^j ∈ S_U,
  (τ^{-j} = τ^j or X^{-j} = Dead).
  )
  then begin
    τ := τ^j + 1;
    π := π^j − j mod P;
    All other variables get their default values:
    end
  end
```

3. The goal of the procedure *Purge'* is to erase an interval of w^2 undesirable cells. We make use of a variable *Cn*. In the first part, from any break, a message $Cn = 0$ propagates to the right. In the second part, cells will be killed repeatedly if they contain the message and have an inconsistent right neighbor.

The details of the organization, here as well as in *Heal'*, though given below, are not particularly important, since the effect of these procedures will be considered only in error-free space–time areas. However, it is important to note that *Purge'* does not affect a large homogenous group of cells, and does not resurrect dead cells.

```
procedure Purge';
begin
  Cn := 1;
  repeat w^2 times
    if (Cons'(−1) = 0 or Cn^− = 0)
      then Cn := 0;
  repeat w^2 times
    if (Cons'(1) = 0, Cn = 0)
      then X := Dead;
  end
```

The procedure *Heal'* tries to resurrect the cells in a "gap" of size w^3:

```
procedure Heal';
begin
  repeat w^3 times
    for j = −1, 1 do begin
      Conform'(j); Temp := 1;
      end
  end
```

4. We mentioned that we want to "dovetail" certain procedures into other ones. Let us introduce the following notation. If Q and R are two procedures then we get the procedure

$$Q \times R$$

by inserting an execution of the whole procedure Q after every step of R. The whole program will have the structure

$$M_2 = \mathit{Purge}' \times (\mathit{Heal}' \times M_1(3^w)).$$

The argument 3^w after M_1 refers to the definition of M_1, where the number of the final idling steps was chosen to make the length of the program T. Now the length of M_1 has to be 3^w in order to get length T after the dovetailing.

5. This completes the definition of the medium M_2. The corresponding code μ_2 differs from μ_1 only in that we must give some default values to the new variable *Cn*, too. Let us enter the setting of Theorem 3 again. There is a trajectory z of U over $\mathbf{Z} \times \mathbf{Z}_{RP_0}$, and a trajectory y of M_2 over $\mathbf{Z} \times \mathbf{Z}_{RP}$ with

$$y[hP, \mathbf{Z}_{RP}] = \mu_{2*}(z[hP_0, \mathbf{Z}_{RP_0}])$$

holding for $h = 0$ and thus for all h. There is an evolution x of M_2 that is equal to y at time 0 and contains only a 1-sparse set of errors. *Errors* will always mean errors in x, and *deviations* mean deviations of x from y.

6. Now there are several possible values for the variables τ, π at each site and time. Cell n in time t thinks that the time origin of the computation has a remainder $t - \tau$ modulo T, and the space-origin has a remainder $n - \pi$ modulo P. Let us distinguish the cells according these beliefs. We call a pair (a, b) of numbers $a \in \mathcal{T}$, $b \in \mathcal{P}$, a space–time *origin*. For an origin (a, b), we denote by $L(a, b)$ the set of points (t, n) with

$$t - \tau(x[t, n]) \equiv a \bmod T,$$
$$n - \pi(x[t, n]) \equiv b \bmod P.$$

The sets $L(a, b)$ will be called *tribes*. By definition, they are disjoint. Only the dead cells do not belong to any of the tribes. All points where there is no deviation belong to the *canonical tribe* $L(0, 0)$. It is convenient to introduce a notation for a time-snapshot of a tribe: we write

$$L(t; a, b) = \{n: (t, n) \in L(a, b)\}.$$

We will omit the parameters (a, b) whenever they are $(0, 0)$ and this does not lead to confusion.

LEMMA 6.1. *Let t_0, n_0, n_1 be given. Suppose that*

(i) *there are no errors in the rectangle*

$$I' \times I'' = [t_0 \cdots t_0 + w^6] \times [n_0 \cdots n_1],$$

(ii) *for all t in I', both n_0 and n_1 are in $L(t; 0, 0)$,*

(iii) *for any (a, b) different from $(0, 0)$, the set $L(t_0; a, b) \cap I''$ can be covered by an interval of length $0.5w^2$,*

(iv) $n_1 - n_0 \leqslant w^3$.

Then I'' is in $L(t_0 + w^6; 0, 0)$.

This lemma says that if a small interval of space is surrounded by the canonical tribe for a sufficient number of error-free steps, and initially contains only small parts of other tribes, then it will be taken over by the canonical tribe.

Proof. 1. Let (a, b) be a space-time origin different from $(0, 0)$. We prove first that the set

$$L(t_0 + 4w^2 + 1; a, b)$$

is empty.

Each of the tribes $L(a, b)$ has its own conception about when a working period begins. What is common is that the procedure *Purge'* has length $2w^2$ and it is restarted after every step of the rest of the program. Therefore there is a time t_1 in $t_0 + [0 \cdots 2w^2 + 1]$ when cells in the set $L(t_1; a, b)$ start applying *Purge'*.

The set $L(t; a, b)$ does not contain n_0 and n_1 for any t in I'. By condition (iii), for $t = t_0$, it is contained in an interval of length $0.5w^2$. It can grow only during non-*Purge'* steps, hence will consist of intervals of the size at most $0.5w^2 + 2$ when *Purge'* is restarted. Therefore in the time-interval $[t_1 \cdots t_1 + 2w^2)$, the procedure *Purge'*, recognizing the breaks on both sides of these intervals, kills all these cells.

After this, the only procedure that would give rise to a new cell in $L(t; a, b)$ is *Conform'*. But it needs a neighbor in $L(t-1; a, b)$ for this, and no cell in $(n_0 \cdots n_1)$ has such a neighbor after *Purge'* killed all of them.

2. Let t_2 be the first step after $t_0 + 4w^2 + 1$ when the procedure *Heal'* is restarted in the canonical tribe. If *Heal'* was started just before t_0 we have to wait for it to finish. After dovetailing with *Purge'*, it takes $(2w^2 + 1)\, w^3$ steps for *Heal'* to finish. The latter is restarted after every $2w^2 + 1$ steps of the rest of the program. (This is because *Heal'* is restarted after every non-*Purge'* step, and *Purge'* of length $2w^2$ is inserted after every such step.) Therefore we have

$$t_2 \leqslant t_0 + 1 + 4w^2 + (2w^2 + 1)\, w^3 + 1 + 2w^2 = (2w^2 + 1)(w^3 + 3) + 2w^2. \quad (6.1)$$

The statement proved in 1 holds for all origins (a, b) different from $(0, 0)$. Therefore at time t_2, the interval I'' contains only cells belonging to $L(t_2; 0, 0)$ and dead cells. Let J_t be the largest interval of elements of $L(t; 0, 0)$ containing the cell n_1. Since the latter cell never dies during I', the procedure *Purge*' never kills any element of J_t (it only kills the left cell of a break). Therefore J_t never decreases. But *Heal*' keeps adding cells to the left end of J_t until it reaches n_0, before time

$$t_0 + 2w^2 + 2(2w^2 + 1)(w^3 + 3) < w^6$$

for large enough w. ∎

THEOREM 4. *If in Theorem* 3, *we replace* M_1 *by* M_2 *and* μ_1 *by* μ_2 *then it will hold even without the condition* (5.4).

Proof. We can follow the proof of Theorem 3 step for step. Let us use the notation of that proof, with

$$D' = [u_0 \cdots u_1), \qquad D'' = [v_0 \cdots v_1), \qquad J = [w_0 \cdots w_1).$$

We have to eliminate condition (5.4). We distinguish three cases:

(A) $u_0 > hT + w^6$.

(B) There is a distance greater than $2w^2$ between J and D''.

(C) None of (A) or (B).

In the second case, the cells in $I_2 \backslash (J \cup D'')$ never cease to belong to the canonical tribe since the distance between J and D'' is too large for the procedure *Purge* to cross. Therefore in the first and second cases, the conditions of (6.1) are satisfied both for $t_0 = hT$, $n_0 = v_0$ and for $t_0 = u_1$, $n_0 = w_0$. We find that the set

$$C_2 \backslash ([hT \cdots hT + w^6) \times J) \cup [u_0 \cdots u_1 + w^6) \times D'')$$

belongs to the canonical tribe.

The third case is when $u_0 \leqslant hT + w^6$, and the set $J \cup D''$ can be covered by an interval $J_1 = [v_0' \cdots v_1')$ of size $6w + 2w^2$. Therefore Lemma 6.1 can be applied with $t_0 = u_1 \leqslant hT + w^6 + 3w$ and $n_0 = v_0'$. We obtain that the set

$$C_2 \backslash [hT \cdots hT + 2w^6 + 3w) \times J_1$$

belongs to the canonical tribe. In all cases therefore, the places outside the canonical tribe are confined to a small rectangle at the beginning of the working period and possibly another small rectangle elsewhere. The small rectangle at the beginning of the working period disturbs only the idling steps before the first *Readin*'. Therefore the reasoning of the end of the proof of Theorem 3 can be applied to these somewhat larger rectangles just as it was applied there to $\{hT\} \times J$ and D. ∎

7. Restoring Partial Blocks

The program given in Section 6 does not correct a large group of errors. Let us discuss the necessary changes and additions:

(1) The procedure *Heal'* as it stands now, lets all groups of consistent cells longer than w^2 grow out of any control. Therefore we have to change *Heal'* in a way that confines it to its original purpose: the closing of gaps shorter than w^3. A new variable

Temp

will be used. The cells resurrected by *Heal'* will be marked by $Temp = 1$. After the resurrecting steps, w^3 more steps will be added in which a cell with $Temp = 1$ will be killed if it is next to a break. The final definition of *Heal* will only be given in Section 9.

(2) The procedure *Shrink* is somewhat similar to *Purge'* in that it kills cells and is a w^2-step procedure that will be applied many consecutive times. It is intended to kill a *partial block* left after the occurence of a large group of errors. It is applied enough times to kill a partial block of any size, but it works slowly (kills only one cell) in order to control the propagation of the effects of errors.

(3) The procedure *Grow* is somewhat similar to *Heal* in that it resurrects cells and is repeated several consecutive times. But it is not limited to closing small gaps: it gives a chance to a healthy block to impose its own structure on a neighbor block of dead cells. This happens by extending an "occupying arm."

The occupying arm will be partially withdrawn (in the "retreating" part of *Grow*) for the following reason. It could happen that a small group of errors takes away the end of a good block and gives it to the occupying arm of a bad block. Now the good block could be killed by its *Shrink* procedure. The retreat gives a chance to the good block to repair itself using its *Heal* procedure.

We need to be able to distinguish the cells of the occupying arm from ordinary cells. The reason is that the occupying arm will be sometimes withdrawn across block boundaries, while the cells at the end of the home block should not be killed. Therefore during applications of *Grow*, we extend the range of the variable π to

$$[-1.1P \cdots 2.1P).$$

Outside these periods, we will always have $\pi \in \mathscr{P}$.

(4) Now the procedure *Purge'* will have to watch the continuity of the occupying arm, too. This requires the redefinition of the function *Cons'*(j). We define $consis(x_0, x_1) > 1$ if in addition to all conditions of $consis'(x_0, x_1) = 1$, the following holds: If one of π_0, π_1 is outside $\mathscr{P}$ then $\pi_1 = \pi_0 + 1$ or $\pi_0 > 0.5P$, $\pi_1 < 0.5P$. The last possibility must be permitted since if the left occupying arm of a

block meets the right occupying arm of a block consistent with it we want these two arms to be able to join. More formally: $consis(x_0, x_1) = 1$ or 2 if the conditions (C1), (C2) hold.

$$X_0, X_1 \in S_U, \qquad \tau_0 = \tau_1, \qquad \pi_1 \equiv \pi_0 + 1 \mod P. \tag{C1}$$

if $\{\pi, \pi^j\} \not\subset \mathscr{P}$
then $(\pi_1 = \pi_0 + 1$ **or** $(\pi_0 > 0.5P, \pi_1 < 0.5P))$. (C2)

Otherwise, $consis(x_0, x_1) = 0$. We have $consis(x_0, x_1) = 2$ if $\pi_1 = \pi_0 + 1$. We define again $Cons(j) = consis(x^-, x)$ for $j = -1$ and $consis(x, x^+)$ for $j = 1$.

(5) The procedure *Purge* uses the breaks where $consis(x_0, x_1) < 2$.

```
procedure Purge;
begin
  Cn := 1;
  repeat w^2 times
    if (Cons(-1) < 2 or Cn^- = 0)
      then Cn := 0;
  repeat w^2 times
    if Cn = 0 and (Cons(1) < 2)
      then X := Dead;
end
```

(6) The procedure *Conform'* must also be slightly changed. Its application must have the additional condition that $\pi^j - j$ is in $[-1.1P \cdots 2.1P)$. In the definition of π, at times when τ is in $[0 \cdots s_0)$ we will not reduce mod P. The final definition of *Conform* will be given only in Section 9.

(7) We will have to place another curb on *Heal*. The application of *Conform* will have the following condition: either $\pi^j - j \in \mathscr{P}$ or $j(\pi^j - j - e^j) > 0$. This condition prevents *Heal* from extending in the "attack" direction of the procedure *Grow*, for the same reason that we introduced the "retreat" part of *Grow*.

(8) Let us combine these constructions. We call, informally, the "current group of cells" the maximum interval of consistent cells to which our cell belongs. First, the repetitions of the procedure *Grow* try to extend the current group by slightly more than P on both sides. Then, the repetitions of the procedure *Shrink* eliminate any partial block. Then follows procedure M_1. Finally, we dovetail the program with *Heal* and *Purge* just as we did to get M_2. We do not write yet out the formal details because this program still does not do what we expect from it. The set of states and the code μ_2 are the same for this new medium as they were for M_2.

The new medium does not correct healthy blocks that are either misplaced or have the wrong information content. This requires redundancy spread over a longer

range than one block. The crucial new idea (going back to Kurdyumov) is to use the universality of U and let the blocks $\mathscr{B}_0[i]$ of U perform the simulation of an error-correcting medium, e.g., M_2. We must make several modifications to our program to make this idea work. The construction will only correct a 2-sparse set of errors. But its *iteration* will lead us to the solution of the original problem.

8. A Universal Medium

For the following sections, we need a somewhat more specific definition of the simulations by the universal medium U that we want to consider.

The literature contains examples of universal media with a very small number of states. I propose the following medium, which is not minimal but is easy to program and simulate. Let $z = \langle a, b, c \rangle$ be a "pairing" operation over the natural numbers, with $a = \langle z \rangle_0$, $b = \langle z \rangle_1$, $c = \langle z \rangle_2$ denoting the inverses. For a string $x = x_1 \cdots x_n$ we will write

$$\langle x \rangle_0 = \langle x_1 \rangle_0 \cdots \langle x_n \rangle_0.$$

For natural numbers $p, x, y, z < b$, let $\mathscr{U}_b(p, x, y, z)$ be the output of some universal Turing machine $\mathscr{U}$ after b steps of computation, with program p and arguments x, y, z. We define

$$U_b(x, y, z) = \mathscr{U}_b(\langle y \rangle_0, x, y, z).$$

Thus a cell of the medium U_b computing its new state treats the first part of its present state as a program, and applies it to the states of its three neighbor cells (including itself).

The medium U_b is obviously universal for a sufficiently large b (in fact, a fairly small one). Here is the outline of a simulation of an arbitrary medium D by U_b. Each cell of D is represented by a block of consecutive U_b-cells delimited by markers. A block divides into a segment of length $O(\log|D|)$ to store the current state of the D-cell, a working area of the same length, and a segment of length $O(|D|^3 \log|D|)$ for the transition table of D. During the simulation period, first the states x, y, z represented by the three neighbor blocks are read into the working area, then $D(x, y, z)$ is looked up in the transition table and stored as the new value represented by the group. It is clear that for a suitable b independent of D, we can write a program for $\mathscr{U}_b$ to control all these operations. Let us thus choose a constant b for which U_b is universal and write $U = U_b$.

Medium U does not have to carry out the simulation in the way outlined in the previous paragraph. In fact, the simulation described in the next section works differently: it uses the capability of the universal machine $\mathscr{U}$ to execute any program for a Turing machine in the usual way.

Let us standardize the simulations we will consider. We can agree that the set of states of any simulated medium D is the set $\{0, 1, ..., |D|-1\}$ of integers. We denote by $\beta(n)$ the binary representation of the nonnegative integer n. Let P_0 be a positive integer such that $\log_2|D| < P_0$. For any string q of numbers $<b$ whose length is less than P_0-2, we define the simulation

$$\mathrm{Sim}(q, P_0) = \delta$$

as follows. The blocklength is P_0. For any state s of D, the word $x = \delta_*(s)$ is given as follows:

$$\langle x\rangle_0 = 0q20\cdots 0, \qquad \langle x\rangle_1 = 0\beta(s)\,20\cdots 0, \qquad \langle x\rangle_2 = 0\cdots 0$$

where $0\cdots 0$ always denotes as many 0's as necessary to pad the string to length P_0. The inverse δ^* is defined as follows: if there is an s with $\delta_*(s) = x$ then we define $\delta^*(x) = s$, otherwise $\delta^*(x) = 0$. The string q can be considered the "program" of the simulation. It is now easy to prove that for any medium D there is a string q and natural number P_0 such that $\mathrm{Sim}(q, P_0)$ simulates D with blocklength P_0 and work period $6P_0$.

9. Self-simulation

This section completes the definition of the error-correcting medium M. We indicated at the end of Section 7 that in order to introduce error-correction on the block level, we will use the blocks $\mathscr{P}_0[i]$ of the universal medium U to simulate some other medium. For the time being, it is best to think of this medium as the medium M_2. If $P_0 = 2^w$ is large enough then the blocks $\mathscr{P}_0[i]$ of U can be used for a simulation of the cells of the error-correcting medium M_2.

This construction is mildly circular since the parameter w was used in the definition of M_2. However, we need only a sufficiently large blocksize and period for the simulation of the cells of M_2. These cells contain variables whose maximal range is T. Since $T < 4^w$, such variables are easily describable by a word of size $2w$, and hence the description of a cell of M_2 fits comfortably into a word of size 2^w of U. We will have to handle more substantial circularity later, therefore let us not specify yet exactly what medium we are going to simulate by U. The blocks of U perform some simulation $\mathrm{Sim}(q, 2^w)$. With an appropriate choice q_2 of q, the medium M_2 is simulated, but we keep our options open. The medium we are about to define is denoted by $M_{q,w}$. Our starting point for $M_{q,w}$ is the program defined by the end of Section 7.

For $q = q_2$ the simulation

$$\delta = \mathrm{Sim}(q, 2^w)$$

is a simulation of M_2 by U. (Remember (3.1).) Therefore

$$\phi_2 = \mu_2 \circ \delta$$

is a simulation of M_2 by U. (Remember (3.1).) Therefore

$$\mu_3 = \phi_2 \circ \mu_2$$

is a simulation of U by $M_{q,w}$ composed of a code of U by M_2 by $M_{q,w}$. With this nested contruction, we hope that there is error-correction not only on the cell level but also on the block level.

The hope is not yet justified. The restoration of the information content by the simulation is possible only after *Shrink and Grow* restored the structure in the damaged blocks so that they can perform the simulation. But even this is not enough. We have to make sure that whenever the control variables are correct the simulation will be performed, independently of the values of the other variables. As it stands now, the program of the medium $M_{q,w}$ does not depend on q. If U as imitated by $M_{q,w}$ happens to perform, with its blocks $\mathscr{P}_0[i]$, the simulation $\mathrm{Sim}(q, P_0)$, then we are lucky. But the medium $M_{q,w}$ does nothing to enforce this behavior. Now we make the necessary changes to the program for this enforcement. These changes make the program dependent on q. After them, the medium $M_{q,w}$ does not simply imitate U. It imitates U in the latter's activity $\mathrm{Sim}(q, 2^w)$. For completeness, in the next paragaphs we collect all information concerning the definition of $M_{q,w}$:

(M1) The default updating steps for τ and π are given by (5.2), together with $\pi := \pi \bmod P$ whenever τ is not in $[0 \cdots s_0)$.

(M2) The condition $Cons(j) = 1$ is defined according to (C1) and (C2) of Section 7.

(M3) In the operation *Readin*, there must be a way for a block to recognize if the block from which it reads is defective. This requires some changes with respect to Section 5. Let us introduce a *new value* for all variables of type U, namely the value *Dead*. A cell whose $Mail_j$ variable has value *Dead* is not necessarily dead: it only carries a message about some defective neighbor. On the other hand, by definition, if $X = Dead$ then the state x is dead. We modify the step (5.3) as follows:

if $Cons(j) = 1$
  then $Mail_j := Mail_j^j$ (9.1)
  else $Mail_j := Dead$;

We make another, less important change in connection with *Readin*': It turns out that one does not need all five input variables, only $Input_i$ for $i = 1, 2, 3$. The leftmost and the rightmost fifth of the block will rather be used as buffer zones.

procedure *Readin*;
begin
  for $s = 1, 2, 3, j = -1, 0, 1$ **do**
    $Input_s[\mathscr{K}_{j+2} + v - \pi] := X[\mathscr{K}_s + v - \pi + jP]$;
  $Y :=$ Maj($Input_1$, $Input_2$, $Input_3$);
  end

(M4) The medium simulated by $\mathrm{Sim}(q, 2^w)$ will have the same state space as $M_{q,w}$. Especially, it will have some dead states. We distinguish one of these and denote it by *dead*. If $q = q_2$ then this is indeed one (an arbitrary one) of the states in S_{M_2} with $X = Dead$.

Now in $M_{q,w}$, before applying the rules of U to the Y variables in the procedure *Core*, the medium $M_{q,w}$ performs some *preprocessing*: If the string $Y[\mathscr{K}_s + v - \pi]$ $(s = 1, 2, 3)$ does not have the form $\delta_*(r)$ for some state r in the medium simulated by $\mathrm{Sim}(q, w)$ then it is replaced by the code of the dead state:

procedure *Prep*;
begin
  for $i :=$ 1, 2, 3 **do**
    if $Y[\mathscr{K}_s + v - \pi]$ is not a codeword of δ
      then $Y[\mathscr{K}_s + v - \pi] := \delta_*$ (*dead*);
  end

The dual of the preprocessing is the *postprocessing* after *Core* which says: if the simulation computed the code of a dead cell then the whole block should die. This is formalized by:

procedure *Postp*;
begin
  if $Y[\mathscr{K}_2 + v - \pi] = \delta_*$ (*dead*)
    then for $n \in \mathscr{K}_2 + v - \pi$ **do**
      $Y[n] := Dead$;
  end

(M5) The heart of the program is the computational procedure which we get by adding *Perp* and *Postp* to M_1. It uses the procedures *Readout* and *Core* defined in Section 5.

procedure *Comp*;
begin
  for $i = 1, 2, 3$ **do begin**
    idle P **steps**;
    Readin;

```
    Prep;
    Core;
    Postp;
    Readout(i);
    end;
  X := Maj (Output_1, Output_2, Output_3);
  end
```

(M6) The procedure *Purge* was defined in (5) of Section 7.

(M7) We define the string *neut* of length P to be the five-fold repetition of the code of the dead state, i.e.,

$$neut[\mathscr{K}_s] = \delta_* \ (dead) \qquad \text{for} \quad s \in [0 \cdots 4].$$

The final version of *Conform* will set the value of X by $X := neut[\pi]$. The reason for this choice is that when the procedure *Grow* imposes the structure of a block on a large group of dead cells then the new block has to simulate a block $\mathscr{P}_0[i]$ of the medium U which represents a dead cell of M_2. Thus the occupation by *Grow* only gives the opportunity for converting the block of dead cells into a live block if the simulated medium M_2 wants so. Otherwise, at the end of *Core*, the string $Y[\mathscr{K}_2 + v - \pi]$ contains δ_* (*dead*). The postprocessing step converts this into the string *Dead* $\cdots$ *Dead*. The latter will be copied by *Readout*(k) into the $Output_k$ variables of all cells. If this happens for at least two out of the three values of k then the voting of the last step computes $X = Dead$ and thus kills each cell of the block.

```
procedure Conform(j);
begin
  if (
    X = Dead, X^j ∈ S_U,
    (τ^{-j} = τ^j or X^{-j} = Dead),
    π^j − j ∈ [−1.1P ··· 2.1P)
    then begin
      τ := τ^j + 1;
      if τ ∈ [0 ··· s_0)
        then π := π^j − j
        else π := π^j − j mod P;
      X := neut [π mod P];
      All other variables get their default values;
      end
  end
```

The conditions of calling *Conform*(j) are the same as those of *Conform*': the cell uses the variable τ of its neighbor cells to determine when to apply *Conform*.

(M8) The procedure *Heal* is now defined according to the remarks (1) and (7) of Section 7, with the new procedure *Conform*, as follows:

```
procedure Heal;
begin
  Temp := 0;
  repeat w^3 times
    for j = −1, 1 do
      if (π^j − j ∈ 𝒫 or j(π^j − j − e^j) > 0)
        then begin
          Conform(j); Temp := 1;
          end;
  repeat w^3 times
    for j = −1, 1 do
      if (π ≠ e^j, Temp = 1, Cons(j) = 0)
        then X := Dead;
  end
```

(M9) We have to add some modifications to the procedure *Shrink*. We began describing it in (2) of Section 7.

Suppose that an error kills an endcell of an interval of consistent cells, and this endcell cannot be recovered by *Heal* because of another tribe nearby. Then *Shrink* will kill the block. If this block simulates a cell of medium M_1 then from the point of view of this simulation, the cell represented by our block died from a "microscopical" error, i.e., an error of such a small scale that we do not even want to consider. But we have to be prepared that cells will sometimes die of microscopical errors. It turns out that this can only happen to the endcell of an interval of consistent cells (remember the other tribe nearby), and only twice during a working period: before and after the one "normal-size" error permitted in the current work rectangle. Indeed, the definition of procedure *Grow* will make it sure that different tribes do not grow too close to each other by themselves.

Microscopical errors are inconvenient because they interfere with our desire to have, at the beginning of the procedure *Comp*, either almost a full block of live cells or an almost empty block. We deal with them by defining three versions of the procedure *Shrink*, i.e., it will depend on a parameter $i = 0, 1, 2$. The call *Shrink*(i) will be applied r_i times. Here, r_0 will be chosen to be so large that the τ_0 applications of *Shrink*(0) span 80% of the length T of the whole program. This choice will guarantee that "live" blocks are filled most of the time with live cells.

If a microscopical error kills an endcell during the last few applications of *Shrink*(0) then it could result in a partially killed block. To avoid this, we add

$$r_1 = 2P$$

applications of the procedure *Shrink*(1). During these steps, only those cells will be killed that are farther than $3w^3$ from the block-ends and have an inconsistent neighbor.

If a second microscopical error kills an endcell during the last few applications of

Shrink(1) and brings the end of the interval of live cells farther then $6w^3$ from the block-end then the $r_2 = 2P$ applications of *Shrink*(2) can bring the killing of the block to completion:

procedure *Shrink*(i); ($i = 0, 1, 2$)
begin
  for $j = -1, 1$ **do**
    if ($j(e^j - \pi) > 3iw^3$, $Cons(j) = 0$)
      then $X := Dead$;
  idle $w^2 - 1$ **steps**
  end

(M10) The procedure *Grow* will be defined, according to the remark (3) of Section 7, with the new procedure *Conform*, as follows:

procedure *Grow*;
begin
  (Attack)
  repeat $2w^4$ **times**
    for $j = -1, 1$ **do**
      if $\pi^j - j \notin \mathscr{P}$
        then *Conform*(j);
  (Retreat)
  repeat w^4 **times**
    for $j = -1, 1$ **do**
      if ($\pi \notin \mathscr{P}$, $j(\pi - e^j) > 0$, $Cons(j) = 0$)
        then $X := Dead$;
  end

(M11) The procedures *Grow* and *Shrink* will be combined with *Comp* in the procedure *Main*. Let us remember the definition of r_0, r_1, r_2 in (M9).

procedure *Main*;
begin
  repeat $1.4P/w^4$ **times**
    Grow;
  for $i = 0, 1, 2$ **do**
    repeat r_i **times**
      Shrink(i);
  idle steps to make the length of *Main* equal to 3^w;
  Comp;
  end

(M12) The whole program is

$$M_{q,w} = Purge \times (Heal \times Main).$$

The medium $M_{q_2,w}$will indeed correct a 2-sparse set of errors, though this is rather tedious to prove. A simple way to carry the idea further to 3-sparse sets of errors and beyond is to use a trick (Kurdyumov's idea) in the definition of $M_{q,w}$. There is a choice q_0 of q such that $\mathrm{Sim}(q_0, 2^w)$ is a simulation of the medium $M_{q_0,w}$ itself. This seems first circular but nevertheless possible.

Before showing how to do this, let us reflect on the resulting construction. Together with the medium

$$M = M_{q_0,w},$$

we also defined a code

$$\mu = \mu_{q_0,w}$$

that is a simulation of U by M.

$$\phi = \mu \circ \delta$$

is a self-simulation of M. It can give rise to an infinite hierarchy of simulations of M by itself. Let us try to convince ourselves that this structure has a chance to be self-repairing.

The crucial property of this organization is that in any gap of size P^k caused by errors, the structure of the hierarchical simulation will be spontaneously restored, provided the set of errors is k-sparse for about T^k steps. Indeed, any partial blocks are eliminated by the procedure *Shrink*. Due to the forced simulation, live (consistent) blocks simulate the behavior of cells of M (small errors are corrected by *Heal* and the voting). Thus they also apply *Shrink* and *Purge* in simulation. If a high-level cell dies then the effect of *Postp* is that it dies on all levels. In this way, after less than $2T^k$ steps, the gap does not contain "almost" any live cells that do not belong to a level k organization consistent with the one around the gap. Now the procedures *Grow* and *Heal* can refill the gap. The *Grow* procedure at a high level uses the *Grow* procedure of lower levels indirectly to overtake a region of dead cells. Indeed, a block inour program will always try to overtake a neighbor block of dead cells. If the simulation permits (e.g., because *Grow* is going on in the simulated cell) then it will keep its conquest. Buildup and destruction will thus ripple up and down recursively.

(M13) After we fixed the content of the U-computation to be a simulation of M, the reader may wonder where is our freedom to define several different trajectories. Indeed, what is the medium M computing besides simulating itself? The answer is that we indeed gave up the old way of simulating an arbitrary U-computation, but it is easy to create a new way. We introduce one more variable, *Misc*, of type U, to perform a miscellaneous computation. On *Misc*, an operation of the universal medium U will be performed, thus:

$$Misc := U(Misc^-, Misc, Misc^+).$$

In regions where there is even one error the *Misc* values are meaningless. Their usefulness is confined to the highest level where errors are improbable.

(M14) There is an even more ridiculous problem which nevertheless needs solution. At the highest level of simulation in our finite space, a group of P^l cells of M will simulate one cell of M. This one cell sees itself both on the left and the right, and thus would think it is inconsistent with its neighbor (since π is the same in the neighbor). Therefore it would kill itself, resulting in a premature death of the whole structure. To avoid this, let us introduce one more value for the variable X, the symbol *Single*. The states where $X = Single$ are inconsistent with any other state, and obey the following updating rule:

if $X = Single$
  then if $X^- = X^+ = Single$
    then $X := Single$
    else $X := Dead$

(M15) Let us show now that q, w can really be chosen so as to make $\mathrm{Sim}(q, 2^w)$ simulate $M_{q,w}$. The trick we will be using is the technique used in the proof of the so-called *recursion theorem* (fixpont theorem) of recursion theory. We will be looking for a string q which has the form $q = pp$, i.e., is the concatenation of the string p with itself.

The crucial observation is the following: There is a string q_1 such that for all p, w with

$$2^{w-1} > 4(|q_1| + |p| + w + 1), \tag{9.2}$$

the simulation

$$\mathrm{Sim}(q_1 \beta(w)\, 2p, P_0)$$

is a simulation of $M_{pp,w}$. (The digit 2 serves only for separation.) What is this string q_1? It is essentially the description of the program given above in (M1)–(M14), in the language of the universal Turing machine U_b mentioned in Section 8. The program uses pp and w as parameters. Therefore p and $\beta(w)$ have to be added to q_1 so that it can refer to them. The condition (9.2) just makes sure that the string $q_1 \beta(w)\, 2p$ fits into a half of a block $\mathscr{B}_0[i]$ of U, therefore the second half can be used to store the state of the simulated cell of $M_{q,w}$ and the computations connected with the simulation. It was shown at the beginning of this section that an area of size 2^{w-1} is sufficient for this.

For w large enough the condition (9.2) is satisfied. Let us define

$$p = q_1 \beta(w)\, 2, \qquad M = M_{pp,w}.$$

Then by the simulation $\mathrm{Sim}(pp, 2^w) = \delta$, the medium U simulates one operation of a cell of M, i.e., we have the desired string $q_0 = pp$.

The simulation $\phi = \mu \circ \delta$ is the one we need in the scheme of Theorem 2. The code γ is given as follows: Given an element x of S_M and an element s of S_U, we define $\gamma(u, x)$ by changing $Misc(x)$ to s and leaving all other variables in x unchanged. To complete the definition of the scheme we have to define the distinguished state *single* of S_M. Let this be the state with $X = Single$ where all other variables have their default values.

The following statement is an immediate consequence of our definitions.

LEMMA 9.1. *All legal trajectories of medium M connected with the scheme of Theorem 2 are live.*

10. TRIBES

The crucial part of the proof of Theorem 2 is the formalization of the reasoning presented in Section 9 about how the self-simulating medium M restores its structure. For this, we will need first a definition of the notion of higher-order *tribes*, patterned after Section 6. This notion will have to allow for a sparse set of deviations.

We want to prove that if over a segment I of space, the set of errors is k-sparse for some T^k steps then the interior of I becomes in some sense structured. We cannot assert that all space–time points in the interior will belong to the same k-tribe, only that it can be broken up into disjoint k-tribes swimming in a sea of dead cells. For $k = 0$ in Section 6, this breakup happened simply by definition. But for $k > 0$, a k-tribe has an elaborate structure. Therefore the statement that cells organize themselves into disjoint k-tribes is nontrivial. Besides this partitioning, we also want to prove that the k-tribes that arise will perform the simulation ϕ^k.

We will consider a pair (x, B), where $x[t, n]$ is an evolution over a space–time set B, and the set of errors is k-sparse on B:

1. A k-tribe will be determined by the position, "modulo V^{k+1}" (see Sect. 3), of the space–time *origin* (a, b). The pairs (a, b), (a', b') define the same k-origin if $a \equiv a' \bmod T^k$ and $b \equiv b' \bmod P^k$. The pair $(0, 0)$ is the *canonical* k-origin for each k. The definition of a k-tribe will depend on a $k+1$ origin. Therefore even of canonical k-tribes there are many (exactly TP) since there are many $k+1$-origins corresponding to the same k-origin.

For a k-origin (a, b), the rectangle $(a, b) + V^k[h, i]$ is called the *work rectangle of the* k-cell $(h, i; a, b)$, i.e., the k cell (h, i) with origin (a, b). All notions which are relative to an origin (a, b) are called *canonical* if $(a, b) = (0, 0)$. The argument (a, b) will often be omitted if it is $(0, 0)$ and this does not lead to confusion. Then we may simply speak of a canonical k-cell or a *cell*. Notice that cells are determined by a k-origin while tribes by a $k+1$-origin. Therefore a cell is completely determined by its work rectangle.

We will use the notation Γ^k to denote the union of the work rectangles of a set of k-cells. More formally, we define

$$\Gamma^k J = \bigcup_{(h,i)\in J} V^k[h, i].$$

Similarly, we write

$$\Gamma_0^k H = \bigcup_{h\in H} \mathscr{T}^k[h], \qquad \Gamma_1^k I = b + \bigcup_{i\in I} \mathscr{P}^k[i].$$

For $k>0$, we say that the k-cell $(h, i; a, b)$ is *protected* if the rectangle

$$(a, b) + \Gamma^k((h, i) + [-0.2w \cdots 0] \times [-0.2w \cdots 0.2w))$$

is contained in B. The protected k-cells are the ones whose work rectangle is "comfortably" (with a margin of width $0.2wP^k$ in space and $0.2wT^k$ in time) inside the space-time area B. It is about these cells that we can make conclusions. A k-cell $(h, i; a, b)$ is *semi-protected* if all $k-1$-cells with origin (a, b) whose work rectangle is contained in $(a, b) + V^k[h, i]$ are protected (as $k-1$-cells, which means only a margin of width, e.g., $0.2wP^{k-1}$ in space). It follows from these definitions that if a (e.g., canonical) k-cell (k, i) is protected then all canonical k-cells of the form $(h+h', i+i')$ with $|h'|, |i'| < 0.2w-1$ are semiprotected.

2. Each k-cell $(h, i; a, b)$ *represents* a value $x^k[h, i; a, b]$. As an element of S_M, this value is the collection of variable values $X(x^k[h, i; a, b])$, $\tau(x^k[h, i; a, b])$,....

A k-tribe $L^k(a, b)$ with $k+1$-origin (a, b) will be defined with the help of the set $L_k(h; a, b)$ of *live* k-cells at period h as follows:

$$L^k(a, b) = (a, b) + \Gamma^k\{(h, i): i \in L_k(h; a, b)\}.$$

The sets $L_k(h; a, b)$ will be defined recursively and simultaneously with the function x^k. We present the definition for $(a, b) = (0, 0)$. The generalization is obvious: only the argument (a, b) must be inserted everywhere.

3. For strings $x[1]\cdots x[n]$ and $y[1]\cdots y[n]$ we will say that $x \equiv y \bmod E$ if we have $x[i] = y[i]$ for all $i \notin E$. We will use the notation

$$\mathscr{P}^{(j)}[i] = iP + [3jw^3 \cdots P - 3jw^3)$$

for $j = 0, 1, 2$. Alternatively, we write $\mathscr{P}'[i]$ for $\mathscr{P}(1)[i]$ and $\mathscr{P}''[i]$ for $\mathscr{P}^{(2)}[i]$. The argument $i=0$ or $j=0$ will often be omitted.

Let $x^0 = x$. For $k>0$, suppose that x^{k-1}, L_{k-1} are defined already. We want to decide about the canonical k-cell (h, i) whether it is live, and if it is live what is $x^k[h, i]$, using only information about the status of the $k-1$-cells at the input of $V^k[h, i]$. In other words, we will look at the $k-1$-cells of the form

$$(hT, iP + n)$$

for n in $\mathscr{P}$. The reason that we look only at these P cells is that they reflect most immediately the decision made in the procedure *Comp*.

We say that the canonical k-cell (h, i) is *formatted* if there is an element σ of S_M and an interval E of length $2w^2$ such that we have for all $s = 1, 2, 3$,

$$\begin{aligned} \mathscr{P}''[i] &\subset L_{k-1}[hT] \backslash E, \\ X(x^{k-1}[hT, iP + \mathscr{K}_s]) &\equiv \delta(\sigma) \bmod E. \end{aligned} \tag{10.1}$$

We define $x^k[h, i]$ to be σ in this case, and *dead* otherwise. As can be seen from this definition, a k-cell will still be considered live if some of the $k-1$-cells among the ones that determine its state are dead or wrong. But these must be confined either to two border-intervals of length $6w^3$ or to an interval E of length $2w^2$.

4. We say that the canonical k-cell (h, i) is *proper dead* if the set $\mathscr{P}[i] \cap L_{k-1}[hT]$ can be covered by an interval of length $0.5w^2$. Let (a, b) be a k-origin different from $(0, 0)$. We say that the work rectangles of the k-cells $(h, i; 0, 0)$ and $(h', i'; a, b)$ *disturb* each other if the rectangle

$$\mathscr{T}^k[h] \times [iP^k - 8w^3P^{k-1} \cdots (i+1)\,P^k + 8w^3P^{k-1})$$

intersects with $(a, b) + V^k[h', i']$. The k-cells $(h', i'; a, b)$ and $(h, i; 0, 0)$ disturb each other if they are not proper dead and their work rectangles disturb each other.

We have $i \in L_k(h)$, if (h, i) is protected, formatted, and we have

$$\begin{aligned} \tau(x^k[h, i]) &\equiv h \bmod T, \\ \pi(x^k[h, i]) &\equiv i \bmod P. \end{aligned}$$

A canonical k-cell (h, i) is *live* if there are a, b such that

$$i \in L_k(h; aT^k, bP^k).$$

This is equivalent to the assertion that (h, i) is protected, formatted, and $x^k[h, i]$ is not dead. A canonical cell (h, i) is *proper live* if it is live, undisturbed (by any k-cell of any other tribe), and we have

$$iP + [1 \cdots P - 1) \subset L_{k-1}(hT) \cup E$$

for some interval E of length $2w^2$.

5. Let

$$m_k(h, i) = M(x^k[h, i-1], z^k[h, i], x^k[h, i+1]).$$

We say that the pair (x, B) is *organized* if the conditions (O1)–(O4) hold. Suppose that $(h, i; a, b)$, $(h+1, i; a, b)$ are protected k-cells:

(Regularity)

(O1) (i) Each protected k-cell is either proper dead or live.

(ii) If $(h, i; a, b)$ is proper dead or undisturbed live then $(h+1, i; a, b)$ is proper (live or dead).

(O2) The k-tribes $L^k(a, b)$ corresponding to different $k+1$-origins (a, b) are disjoint.

(Computation)

(O3) (i) The value $x^k[h+1, i; a, b]$ is either $m_k(h, i; a, b)$ or is dead.

(ii) If $(h, i; a, b)$ is proper live then we have the former case.

(Advance)

(O4) If $k>0$, and the cells $i-2,..., i+2$ with origin (a, b) are proper and undisturbed at h then

$$x^k[h+1, i; a, b] = m_k(h, i; a, b), \qquad x^k[h+1, i+1; a, b] = m_k(h, i+1; a, b).$$

The last condition is used to show that a k-tribe is able to "advance into the no-man's-land." Even if the cell i is dead, i.e., does not belong to the set $L_k(h; a, b)$ of live cells of origin (a, b), it will be resurrected by the cell $i-1$ if the program of the latter says so (this is expressed by m_k), and there are no other tribes nearby to block this advance (this is expressed by requiring the cells $i-1,..., i+2$ to be undisturbed).

For the next lemma, remember the definition of k-windows from Section 3.

LEMMA 10.1 *A pair (x, B) is k-organized if and only if for all possible k-windows I the pair $(x, B \cap I)$ is k-organized.*

Proof. If w is large enough then for each of the properties (O1)–(O4), any origin and any i, there is a k-window I such that the property holds for (x, B) if and only if it holds for $(x, B \cap I)$. ∎

Over a k-organized domain, we can make many assertions about the function x^k. For each h, the live k-cells (h, i) form intervals according to their tribe. New intervals do not arise out of nothing, except possibly at the left and right ends of B. The old intervals can grow, shrink, or break up. If $i-1$, i, $i+1$ are live then $x^k[h+1, i]$ depends only on $x^k[h, i+j]$ for $j=-1, 0, 1$, and is equal to $m_k(h, i)$. This follows from (O2) and (O3) and is worth noting in a lemma.

LEMMA 10.2. *Suppose that (x, B) is k-organized. If the triple $i-1$, i, $i+1$ belongs to $L_k(h)$ then we have $x^k[h+1, i] = m_k(h, i)$.*

Proof. It follows from the disjointness of the k-tribes that i is undisturbed, hence proper live. Therefore we can use (O3) to conclude the assertion of the lemma. ∎

If $(h, i-1)$ is not live then the value $x^k[h, i-1]$ is dead by definition. Therefore

k-organizedness implies that in computing $x^k[h+1, i]$, the value *Dead* will be used for this cell, i.e., cell i recognizes that is left neighbor is unformatted. (This will not be so hard since by (O1), any unformatted cell will be proper dead.)

If the middle cell i is not live but its neighbors are, then if the neighbors are in the appropriate phase of their working period they will "overtake" i, thus we will have $i \in L_k(h+1)$. A proper live cell i can die at period h only if $m_k(h, i)$ is dead. The condition (O4) asserts that an interval can increase at an endpoint if it is in the appropriate phase of the working period, and other tribes are not too close to block the growth.

LEMMA 10.3 (main lemma). *If the set of errors is k-sparse over B then the pair (x, B) is k-organized.*

The lemma is obviously true for $k=0$. Section 11 uses the main lemma to prove Theorems 1 and 2. Sections 12–16 are devoted to the inductive proof of the main lemma. Properties (O1)(i) and (O3)(i) will be proved in Section 14. Property (O2) will be proved in Section 15. Finally, properties (O1)(ii), (O3)(ii), and (O4) will be proved in Section 16.

11. PROOF OF THEOREMS 1 AND 2

Proof of Theorem 2. We use the medium M, and the codes ϕ, γ defined in Section 9. Let y be a legal trajectory with a given $r, s, k = \log(r+s-\log\varepsilon)$, $m = P^{r+k}$. Let ξ be a ρ-perturbation of y. Let p be the probability that the set of errors in ξ is not k-sparse on the whole space-time $A = (-T^{s+k} \cdots T^{s+k}) \times \mathbf{Z}_m$. By Lemma 3.1, we have

$$p \leqslant T^s P^r \rho^{2^{k-1}+0.5}.$$

Indeed, the space–time A can be covered with $T^s P^r$ k-windows. Substituting the definition of k, we get

$$p \leqslant \rho^{0.5}(T^s P^r \rho^{0.5(s+r-\log\varepsilon)}).$$

For ρ small enough, this expression is less than ε, for any r, s. Therefore the set of errors is k-sparse with probability $1-\varepsilon$.

Let us thus suppose that the set of errors in our sample realization x of ξ is k-sparse. Define $B = A$. The pair (x, B) satisfies the conditions of the main lemma. We have to prove that (3.5) holds. The main lemma implies that the pair (x, B) is k-organized and all k-cells are live at period $h=0$. If this is true for all $h \leqslant T^s$ then we are done. Indeed, in this case the evolution

$$\gamma^*(x^k[[0 \cdots T], \mathbf{Z}_{P^r}])$$

is a trajectory z of the universal medium U. Therefore (3.5) holds.

The fact that all k-cells are live for all periods $h \leqslant T^s$ can be proved by induction over h. It is true for $h=0$. Let us suppose that it is true for all h, then we prove it for $h+1$. By (O3), the evolution

$$x^k[[0 \cdots h], \mathbf{Z}_{P^r}]$$

is a legal trajectory of M. By (O3) we have $x^k[h+1, i] = m_k(h, i)$, therefore $x^k[[0 \cdots h+1], \mathbf{Z}_{P^r}]$ is an extension of the same legal trajectory. All legal trajectories are live by Lemma 11.1. ∎

The proof of Theorem 1 requires a slight redefinition of the medium M. The reason is that the states of medium M contain the variable *Misc* whose values are never restored from errors. This is not permissible (though rather irrelevant) in stable trajectories. Let us therefore alter the definition of medium M by simply deleting the variable *Misc* in the definition of $M_{q,w}$ (and finding the program q again that yields the self-simulation). We call the new medium M'. For simplicity, we use the same letters μ, δ, ϕ to denote the simulations relating to M'. The proofs of Lemmas 9.1, 10.1–10.3 remain unchanged for M'.

There seems to be some difficulty in finding the initial state of a stable trajectory y over $\mathbf{Z}$. But there is a unique choice.

LEMMA 11.1 *There is a unique trajectory y of M' over $\mathbf{Z}^2$ with the property that for all h, i, k there is a state $s \in M'$ with*

$$y[hT^k, \mathscr{P}^k[i]] = \phi_*^k(s).$$

Proof. For a cell i and an arbitrary k, let i_k be such that $i \in \mathscr{P}^k[i_k]$. The sequence $i = i_0, i_1, \ldots$ will always become all -1 or all 0 after a while. Let k_i be the first k such that $i_k = -1$ or 0.

Let us remember the definition of the simulation $\delta = \mathrm{Sim}(q, P_0)$. It can be seen from the formulas that the first and last letters of the word $\delta_*(s)$ did not depend on s at all. For any word u, the first and last letters of $\mu_*(u)$ depend only on the first and last letters of u. Therefore the first and last letters a and b of $\phi_*(s) = \mu_*(\delta_*(s))$ do not depend on s.

Let $k = k_i$. If $i_k = -1$ then

$$y[hT^k, \mathscr{P}^k[i_k]] = \phi_*^k[b],$$

otherwise it is $\phi_*^k[a]$. This defines the trajectory y uniquely. ∎

We will prove that y is a stable trajectory of M'. With this, we will be done since it is easy to find a stable trajectory y' everywhere different from y by $y'[t, i] = y[t+1, i]$. We have $y[t+1, i] \neq y[t, i]$ everywhere since the medium M' increases the value of the variable τ by 1 mod T in every step.

The proof of Theorem 1 needs a lemma that will come as a side result in the proof of the main lemma. We introduce the intervals

$$W_0^k[h, i] = (hT^k, iP^k) + [-0.4wT^k \cdots 2T^k) \times [-0.4wP^k \cdots 0.4wP^k),$$
$$W_1^k[h, i] = (hT^k, iP^k) + [0.5T^k \cdots 2T^k) \times [0 \cdots 2P^k).$$

LEMMA 11.2. *Suppose that the set of errors is $k-1$-sparse over the rectangle $W_0^k[h, i]$. Suppose that the k-cells in the interval $i + [-2 \cdots 3]$ are in $L_k[h]$ and both $m_k[h, i]$ and $m_k[h, i+1]$ are live. Then $W_1^k[h, i]$ is contained in L^{k-1}.*

The proof will be given in Section 14. This lemma says that if our input deviates in "something" like a k-sparse set form a string of the form $\phi_*^k(u)$, and the set of errors over our local space-time domain (a few copies of V^k) is not only k-sparse but also $k-1$-sparse, then the largest deviations in the input will soon be corrected.

Proof of Theorem 1. It is left to prove that the trajectory y is stable. Let ξ be a ρ-perturbation of y. Under the assumptions of the theorem, let us look at a space–time point (t, n). Let $R_0, R_1, \ldots$ be a sequence of rectangles where $R_0 = \{(t, n)\}$, while R_k for $k > 1$ has the form $W_0^k[h, i]$ with

$$R_{k-1} \subset W_1^k[h, i].$$

Such a sequence obviously exists:

1. Let U be the event that for all $k > 0$, the set of errors in ξ is $k-1$-sparse over R_k. By Lemma 3.1, for any $k > 0$, the set of errors will not be $k-1$-sparse over R_k only with probability $\rho^{2^{k-2}+0.5}$. The sum of these terms for all $k > 0$ is $O(\sqrt{\rho})$. Therefore U holds with probability $1 - O(\sqrt{\rho})$.

2. Let x be a realization of ξ for which U holds. It remains to show that $x[t, n] = y[t, n]$. Let r be the first k such that R_k intersects the start line $\{0\} \times \mathbf{Z}$. If $r = 0$, there is nothing to prove. We will show that R_k is contained in L^k for all $k < r$. Especially, $(t, n) \in L^0$, which implies $x[t, n] = y[t, n]$.

For $R_r = W_0^r[h, i]$, the conditions of Lemma 11.2 are satisfied by definition, hence also $R_{r-1} \subset W_1^r[h, i]$ is contained in L^{r-1}. Similarly, for all $k \in [1 \cdots r-1]$, if $R_k \subset L$ then $R_{k-1} \subset L^{k-1}$ by Lemma 11.2. Therefore by induction, we have $R_k \subset L^k$ for all $k < r$. ∎

12. HEALING

By induction, we assume that the main lemma holds for $k-1$. We will always assume that the parameter w is as large as needed. Throughout the remainder of the paper, we are given a pair (x, B) satisfying the conditions of the main lemma. Our aim is to prove the properties (O1)–(O4). It follows from Lemma 10.1 that we can assume that B is contained in a k-window. Since the set of errors is k-sparse over B

and B is a k-window there is a rectangle J_0 with the property that the set of errors is $k-1$-sparse in $B\backslash J_0$. We will call J_0 the *error rectangle.*

An important tool of the proof of the main lemma is an analog of Lemma 6.1 describing the healing process.

LEMMA 12.1. *Let t_0, n_0, n_1 be given. Suppose that*

(i) *the rectangle $[t_0-1\cdots t_0+w^6]\times[n_0\cdots n_1]$ consists of protected $k-1$-cells;*

(ii) *for all t in $[t_0\cdots t_0+w^6]$, both n_0 and n_1 are in $L_{k-1}(t;0,0)$;*

(iii) *for any (a,b) different from $(0,0)$, the set of cells n in $[n_0P^{k-1}\cdots n_1P^{k-1})$ such that (t_0T^{k-1}, n) belongs to $L^{k-1}(a,b)$ can be covered by an interval of length $0.5w^2P^{k-1}$;*

(iv) $n_1-n_0<w^3$.

Then there is a time $u\leqslant w^6$ such that $[n_0\cdots n_1]$ is contained in $L_{k-1}(t_0+u;0,0)$.

This lemma says that if a small interval of space is surrounded by the canonical $k-1$-tribe for a sufficient number of error-free steps, and it initially contains only small parts of other $k-1$-tribes, then it will be taken over by the canonical tribe. We cannot necessarily choose $u=w^6$ because the final step of the procedure *Comp* can kill a cell, even if there are no inconsistencies nearby. Therefore some cells in the gap $[n_0\cdots n_1]$ healed by the time t_0+u may be killed again by the time t_0+w^6.

Proof. The proof is analogous to the proof of Lemma 6.1. The main lemma implies that the pair (x, B) is $k-1$-organized:

1. Let (a,b) be a $k-1$-origin different from $(0,0)$. We prove first that for any $t>(t_0+5w^2)\,T^{k-1}$, there are no points n in $[n_0P^{k-1}\cdots(n_1+1)P^{k-1})$ with the property that $(t,n)\in L^{k-1}(a,b)$. Let

$$t_0'=\lfloor(t_0-a)/T^{k-1}\rfloor,$$
$$n_0'=\lfloor(n_0-b)/P^{k-1}\rfloor,$$
$$n_1'=n'_0+n_1-n_0.$$

Then the point (t_0T^{k-1}, n_0P^{k-1}) is contained in the working rectangle $(a,b)+V^{k-1}[t_0', n_0']$ of origin (a,b). The condition (ii) says that strips of width P^{k-1} on the left and right sides of the rectangle

$$[t_0T^{k-1}\cdots(t_0+w^6+1)\,T^{k-1})\times[n_0P^{k-1}\cdots(n_1+1)\,P^{k-1})$$

are contained in the canonical $k-1$-tribe. It follows from the main lemma that the different $k-1$-tribes do not intersect. Therefore the set $L^{k-1}(t;a,b)$ does not contain n_0' and n_1' for any t in $[t_0'+1\cdots t_0'+w^6]$.

Let t_1' be the first time of application of *Purge* in the program after t_0'. Then $t_1' \leqslant t_0' + 2w^2 + 1$. According to (O3) of the main lemma, the cells of $L_{k-1}(h; a, b)$ perform the work of the medium M, treating $k-1$-cells of origin (a, b) outside $L_{k-1}(h; a, b)$ as dead cells. The only possible deviation from this norm is that improper live cells may die. Such cells can only be at the ends of the intervals of live cells.

By condition (iii), the set $L_{k-1}(t; a, b)$ can be covered by an interval of length $0.5w^2$ at period $t = t_0'$. That the cells of this set will be killed during the time steps $t \in [t_1' \cdots t_1' + 2w^2]$ is proved exactly as in part 1 of the proof of Lemma 6.1.

2. Let t_2 be the first step after $t_0 + 5w^2$ when the procedure *Heal* is restarted. We have now an estimate similar to (6.1) again. The statement proved in 1 holds for all origins (a, b) different from $(0, 0)$. Also, we have

$$a + t_0' T^{k-1} \leqslant t_0 T^{k-1}$$

for all $k-1$-origins (a, b). Therefore the set

$$\{t_2 T^{k-1}\} \times [n_0 P^{k-1} \cdots (n_1 + 1) P^{k-1})$$

has an empty intersection with each of the noncanonical sets $L^{k-1}(a, b)$. Now the proof can again be finished just as in 2 of the proof of Lemma 6.1. This part of the proof relies also on (O4): the fact is used that a dead $k-1$-cell is resurrected if the program of its live neighbor says so and if there are no $k-1$-cells of other tribes nearby. This completes the proof of the lemma. ∎

The previous lemma motivates some definitions for taking into account the effect of the error rectangle J_0. Let q_0 and q_1 be the smallest and largest of all those n for which the strip

$$\mathbf{Z} \times \Gamma_1^{k-1}(n + [-0.2w \cdots 0.2w))$$

has a nonempty intersection with J_0. Then $q_1 - q_0 \leqslant 4w$. Similarly, let p_0 and p_1 be the smallest and largest of all t for which the strip

$$\Gamma_0^{k-1}(t + [-0.2w \cdots 0] \times \mathbf{Z}$$

intersects with J_0. Then $p_1 - p_0 \leqslant 4w$. We define

$$J = J' \times J'' = [p_0 \cdots p_1] \times [q_0 \cdots q_1].$$

The $k-1$-cells that are protected in B and do not belong to J are protected in the set $B \backslash J_0$, over which the set of errors is $k-1$-sparse. Any time interval disjoint from J' is called *error-free*. The set $E_0(t)$ is defined to be

$$[q_0 - w \cdots q_1 + w] \text{ for } t \in (p_0 + [0 \cdots 5w^2))$$

and empty otherwise. It can be easily seen from part 1 of the proof of Lemma 12.1 that if the error rectangle J occurs in a large domain of dead $k-1$-cells then the set of live cells possibly caused by J is confined to $E_0(t)$.

Let p_2 be the first multiple of T after p_0 if p_0 is in $[s_0 \cdots T)$, and let it be p_0 otherwise. The set $E_1(t)$ is defined to be

$$[q_0 - w \cdots q_1 + w] \qquad \text{for} \quad t \in (p_0 + [0 \cdots 5w^2)) \cup (p_2 + [0 \cdots w^2)),$$

and empty otherwise. If the error happens during the computation period $[s_0 \cdots T)$ then the information effect of the error may hold over to the end T of the current working period and manifest itself again. This is reflected by the second part of the union in the above formula. The set $E(t)$ is defined to be

$$[q_0 - 0.2w^3 \cdots q_1 + 0.2w^3] \qquad \text{for} \quad t \in (p_0 + [0 \cdots w^6)) \cup (p_2 + [0 \cdots w^6)),$$

and empty otherwise. It can be seen that if J occurs in a large domain of live $k-1$-cells then the set of dead cells possibly caused J is confined to $E(t)$. Indeed, besides the two non-*Purge* steps that can kill two cells on both sides, the most that can happen is that the procedure *Purge* kills w^2 cells on any side of the damage. (This can only happen if the damage is close to a block-end and t is in $[0 \cdots s_0)$ mod T.) Therefore the killing does not extend further than $0.2w^3$ steps. After this, the gap will be closed as shown in Lemma 12.1, in w^6 steps. The gap can reoccur after q_2, as reflected by the second part of the union.

13. The Integrity of Blocks

In what follows we will often use the constants

$$d_0 = (2w^2 + 1)(2w^3 + 1), \qquad d_1 = d_0 w^2.$$

The number d_0 is the multiplier by which the length of the program increases after dovetailing with *Heal* and *Purge*. The number d_1 is length of an application of *Shrink*(i) after dovetailing with *Heal* and *Purge*.

Let $G^i(t)$ denote the set of cells n in $L_{k-1}(t)$ with the property that

$$\pi(x^{k-1})[t, n] = n - iP.$$

The set $G^i(t)$ is an extension of the block of $k-1$-cells forming the k-cell i, by possibly adding "occupying arms" while t is in the time period $[0 \cdots s_0)$ of the applications of *Grow*. Let us write $G(t)$ for $G^0(t)$.

The next lemma says that if the set of $k-1$-cells in a block is almost an interval then it remains so at least until the last step of the working period. The "almost" is expressed with the help of the sets $E_1(t)$, $E(t)$ defined at the end of Section 12. For any set $H(t)$ depending on t we will use the notation

$$H(t, +) = H(t) \cup E(t), \qquad H(t, -) = H(t) \setminus E_1(t).$$

We say that the set A is an *interval* modulo a set B if either $A \backslash B$ or $A \cup B$ is an interval.

LEMMA 13.1. *Suppose that the canonical k-cell (h, i) is semiprotected. Let t_0 be an element of $hT + [s_0 \cdots T + d_0)$.*

(i) *If $G^i(t_0)$ is an interval* mod $E(t_0)$ *then $G^i(t)$ is an interval* mod $E(t)$ *for all t in $[t_0 \cdots T + d_0)$.*

(ii) *If n is in $G^i(t_0) \backslash E_0(t_0)$ then the interval $n + [-3w^3 \cdots 3w^3)$ has a nonempty intersection with $G^i(t, -)$ for all t in $hT + [s_0 \cdots t_0)$.*

Proof. (i) Without loss of generality, we can set $h = i = 0$. Only the error can break the continuity of $G(t)$. The steps affected by the error begin at time p_0 and end at time p_1, as defined at the end of the last section. The interval J'' is immediately affected. The first start of *Purge* after p_1 comes at some time $t' < p_1 + 2w^2 + 1$.

There were at most $p_1 - p_0 + 1$ steps in which new cells could by created. Therefore the set $G(t')$ consists of intervals only the leftmost and rightmost of which can be longer than

$$q_2 - q_0 + 2(p_1 - p_0) + 2 < 12w + 2.$$

Therefore all these intervals other than possibly the leftmost and rightmost one will be killed by *Purge*. If the leftmost and the rightmost intervals survive during the next application of *Heal* then the distance of their inner ends is still at most $4w + 2$. Therefore *Heal* will join them just as described in the proof of Lemma 12.1.

(ii) In the time interval $[s_0 \cdots T + d_0)$, only the procedure *Heal* and the error can add new cells to $G(t)$. The procedure *Heal* can add w^3 cells to each side. These cells will be killed again unless they reach the block-end or an error occurs. The error can add $4w$ more cells. With another application of *Heal* we obtain the bound $2w^3 + 4w$ on the distance to which the set $G(t, -)$ can extend on each side. ∎

We expect the procedure *Shrink* to guarantee that during *Comp*, the set $G(t)$ is either almost empty (corresponding to a proper dead cell) or almost covers the working area $\mathscr{K}$ of the computation.

For $j = 0, 1, 2$, let s_j denote the beginning of the first application of *Shrink*(j). Note that s_0 was introduced already in Section 7. Let s_0 denote the beginning of *Comp*.

For some h, i, the rest of the section deals with the work of the $k-1$-cells of the block $\mathscr{P}[i]$ in their periods t for t in $\mathscr{T}[h]$. We will suppose that these $k-1$-cells are protected during this time. Without loss of generality, we can assume $i = 0$.

We will call a *gap* in $\mathscr{P}$ at time t any contiguous interval of $\mathscr{P} \backslash G(t)$. The part of space–time outside the error rectangle J_0 is $k-1$-sparse, hence we can apply the inductive assumption there. It follows from (O3) and the main lemma that in this

region, a cell n of $G(t)$ dies only if either its program says so (i.e., $m_{k-1}[t, n]$ is dead) or it is an improper live cell at the edge of a gap.

No program step can create a gap. Indeed, the cause of killing a cell in any step but the last one of the working period, is always some inconsistency. But the cells of $G(t)$ are consistent with each other. Therefore a gap arises in an error-free step only if one of the endcells is an improper live cell and dies (see the discussion after the definition of k-organizedness in Sect. 10).

No error-free step before T can make a gap narrower without closing it. The only steps narrowing a gap are those of *Heal*. But in an error-free application of *Heal*, all newly created cells will be repealed if they did not close a gap. Hence outside the time interval J' the gaps can not contract if they do not disappear. Therefore we will say that a gap *persists* during a time interval if it does so in the steps of the time interval not belonging to *Heal*. An application of *Shrink*(j) extends a gap if the gap intersects $\mathscr{P}^{(j)}$. Let us call such steps *extending* for the gap.

The next lemma says that if a gap persists too long then it kills the block. The set $E_1(t)$ was defined at the end of Section 12.

LEMMA 13.2. *Suppose that the canonical k-cell* $(0, 0)$ *is semiprotected. Suppose that for some* $t' < s_c$, *during times* $t < t'$, *a gap in* $G(t)$ *has size* l *or persists through* l *consecutive extending steps.*

(i) *If* $l \geqslant 2.5w^3$ *and it contains an endcell then the gap will never be closed completely before time* $T + s_0$.

(ii) *If* $l \geqslant 7.5w^3$ *and* $t' < s_c - 1.1Pd_1$ *then* $G(t)$ *is contained in* $E_0(t)$ *for all* t *in* $[t' + 1.1Pd_1 \cdots T + s_0)$.

Proof. (i) Suppose that the gap contains the endcell 0. The effect of the error rectangle, whether it occurs during the l steps or afterward, can decrease a gap by at most $4w + w^3$ without closing it. (Changing the middle of a gap just when *Heal* was trying to close the right end.) The time of the error could cover at most one extending step. Therefore if a gap persisted through the l consecutive extending steps then at the end, even after a possible error, its size is at least $l - w^3 - 4w$. Therefore it cannot be closed by *Heal* and has a nonempty intersection with $\mathscr{P}''$.

(ii) Before time T, the gap will always be at least $l - 1.2w^3$ cells wide. Therefore all shrinking steps are extending steps for it. Among the $1.1d_1P$ extending steps after t' there are enough error-free ones to widen the gap over the whole $\mathscr{P}$. After this, all cells of $G(t) \cap \mathscr{P}$ until $t = T + s_0$ will be due to the error rectangle. They will be eliminated by *Purge* at the earliest possibility, and therefore will be enclosed into $E_0(t)$. ∎

Let us define

$$c_i = 8w^3 d_i \qquad \text{for} \quad i = 0, 1.$$

The following lemma is an immediate consequence of Lemmas 13.1 and 13.2.

LEMMA 13.3. *Suppose that the canonical k-cell* $(0, 0)$ *is semiprotected. Then one of the following cases holds*:

(A) *We have* $G(t) \subset E_0(t)$ *for all* t *in* $[s_1 \cdots T + s_0)$.

(B) *The set* $G(t)$ *is an interval* mod $E(t)$ *for all* t *in* $[s_0 + c_1 \cdots T)$.

LEMMA 13.4. *Suppose that the canonical k-cell* $(0, 0)$ *is semiprotected. If case* (A) *of Lemma* 13.3 *does not hold then* $G(t, +)$ *contains* $\mathscr{P}''$ *for all* t *in* $[s_0 + c_1 \cdots T)$. *It contains* $\mathscr{P}$ *for all but possibly* c_1 *steps* t *in* $[s_0 + c_1 \cdots s_1)$.

Proof. We suppose that case (A) of Lemma 13.3 does not hold. Therefore $G(t)$ is an interval mod $E(t)$ for all t in $[s_0 + c_1 \cdots T)$. From now on, we will look at the leftmost element $l(t)$ of the set $G(t, -)$. Reasoning about the right end is analogous.

Let t_1 be the first time step t before the error interval J' with the property that for some i we have $t < s_i$, $l(t) > 3iw^3$. Let t_2 be the first time-step after J' with the same property. We can assume without loss of generality that both t_1 and t_2 exist:

1. If $l(s_{i+1}) \leqslant 3iw^3$ for some i then there is no t_j after s_{i+1}. Indeed, the procedure *Shrink*$(i+1)$ does not extend the gaps in $[0 \cdots 3(i+1)w^3)$. If there is no t_j before s_1 then $l(s_1) = 0$ and the condition is fulfilled. Suppose therefore $t_1 < s_1$.

2. Suppose that t_j is in $[0 \cdots s_1 - 4.5w^3 d_1]$ for some j. By Lemma 13.2 if the left endgap persisted until $t_j + 2.5w^3 d_1$ then it would never be healed. By the time s_1 it would be extended to a size of at least $3w^3$, even if the error decreases it maximally. After that, *Shrink*(1) would extend it over the whole block. Since this does not happen, we have $l(t'_j) = 0$ for some $t'_j < t_j + 2.5w^3 d_1$ and $l(t) < 2.5w^3$ for all t in $[t_j \cdots t'_j]$.

3. If the assumption 2 holds for $j = 1, 2$ then it follows from 1 that the statement of the lemma is proved. Suppose therefore that the assumption 2 does not hold for $j = 1$. Then we have $l(t) < 5w^3$ for all t in $[s_0 + c_1 \cdots s_1]$.

We will show that there is a t'_1 in $[s_1 \cdots s_1 + 2.5w^3 d_1]$ such that $l(t'_1) \leqslant 3w^3$, and $l(t) < 6w^3$ for all t in $[s_1 \cdots t'_1]$.

If $l(s_1) \leqslant 3w_3$ then we can choose $t'_1 = s_1$. If $l(s_1) > 3w^3$ then by Lemma 13.2, if the condition $l(t) > 3w^3$ persists until $s_1 + 2.5w^3 d_1$ then it will never be repaired and it kills the block. Therefore we will have $l(t'_1) \leqslant 3w^3$ for some $t'_1 < s_1 + 2.5w^3 d_1$ and $l(t) < 6w^3$ for all t in $[s_1 \cdots t'_1]$.

4. If $t_2 < t'_1$ then condition 1 is fulfilled for $i = 1$. If $t_2 > t'_1$ then the extension of the left endgap is error-free after t_2, and will not kill the block only if we have $l(t) < 6w^3$ for all t in $[t_2 \cdots s_2]$.

5. If the assumption 2 holds for $j = 1$ but does not hold for $j = 2$ then the argument in 4 can be repeated again, now with the t'_1 defined in 2. ∎

We can summarize the results of Lemmas 13.1–13.4 as follows:

LEMMA 13.5. *Suppose that the canonical k-cell* (h, i) *is semiprotected. Then one of the following cases holds*:

(A) *We have* $G^i(t) \subset E_0(t)$ *for all* t *in* $hT + [s_c - w^6 \cdots T + s_0)$. *Especially,* (h, i) *is proper dead.*

(B) *For all* t *in* $hT + [s_0 + c_1 \cdots T)$ *the set* $G^i(t)$ *is an interval* mod $E(t)$. *The set* $G^i(t, +)$ *contains* $\mathscr{P}[i]$ *for all but* c_1 *values of* t *in* $hT + [s_0 + c_1 \cdots s_1]$. *It contains* $\mathscr{P}''[i]$ *for all* t *in* $hT + [s_0 + c_1 \cdots T)$.

The next lemma strengthens the conclusion of Lemma 13.5 in the case when two adjacent inner blocks are live. It says that in this case, the area joining them is also live.

LEMMA 13.6. *Suppose that the canonical* k*-cells* $n, n+1$ *are semiprotected in period* h, *and we have case* (B) *of Lemma* 13.5 *with both* $i = n$ and $n+1$. *Then for all* t *in* $hT + [s_0 + c_1 \cdots T)$, *the set* $G^n(t) \cup G^{n+1}(t)$ *is an interval* mod $E(t)$, *and we have*

$$G^n(t) \cup G^{n+1}(t) \cup E(t) \supset [nP + 6w^3 \cdots (n+2)P - 6w^3).$$

Proof. The proof of Lemma 13.5 can be literally repeated for the interval $[nP \cdots (n+2)P)$ as a whole. ∎

The next lemma says that if a block was filled by the procedure *Grow* then it represents a dead k-cell.

LEMMA 13.7. *Suppose that the canonical* k*-cell* i *is semiprotected in the periods* g *and* $g+1$. *Then Lemma* 13.5 *applies with both* $h = g$ *and* $h = g+1$. *Suppose that case* (A) *holds with* $h = g$ *and case* (B) *holds with* $h = g+1$. *Then for all* t *in* $(g+1)T + [s_2 \cdots T)$ *and for all* n *in* $\mathscr{P}'' \setminus [q_0 \cdots q_1]$ *we have* $X(x^{k-1}[t, n]) = neut[n]$, *where* $neut[n]$ *was defined in Section* 9.

Proof. Without loss of generality, we can assume $g = i = 0$. Since case (A) holds for $h = 0$, all $k-1$-cells but possibly the ones in $E_0(T-1)$ are dead at time $T-1$. Even the ones in $E_0(T-1)$ will die in $2w^2 + 1$ steps in case they are live. The operations resurrecting the cell use *Conform*, and hence write $neut[n]$ into the X of cell n. Until time $2T$, no program step changes the X variable of live cells. Since case (B) holds for $h = 1$, each $k-1$-cell in $\mathscr{P}''$ except those immediately affected by the error is a resurrected one during $T + [s_2 \cdots T]$. ∎

14. COMPUTATION

The next lemma prepares Lemma 14.2.

LEMMA 14.1. *Suppose the canonical* k*-cells in the set*

$$[h-1 \cdots h+1] \times [i-1 \cdots i+1]$$

are semiprotected. Then either case (A) *or case* (B) *below holds*:

(A) $G^i(t, -)$ *is empty for all* t *in* $(h+1)\,T+[0\cdots s_0)$. *Especially, the* k*-cell* $(i, h+1)$ *is proper dead.*

(B) $G^i(t, +)$ *contains* $\mathscr{P}''[i]$ *for all* t *in* $hT+[s_0+c_1\cdots T+s_0)$. *Moreover, we have*

$$X(x^{k-1}[(h+1)\,T, iP+\mathscr{K}_s]=u$$

with the same string u *in* $S_U^{P_0}$ for $s=1, 2, 3$.

Proof. Without loss of generality, we can assume $h=i=0$. We can apply Lemma 13.5 with $h=0$ to any of the blocks $\mathscr{P}[j]$ for $j=-1, 0, 1$. Let Λ be the set of numbers j such that case (B) of this lemma holds for j. If j is not in Λ then the k-cell $(0, j)$ is proper dead. Therefore if $0\notin\Lambda$ then (A) holds and we are done. Assume $0\in\Lambda$. It follows from Lemma 13.6 that for any t in $[s_0+c_1\cdots T)$ the set

$$G'(t)=\bigcup_{j\in\Lambda} G^j(t)$$

is an interval mod $E(t)$, and for any j in Λ, it contains $\mathscr{P}''[j]\backslash E(t)$:

1. First we suppose that the error rectangle J does not intersect the rectangle

$$[s_c-w^6\cdots T]\times\mathscr{P}.$$

Then for all t in $[s_c-w^6\cdots T)$, the set $G'(t)$ is an interval that contains $\mathscr{P}''$. This interval can only decrease, by at most one cell on each end, if an improper endcell dies. The three identical computational parts of the program in its application to the values x^{k-1} of the $k-1$-cells in $\mathscr{K}\subset G'(t)$ during steps t in $[s_c\cdots T]$ are undisturbed by any error. Thus in the ith part, five exact copies of some string u_i are written to

$$Output_i(x^{k-1}[t, \mathscr{K}_s\cap G'(t)])$$

for $s=0,..., 4$. If any element of u_i is the value *Dead* then all of its elements are *Dead*, since an errorless computation produces such outputs (see *Postp*).

(We cannot claim any relation among the three sequences u_1, u_2, u_3. Indeed, the information is not necessarily in redundant form in the neighbor blocks from which it is read in. Therefore the change caused by an error in these blocks cannot always be filtered out by the voting in the procedure *Readin*.) The final step of the program is a cell-for-cell vote among the three strings

$$Output_i(x^{k-1}[T-1, \mathscr{P}\cap G'(i)])$$

for $i=1, 2, 3$. The result of this vote may be a string which does not "code" anything, but its five fifths will be equal, and it is either all dead or all live. In the former case, the k-cell 0 is proper dead, in the latter case, it has the form required in (B) of the present lemma.

2. Now we suppose a little more: that the error rectangle does not intersect the rectangle

$$R = [s_c - w^6 \cdots T] \times [-P \cdots 2P).$$

Then all conclusions of case 1 remain in force. But additionally, the strings u_i will be equal to the same string $\delta_*(s)$ for some $s \in S_M$ for all three values of i. Indeed, now the input of the ith computation will be the same for each i. Thus, in this case the k-cell $(1, 0)$ will be formatted.

3. Let us suppose now that the error rectangle intersects R. Then the reasoning of case 1 can be applied to the work of each of the k-cells $-1, 0, 1$ at the period -1. Hence each such k-cell j is either consists of dead $k-1$-cells at time T or has the property that

$$X(x^{k-1}[T, jP + \mathscr{K}_s]) = v_j \tag{14.1}$$

for some string v_j independent of $s = 1, 2, 3$.

If the second case holds then j has the form (14.1) over $\mathscr{P}[j]$ at time $s_c - w^6$ too, since no error changed it yet. In the first case, it may happen that not all $k-1$-cells are dead in $\mathscr{P}[j]$ at time s_c. But this can only happen if the interval $\mathrm{P}[j]$ was overtaken during $[0 \cdots s_0)$ by the procedure *Grow* of some neighbor blocks. Moreover, it was overtaken completely, since it was not killed during the time $[s_0 \cdots s_1)$ of the application of *Shrink*(1). Therefore by Lemma 13.7 we have

$$X(x^{k-1}[s_c - w^6, jP + \mathscr{K}_s]) = \delta_*(Dead)$$

for $s = 1, 2, 3$.

4. Having the desired input to the computational part of the program, it is not difficult to see that it gives the desired output. Indeed, the error rectangle will be separated in time from at least two of the three identical parts of the computation. The error may change or kill at most $4w$ cells of the input interval $G'(s_c - w^6)$. If these cells are near the edge of the input interval then they may kill some cells permanently but the affected cells will not belong to any of the $jP + \mathscr{K}$ for $j \in \Lambda$. If the affected cells include some in a $jP + \mathscr{K}$ with $j \in \Lambda$ then these cells are deeply inside $G'(t)$. Then Lemma 12.1 implies that they will be resurrected in at most w^6 steps. Of course, the $X(x^{k-1})$ values in the error-affected interval may be lost. The $Output_i(x^{k-1})$ in the part of the computation affected by the error is probably worthless.

In the two error-free parts i of the program, the input comes from three neighbor blocks. The neighbors with case (A) of Lemma 13.5 consist of dead $k-1$-cells in this time, hence the sign *Dead* will be transmitted from them. The triply redundant information in the other ones may have been changed by the error in a short interval, but this effect will be filtered out by the input voting. Therefore everywhere but

in the cells immediately affected by the error (i.e., belonging to $E_0(t)$ for some t), the *Output* values computed will be equal to the string u repeated five times where

$$u = \delta_*(M(x_{-1}, x_0, x_1))$$

where x_j is dead if $j \notin \Lambda$, and is $\delta^*(v_j)$ otherwise. Also, if $M(x_{-1}, x_0, x_1)$ is dead then, due to *Postp*, all elements of u are *Dead.* ▮

The next lemma implies properties (O1)(i), (O3)(i). Let $L'_k(h)$ denote the set of k-cells for which all conditions of belonging to $L_k(h)$ are satisfied with the exception that, instead of being protected, they are possibly only semiprotected.

LEMMA 14.2. *Suppose that the canonical k-cells in the set*

$$[h-2 \cdots h+1] \times [i-2 \cdots i+3]$$

are semiprotected. Then either case (A) *or case* (B) *below holds*:

(A) *The k-cell* $(h+1, i)$ *is proper dead. Moreover,* $G^i(t, -)$ *is empty for all* t *in* $(h+1)\,T + [0 \cdots s_0)$.

(B) *We have* $i \in L'_k(h+1)$, $x^k[h+1, i] = m_k[h, i]$. *Moreover,* $G^i(t, +)$ *contains* $\mathscr{P}''[i]$ *for all* t *in* $hT + [s_0 + c_1 \cdots T + s_0)$.

If we have the case (B) *for* $(h+1, i+1)$ *too then for the same values of* t, *the set* $L^{k-1}(t, +)$ *contains* $[-P + 6w^3 \cdots 2P - 6w^3)$.

Proof. Without loss of generality, we can assume $h = i = 0$. We can apply Lemma 14.2 to the pairs $(h, i) = (-1, -1)$, $(-1, 0)$, and $(-1, 1)$. In this way, we obtain conditions for the computation of the string $x^{k-1}[(h+1)\,T, \mathscr{P}]$ that allow the same reasoning as the one applied to the computation of the string $x^{k-1}[(h+1)\,T, \mathscr{P}]$ in part 4 of the proof of Lemma 14.1. Therefore the conclusion is the same. ▮

Proof of Lemma 11.2. It follows from the conditions of the lemma and Lemma 14.2 that $L_{k-1}(0, +)$ contains the interval

$$iP + [-2P + 6w^3 \cdots 4P - 6w^3).$$

Since the set of errors is $k-1$-sparse over $W_0^k[h, i]$, the evolution is $k-1$-organized. The possible gap in $E(hT)$ will be quickly filled using Lemma 12.1. After this, *Purge* and *Shrink* will have no more cause to kill any live $k-1$-cells in the blocks $\mathscr{P}[j]$ for j in $[i-1 \cdots i+2]$. These perform the computation as described in part 4 of the proof of Lemma 14.1. Since the values $m_k[h, i]$ and $m_k[h, i+1]$ are live both blocks $\mathscr{P}[i]$ and $\mathscr{P}[i+1]$ remain in $L_{k-1}[t]$ at $t = (h+1)\,T$. They remain live at least until $(h+2)\,T$. ▮

15. The Disjointness of k-tribes

The next two lemmas extend the validity of the conclusion of Lemma 13.1.

Lemma 15.1. *Suppose that the k-cells in the set* $[h-2\cdots h+1]\times[i-3\cdots i+3]$ *are semiprotected. Then* $G^j(t)$ *is an interval* mod $E(t)$ *for all* t *in* $hT+[s_0+c_1\cdots T+s_0)$.

Proof. As always, we can suppose $h=i=0$. It follows from the application of Lemma 13.5 to $(h, i)=(0, 0)$ that $G(t)$ is an interval mod $E(t)$ for all t in $[c_1\cdots T)$. If the case (A) of Lemma 13.5 holds then $G(t)$ is contained in $E_1(t)$ for all t in $T+[0\cdots s_0)$, and the k-cell $(0, 0)$ is proper dead. Suppose therefore that case (B) of Lemma 13.5 holds. Then we can apply Lemma 14.2 with $(h, i)=(0, 0)$. If case (A) of this lemma holds then $G(t)\subset E_1(t)$ for all t in $T+[0\cdots s_0)$. Suppose therefore that case (B) of lemma 14.2 holds. Then $G(t)$ is an interval mod $E(t)$ for $t=T$. Indeed, this is true for $t=T-1$. It follows from Lemma 14.2 that the last step of the program will produce a formatted result, and will not kill any cell outside $E(t)$. An improper endcell may die, changing an end of $G(T-1)$ by at most one cell but leaving it an interval.

It remains to prove that $G(t)$ is an interval mod $E(t)$ for t in $T+(0\cdots s_0)$. This is the time of applications of the procedure *Grow*. The analysis given in the proof of Lemma 13.1 applies here without change. Indeed, error-free steps of *Grow* do not create gaps. They do not extend gaps faster than the steps in *Shrink* either, therefore the gaps that are created by errors during *Grow* will be closed just as the ones during *Shrink*. ∎

Let s_r denote the first retreating step in the last application of *Grow* in the program.

Lemma 15.2. *Suppose that the k-cells in* $[h-2\cdots h+1]\times[i-4\cdots i+4]$ *are semiprotected and the k-cell* $(h+1, i)$ *is not proper dead. Then*

(i) *the set* $L_{k-1}(t, +)$ *contains* $\mathscr{P}[i]$ *for all but* c_0 *values of* t *in* $[s_r+c_0\cdots s_0)$;

(ii) *it contains* $\mathscr{P}''[i]$ *for all* t *in* $[s_r+c_0\cdots T+s_0)$.

Proof. (i) Without loss of generality, we can assume $h=i=0$. The set $L_{k-1}(t)\cap\mathscr{P}$ is the union of the sets $G_j(t)\cap\mathscr{P}$ for j in $[-2, 2]$, since the occupying arms of the two left and two right neighbor blocks can reach into $\mathscr{P}$. It follows from Lemma 15.1 with $h=0$ that $G(0)$ is an interval mod $E(0)$. The sets $G^j(0)\cap\mathscr{P}$ for $j\neq 0$ are empty by definition.

Lemma 15.1 implies that in the time interval $[0\cdots s_0)$, each of the sets $G^j(t)$ is an interval mod $E(t)$. Moreover, it is clear that those with $j<0$ cover left segments of $\mathscr{P}$ and those with $j>0$ cover right segments of $\mathscr{P}$. Since these sets are also disjoint

by definition, the set $\mathscr{P} \cap G^j(t, -)$ can be nonempty for only one of the numbers $j = -1, -2$ and only one of the numbers 1, 2. For $j = \pm 1$, let us denote by

$$G_j(t)$$

the one (if any) among the sets $G^j(t)$ and $G^{2j}(t)$ for which the above set is nonempty.

The set $G_{-1}(t) \cup G(t) \cup G_1(t)$ is the union of three intervals mod $E(t)$. The retreating part of *Grow* will extend the gaps between these intervals in every step that does not belong to *Heal* or *Purge*. Therefore the reasoning of Lemma 13.2 shows that the ones among these gaps with both edges in $\mathscr{P}$ at time $s_r + c_0$ have a size that will kill the k-cell $(1, 0)$ in the time interval $[s_0 \cdots s_1)$ of applications of *Shrink*(0). Since by our assumption the k-cell $(1, 0)$ is not proper dead, there are no such gaps. Therefore the set

$$\mathscr{P} \cap (G_{-1}(t) \cup G(t) \cup G_1(t))$$

is an interval mod $E(t)$ at time $t = s_r + c_0$. Since the endgaps will not kill the block $\mathscr{P}$ either, the set $L_{k-1}(t, +)$ contains $\mathscr{P}$ for all but c_0 values of t in $[s_r + c_0 \cdots s_0)$.

(ii) Now that we know that no cells are killed outside $E_1(t)$ at $t = T$, the reasoning of the proof of Lemma 13.4 can be applied without change to the whole interval $[s_r \cdots T + s_0)$. ∎

LEMMA 15.3. *Under the notation of the previous lemma, suppose that the k-cells in* $[h-2 \cdots h+1] \times [i-4 \cdots i+4]$ *are semiprotected, the k-cell* (h, i) *is dead and for some* t' *in*

$$[s_r + 0.7w^4 d_0 \cdots T + d_0)$$

the $k-1$*-cell* n *is in*

$$L_{k-1}(t') \cap \mathscr{P}[i] \setminus E_0(t).$$

Then $L_{k-1}(t, -)$ *has a nonempty intersection with the interval* $n + [-3w^3 \cdots 3w^3)$ *for all* t *in* $[s_r + c_0 \cdots t')$. *The set* $L_{k-1}(t, +)$ *contains the interval*

$$[n - 0.3w^4 \cdots n + 0.3w^4)$$

for all t *in* $[s_r + c_0 \cdots s_r + 0.3w^4 d_0)$.

Proof. As always, we can assume $h = i = 0$ without loss of generality. The part of the statement concerning in $[s_c \cdots T + d_0)$ follows from (ii) of Lemma 13.1. The proof of this lemma used only the fact that the program in the given time interval creates new cells only by *Heal*. Concerning $L_{k-1}(t)$ instead of $G(t)$, these properties hold for the whole time interval $[s_r \cdots T + d_0)$. Therefore the first statement of the present lemma holds. It follows that there is a $k-1$-cell n' in

$$n + [-3w^3 \cdots 3w^3) \cap L_{k-1}(s_0 - 1, -).$$

Since the k-cell $(0, 0)$ is dead, we have case (A) of Lemma 14.2, and hence $G(t) \subset E_1(t)$ for all t in $[0 \cdots s_0)$. Therefore for $s_0 - 1$ we have

$$L_{k-1}(t, -) = G_{-1}(t, -) \cup G_1(t, -).$$

Without loss of generality, suppose that n' is in $G_{-1}(s_0 - 1)$. Suppose that the gap between $G_{-1}(t)$ and $G_1(t)$ disappears before $t = s_r + c_0$. Then by (ii) of the previous lemma, the sets $G_j(t, +)$ contain $\mathscr{P}''[j]$. Therefore $[-P + 6w^3 \cdots 2P - 6w^3)$ is contained in $L_{k-1}(t, +)$ for all t in $[s_r + c_0 \cdots s_0)$, and we are done.

Suppose now that a gap between $G_{-1}(t)$ and $G_1(t)$ persists for t in $[0 \cdots s_r + c_0)$. Then it will never be closed. The remaining at least $0.7w^4 - c_0$ steps of *Grow* pull back the right end of $G_{-1}(t, -)$ by at least $0.7w^4 - c_0 - 2w^3$. It follows from the previous lemma that $G_{-1}(t, +)$ contains $\mathscr{P}''[-1]$ for all t in $[s_r \cdots s_0)$. Since $G_{-1}(s_0 - 1, -)$ contains n', the whole interval

$$[-P + 6w^3 \cdots n' + 0.7w^4 - c_0 - 2w^3)$$

is contained in $G_{-1}(t, +)$ for $t = s_r + c_0$. This interval can decrease by at most $0.3w^4 + 3w^3$ during the $0.3w^4 d_0$ steps considered in the second statement of the lemma. ∎

The following lemma says that the procedure *Grow* never brings k-tribes of different $k-1$-origins in disturbing closeness to each other.

LEMMA 15.4. *Suppose that the canonical k-cells in the set $[h-3 \cdots h+2] \times [i-5 \cdots i+5]$ are semiprotected, i is in $L'_k(h+1)$ and the canonical k-cell $(h+1, i)$ is disturbed. Then i is in $L'_k(h)$ and the canonical k-cell (h, i) is disturbed.*

Proof. It follows from the assumption of the lemma that for some k-origin different from $(0, 0)$, the work rectangles of the k-cells $(h+1, i; 0, 0)$ and $(h'+1, i'; a, b)$ disturb each other, as defined in Section 10, and the k-cell $(h'+1, i'; a, b)$ is not proper dead. It follows from Lemma 14.2 that i' is in $L_{k-1}(h'+1; a, b)$.

We can suppose without loss of generality that $h = i = 0$. Then defining

$$a' = a + h'T^k, \qquad b' = b + i'P^k,$$

we have $(a, b) + V^k[h', i'] = (a', b') + V^k[0, 0]$. We can suppose without loss of generality that

$$a' \in \mathscr{T}^k, \qquad b' \in [0 \cdots P^k + 8w^3 P^{k-1}). \tag{15.1}$$

By induction, we know that the sets $L^{k-1}(-)$ and $L^{k-1}(-; a, b)$ are disjoint.

Applying Lemmas 15.1–15.2 with $h = i = 0$ we find that the set $L_{k-1}(t, +)$ contains $\mathscr{P}''$ for all t in $T + [s_r + c_0 \cdots T + s_0)$. This means

$$|\Gamma_1^{k-1}(L_{k-1}(t, -) \cap \mathscr{P})| > P^k - 13w^3 P^{k-1} \tag{15.2}$$

for all t in $\Gamma_0^{k-1}[(s_r + c_0) \cdots T + s_0)$. Similarly,

$$|b' + \Gamma_1^{k-1}(L_{k-1}(t', -; a, b) \cap \mathscr{P})| > P^k - 13w^3P^{k-1} \tag{15.3}$$

for all t in $a' + \Gamma_0^{k-1}[h'T + (s_r + c_0) \cdots T + s_0)$. Notice that the interval of the values of t for which (15.2) holds has a nonempty intersection with the interval of values for which (15.3) holds. Namely, it follows from (15.1) that they both contain the subinterval

$$a' + \Gamma_0^{k-1}[(s_r + c_0) \cdots s_0). \tag{15.4}$$

Therefore it follows from the disjointness of $L^{k-1}(-)$ and $L^{k-1}(-; a, b)$ that the intersection of the inervals $\mathscr{P}^k$ and $b' + \mathscr{P}^k$ is shorter then $13w^3P^{k-1}$. In other words,

$$b' \in [P^k - 13w^3P^{k-1} \cdots P^k + 8w^3P^{k-1}). \tag{15.5}$$

We generalize now some earlier notations. E.g., $G^i(t; a, b)$ denotes the equivalent of $G^i(t)$ for origin (a, b). Of course, all earlier lemmas apply when we take an origin different from the canonical one:

1. Suppose that 0 is not in $L'_k(0; a, b)$. Let $t' = h'T + s_r$. It follows then from Lemma 15.3 that $L_{k-1}(t', +; a, b)$ contains

$$i'P + [-0.3w^4 \cdots P + 0.3w^4)$$

for all t in $[t' + c_0 \cdots t' + 0.3w^4d_0)$. Therefore for t in

$$a' + \Gamma_0^{k-1}(s_r + [c_0 \cdots 0.3w^4d_0))$$

the number of elements n in

$$b' + \Gamma_1^{k-1}[-0.3w^4 \cdots P + 0.3w^4)$$

such that (n, t) is in $L^{k-1}(-; a, b)$ is greater then

$$P^k + 0.6w^4 - w^3P^{k-1}.$$

But together with (15.2), this contradicts the disjointness of $k-1$-tribes $L^{k-1}(-)$ and $L^{k-1}(-; a, b)$.

2. Let us suppose now that $i' \in L'_k(h'; a, b)$ but $i \notin L'_k(h)$. After exchanging left for right, the relation between k-cells (h, i) and $(h', i'; a, b)$ is the same as the relation was, in part 1 of the proof, between the k-cells $(h', i'; a, b)$ and $(h + 1, i)$. Therefore part 1 of the proof applies. ∎

The property (O2) is proved in the following lemma.

LEMMA 15.5. *Any two different k-tribes are disjoint. Moreover, if the $k+1$-origin (a, b) is different from $(0, 0)$ then L^k is disjoint from $L'^k(a, b)$.*

Proof. Suppose that the second statement does not hold. If (a, b) is equivalent to $(0, 0)$ as a k-origin then the k-origins $(0, 0)$ and (a, b) differ only in the values $\tau(x^k[h, i])$ and $\pi(x^k[h, i])$ which are defined for any live cell (h, i). This makes $L^k(0, 0)$ and $L'^k(a, b)$ disjoint by definition. Suppose therefore that the k-origin (a, b) is different from $(0, 0)$.

We can assume without loss of generality that L^k and $L'^k(a, b)$ intersect in such a way that $0 \in L_k(2)$, $i \in L'_k(h' + 2; a, b)$ and the rectangle $(a, b) + V^k[h' + 2, i']$ intersects with $V^k[2, 0]$. We will arrive at a contradiction from this assumption. Let us define a' and b' as in the proof of Lemma 15.3. Then we can assume, due to the symmetry of left and right and the symmetrical roles of L^k and $L'^k(a, b)$ that $a' \in \mathscr{T}$, $b' \in \mathscr{P}$, $(a', b') \neq (0, 0)$. The roles do not seem symmetrical since one of them is L^k and the other one is L'^k. But what is important is only that *any* one of the cells $(0, 0)$ and $(h', i'; a, b)$ be protected, in order to imply all the conditions of earlier lemmas on semiprotectedness.

It follows from Lemma 15.4 that 0, 1 are in $L'_k(0)$ and i', $i' + 1$ are in $L'_k(h')$. Now in absolute time, the working period $a' + \mathscr{T}^k$ of the k-cell $(h', i'; a, b)$ is contained in the union $[0 \cdots 2T^k)$ of the working periods 0 and 1 of the canonical k-cell 0. These cells are not proper dead in these working periods. In absolute space, they occupy the intervals $\mathscr{P}^k$ and $b' + P^k$ which have a nonempty intersection. Let us show that this is impossible since the prolonged contract would kill one of the k-cells.

By the definition of the constant r_0 in Section 9, the interval $[s_0 \cdots s_1)$ of application of *Shrink*(0) in cell $(h', i'; a, b)$ is longer than $0.8T$. The interval

$$a' + \Gamma_0^{k-1}[s_0 \cdots s_1)$$

is contained in the union of the intervals $\mathscr{T}^k[0]$ and $\mathscr{T}^k[1]$, therefore it has an intersection I_0 longer than $0.4T^k$ with one of them, say $\mathscr{T}^k(0)$. The intersection I_1 of I_0 with the interval $\Gamma_0^{k-1}[s_0 \cdots s_1)$ of applications of *Shrink*(0) in the canonical cell $(0, 0)$ is longer than $0.2T^k$. Now it follows from case (B) of Lemma 13.5 that for all but c_1 elements t of the time interval I_1, we have

$$\{t\} \times \mathscr{P}^k \subset L^{k-1}(-)$$

and

$$\{t\} \times (b' + \mathscr{P}^k) \subset L^{k-1}(-, a, b).$$

This is a contradiction since the left sides of these equations have a nonempty intersection and the right sides are disjoint. ∎

16. Proper Cells

The first lemma of this section shows that if the canonical k-cell $(0, 0)$ is protected and undisturbed then the canonical $k-1$-cells "belonging" to it can be distur-

bed only by the $k-1$-cells of other tribes which are in one of their attacking periods. This is useful to know since attacks are always followed by retreats.

LEMMA 16.1. *Suppose that the canonical k-cell* $(0, 0)$ *is protected and proper. Suppose that for some* t_0 *in* $\mathscr{T}[1]$, n_0 *in* $G(t_0)\backslash E_0(t_0)$, n_1 *in* $[n_0 \cdots n_0 + 8w^3)$, *the work rectangle of the canonical* $k-1$*-cell* (t_0, n_1) *is intersected by the work rectangle of the* $k-1$*-cell* n' *in*

$$L_{k-1}(t'; a, b)\backslash E_0(t'; a, b).$$

Let $h' = \lfloor t'/T \rfloor - 1$. *Then either* t_0 *is in* $T + [d_0 \cdots s_r + 0.7w^4 d_0)$ *or* t' *is in* $(h'+1)\,T + [d_0 \cdots s_r + 0.7w^4 d_0)$.

Proof. 1. Let $i' = \lfloor n'/P \rfloor$. It follows from the conditions of the lemma that the work rectangle of the k-cell $(h', i'; a, b)$ disturbs the work rectangle of the k-cell $(0, 0)$. Therefore both of these k-cells cannot be live, since if $(0, 0)$ is live then it is proper live. This is the only place where we use that the k-cell $(0, 0)$ is dead or proper live. Therefore from now on, the roles of the k-cells $(0, 0)$ and $(h', i'; a, b)$ are symmetric. We can suppose without loss of generality that $(h', i'; a, b)$ is dead. Let h'_0 be $h'+1$ if the k-cell $(h'+1, i'; a, b)$ is also dead, and h' otherwise.

2. Suppose first that the k-cell $(0, 0)$ is also dead. Let h_0 be 1 if the k-cell $(1, 0)$ is dead and 0 otherwise. Now the role of the points $h_0 T^k$ and $a + h'_0 T^k$ is symmetric. We can suppose without loss of generality that

$$h_0 T^k \leqslant a + h'_0 T^k. \tag{16.1}$$

It follows from the consecutive application of Lemmas 13.1(ii), 15.2(ii), and 15.3 that for all t in

$$H = [h_0 T + s_r + c_0 \cdots t_0),$$

the interval $I = n_0 + [-12w^3 \cdots 12w^3)$ has a nonempty intersection with $L_{k-1}(t, -)$. It follows from Lemma 15.3 that for all t in

$$H' = h'_0 T + s_r + [c_0 \cdots 0.3w^4 d_0),$$

the interval $I' = n' + [-0.2w^4 \cdots 0.2w^4)$ is contained in $L_{k-1}(t, +; a, b)$. By (16.1), the interval $a + \Gamma_0^{k-1} H'$ is essentially (to within a difference T^{k-1} at its right end) contained in $\Gamma_0^{k-1} H$. By its definition, the interval $b + \Gamma_1^{k-1} I'$ contains $\Gamma_1^{k-1} I$. Therefore the rectangles $\Gamma^{k-1}(H \times I)$ and $(a, b) + \Gamma^{k-1}(H' \times I')$ have an intersection with a time projection of length greater then

$$0.2w^4 d_0 T^{k-1} > 0.2w^9 T^{k-1}.$$

The second rectangle is filled by $L^{k-1}(+; a, b)$. For each time t in its time-projec-

tion, the first rectangle has a nonempty intersection with $\{t\} \times \mathbf{Z}_m \cap L^{k-1}(-)$. Since for all but $w^6 T^{k-1}$ values of t we have

$$\{t\} \times \mathbf{Z}_m \cap L^{k-1}(-; a, b) = \{t\} \times \mathbf{Z}_m \cap L^{k-1}(+; a, b),$$

this contradicts the disjointness of $k-1$-tribes.

3. Suppose now that the k-cell $(0, 0)$ is live. Then again, it follows from Lemmas 13.1(ii) and 15.2(ii) that for all t in

$$H_0 = [-T + s_r + c_0 \cdots t_0],$$

the interval I has a nonempty intersection with $L_{k-1}(t, -)$. We have $a + (h'+1)\, T^k \geqslant 0$ by the definition of h'. Therefore

$$a + h'_0 T^k \geqslant a + h' T^k \geqslant -T^k,$$

$$a + h'_0 T_k + (s_r + c_0)\, T^{k-1} \geqslant -T^k + (s_r + c_0)\, T^{k-1}.$$

It follows that $a + \Gamma_0^{k-1} H'$ is essentially (to within a difference T^{k-1} at its right end) contained in $\Gamma_0^{k-1} H_0$, hence the argument of the end of 2 applies. ∎

LEMMA 16.2. *Suppose that the (canonical) k-cell* $(0, 0)$ *is protected, and either proper dead or undisturbed, and that the k-cell* $(1, 0)$ *is live. Let* t_0 *be an element of* $T + [s_0 \cdots s_c)$ *with* $\mathscr{P} \subset G(t_0, +)$. *Suppose that some* $k-1$*-cell in*

$$[0.5P \cdots P + 7w^3 - 1) \backslash E(t_0)$$

is disturbed in period t_0. *Then there is a* t_1 *in* $[0 \cdots 2.4w^4 d_0)$ *such that for all* t *in*

$$t_0 + t_1 + [0 \cdots 0.1w^4 d_0)$$

the $k-1$*-cells in*

$$[0.5P \cdots P + 5w^3) \backslash E(t)$$

are undisturbed.

Proof.

1. It follows from the conditions of the lemma that for some n_0 in

$$G(t_0, -) \cap [P - w^3 \cdots P)$$

and n_1 in $n_0 + [0 \cdots 8w^3)$, some origin (a, b) different from $(0, 0)$, some t'_0 and some n'_0 in $L_{k-1}(t'_0)$, the work rectangle of (t_0, n_1) is intersected by the work rectangle of the $k-1$-cell $(t'_0, n'_0; a, b)$. Let us define again $h' = \lfloor t'_0/T \rfloor - 1$:

2. The cell n'_0 is not in $E_0(t'_0; a, b)$. Otherwise it follows immediately from the definition of E_0 and E in Section 12 that n_0 is in $E(t)$, contrary to the assumption that n_0 is in $G(t_0, -)$.

3. We proved that n'_0 is in $G^{j'}(t'_0; a, b)\backslash E_0(t'_0; a, b)$ for some j'. Since t_0 is not in $T+[d_0\cdots s_r+0.7w^4d_0)$, it follows from the previous lemma that t'_0 is in $h'T+[d_0\cdots s_r+0.7w^4d_0)$. Therefore t'_0 points to the part of the program containing the repetitions of the procedure *Grow*. It follows from Lemmas 14.2 and 15.1 that the k-cell $(h'+1, j'; a, b)$ is live, and the set $G^{j'}(t'_0, a, b)$ is an interval mod $E(t'_0; a, b)$ containing at least $P-13w^3$ elements. Since $G(t_0, +)$ contains $\mathscr{P}$ and the $k-1$-tribes $L^{k-1}(-)$ and $L^{k-1}(-; a, b)$ are disjoint, all points of the set

$$b+\Gamma_1^{k-1}G^{j'}(t'_0, -; a, b)$$

are to the right of $\mathscr{P}^k$.

4. For the left end of the block $b+\mathscr{P}^k[j']$ we have

$$b+j'P^k > P^k+8P^{k-1}.$$

Indeed, otherwise the work rectangle of the k-cell $(h'+1, j'; a, b)$ disturbs the work rectangle of the canonical k-cell $(1, 0)$. The latter is assumed to be live by the conditions of the lemma. It follows from Lemma 15.4 that if both of these cells are live then the k-cells $(0, 0)$ and $(h', j'; a, b)$ are also live. This would contradict the assumption that the k-cell $(0, 0)$ is proper.

5. Let t_2 be 0 if the retreating part of the *Grow* started less than $0.7w^4d_0$ steps before t'_0, otherwise let it be such that t'_0+t_2 is the first place in the program after t'_0 where the retreating part of *Grow* starts. Then $t_2 \leqslant 2.3w^4d_0$. We define $t_1 = t_2+0.2w^4d_0$.

During steps of $t'_0+[t_2\cdots t_1)$, the retreating part of the procedure *Grow* acts on the $k-1$-cells with origin (a, b). It kills $0.2w^4$ cells of $G^{j'}(t'_0; a, b)$ to the left of $\mathscr{P}[j']$ if there are so many. Remember that *Heal* was not permitted to ressurect in the attack direction of *Grow*. Therefore only the error rectangle can resurrect, at most $2w^3$ of them (together with *Heal* working in the retreat direction). Hence the left end of $G^{j'}(t'_0+t_1, -; a, b)$ is either in $\mathscr{P}[j']$, or to the right of

$$n'_0+0.2w^4-2w^3 \geqslant n'+8w^3.$$

In other words, none of the canonical $k-1$-cells in $[0.5P\cdots P+8w^3-1)$ will be disturbed by $L^{k-1}(-, a, b)$ at time t_0+t_1. This situation does not change in the next $0.1w^4d_0$ steps, since the steps in $t'_0+t_1+[0\cdots 0.1w^4d_0)$ still belong to the retreating part of *Grow*. Taking into acount the effect of the error, still none of the $k-1$ cells $[0.5P\cdots P+5w^3)$ will be disturbed by $L^{k-1}(-; a, b)$ for any t in

$$H = t_0+t_1+[0\cdots 0.1w^4d_0).$$

6. Now we have to prove that for t in H, none of the $k-1$-cells in

$$I(t) = [0.5P\cdots P+5w^3)\backslash E(t)$$

is disturbed by the set $L^{k-1}(a, b)$. Suppose that for some t in the range considered, a $k-1$-cell n in $[0.5P \cdots P + 5w^3)$ is disturbed by a $k-1$-cell (t', n') with $n' \in L_{k-1}(t'; a, b)$. It follows from the statement proved in the previous paragraph that n' is in $E_1(t'; a, b)$. Let us recall the definition of $E_1(t'; a, b)$ in Section 12. As mentioned in 1, above, if n' is in $E_0(t'; a, b)$ then n is in $E(t)$. However, the values of t' considered here are in

$$h'T + [d_0 \cdots s_r + 0.7w^4 d_0).$$

For these values of t' we have $E_0(t'; a, b) = E_1(t'; a, b)$ by definition.

7. No $k-1$-tribes with origins different from $(0, 0)$ and (a, b) can disturb the cells of $I(t)$ during the time interval H. If there was a $k-1$-cell $(t_1', n_1'; a_1, b_1)$ with n_1' not in $E_0(t_1'; a_1, b_1)$ doing this then the argument of 2 above would imply a large intersection of the $k-1$-tribes $L^{k-1}(a, b)$ and $L^{k-1}(a_1, b_1)$, though these cannot intersect outside the error rectangle. ▮

Proof of (O1)(ii). Suppose that the k-cells $(0, 0)$ and $(1, 0)$ are protected. Property (O1)(ii) says that if the k-cell $(0, 0)$ is proper then so is $(1, 0)$. We know already from Lemma 14.2 that if $(1, 0)$ is dead then it is proper dead. Let us assume therefore that $(1, 0)$ is live. We have to prove that

(a) The k-cell $(1, 0)$ is undisturbed;

(b) $[1 \cdots P-1) \subset G_{k-1}(T) \cup E$ for some interval E of length $2w^2$.

We first prove (a). Suppose that (a) does not hold. Then the k-cell $(1, 0)$ is live and disturbed. Lemma 15.4 implies that then $(0, 0)$ is also live and disturbed, contrary to the assumption.

Now we prove (b). Since the k-cell $(1, 0)$ is live, it follows from Lemma 13.4 that $G(t)$ contains $\mathscr{P}$ for some t in $s_0 + [c_1 \cdots 2c_1]$. From then on, the only way for a cell of $\mathscr{P}$ to die is by the error or the death of an improper endcell. If this happens after s_c then the only cells not immediately killed by the error that may die can be w^2 cells at the end cut off by the the error and killed by *Purge*. Therefore all killed cells may be covered by an interval of length $2w^2$. Let us suppose therefore that a cell dies before s_c.

Any gap inside $\mathscr{P}$ will be closed by *Heal*, using Lemma 12.1. The only obstacle to Heal at the ends of $\mathscr{P}$ may be if at some time t_0 during this attempted recovery, say, on the right end, some of the canonical $k-1$-cells near the end are disturbed, preventing the application of (O4). But we can apply Lemma 16.2 in this case. Let us use the notation of this lemma.

During the t_1 steps of their program after t_0, some canonical $k-1$-cells in $\mathscr{P}$ may die. They may die if the error or *Purge* kills them, but the total number of cells killed this way is less than $2w^2$. Otherwise, only the procedure *Shrink* kills, at a rate

of one cell in $w^2 d_0$ steps. Altogether therefore, the number of cells killed during this time is less than $3w^2 + 2.5w^2$. Hence, the leftmost cell killed is to the right of $P - 6w^2$. By the lemma, during the time $t_1 + [0 \cdots 0.1w^4 d_0)$ the $k-1$-cells in a large neighborhood of the right end are undisturbed. The time is long enough and the gap is small enough for *Heal* to resurrect all cells on the right end. ▮

Proof of (O3)(ii). If the k-cell $(0, 0)$ is proper live then there is an interval E of length $2w^2$ such that $[1 \cdots P-1)$ is contained in $G(s_c) \backslash E$. Since *Shrink does not work during* $[s_c \cdots T + s_0)$, the only cells killed during this time will be a possible improper endcell, cells killed by the error and *Purge*. The numbers of these cells altogether will not exceed $3w^2$. Therefore the argument of the previous proof is applicable again. ▮

Proof of (O4). 1. The case that is still unproven is when (taking $a = b = 0$) the k-cell (h, i) is dead and $m_k(h, i)$ is live. We have to prove that $(h+1, i)$ is live. The statement for $(h+1, i+1)$ is symmetrical. The statement that $m_k(h, i)$ is live says that according to the hth step of the program, the dead cell i must apply *Conform*. Without loss of generality, we can assume that the k-cell $(h, i-1)$ is live, and the cell (h, i) applies *Conform*(-1). It is enough to show therefore that $G(T, +)$ contains $\mathscr{P}$. Indeed, according to the proof of Lemma 14.2, the only way that $(h+1, i)$ can become a dead k-cell is now if *Shrink* kills the block before the computation begins. That this does not happen is proven just as in the proof of (O1)(ii) above.

2. We assumed that the k-cell $(h, i-1)$ is proper live, i.e., there is an interval E of length $2w^2$ such that $G(hT) \cup E$ contains $(i-1)P + [1 \cdots P-1)$. It follows that there is a t in $hT + [0 \cdots w^6)$ such that $G(t)$ contains $\mathscr{P}[i-1]$. Now we have to see that by the applications of the procedure *Grow*, the block $\mathscr{P}[i-1]$ of $k-1$-cells overtakes the block $\mathscr{P}[i]$ during $hT + [0 \cdots s_0)$. The only obstacle to this can be that some $k-1$-cells in $[iP \cdots (i+1.5)P)$ are disturbed.

3. Suppose that for some t_0 in $hT + [d_0 \cdots s_0)$ and n_0 in

$$[iP \cdots (i+1.5)P) \backslash E(t_0),$$

the canonical $k-1$-cell (t_0, n_0) is disturbed, by some $k-1$-cell $(t', n'; a, b)$. Then n' cannot belong to $E_0(t'; a, b)$ since, as mentioned in the beginning proof of Lemma 16.2, this would imply $n_0 \in E(t_0)$. Let us define $h' = \lfloor t'/T \rfloor$. Let j' be such that n' is in $G^{j'}(t'; a, b)$. Then the k-cell $(h', j'; a, b)$ is dead. Indeed, otherwise it would disturb one of the canonical k-cells in $[i-2 \cdots i+2]$, which is excluded by the conditions.

4. It follows that t' is not in $h'T + [d_0 \cdots T + s_0)$, since according to case (A) of Lemma 14.2, the set $G^{j'}(t'; a, b) \backslash E_0(t'; a, b)$ is empty during this time (remember that the sets $E_0(t'; a, b)$ and $E_1(t'; a, b)$ are different only during intervals of the form $h''T + [0 \cdots d_0)$ for some h''). Now Lemma 15.3 is applicable. It says that the interval $n' + [-3w^3 \cdots 3w^3)$ has a nonempty intersection with

$L_{k-1}(h''T-1, -; a, b)$. This statement, together with Lemma 14.2, implies the existence of a live k-cell $(h'-1, j''; a, b)$ and a $k-1$-cell n'' in

$$n' + [-3w^3 \cdots 3w^3) \cap G^{j''}(h'T-1, -; a, b).$$

But then the k-cell $(h'-1, j''; a, b)$ still disturbs one of the canonical k-cells in $[i-2 \cdots i+2]$ in period h, a contradiction. ∎

17. Conclusions

Simplification. The construction and proof of this paper are surprisingly complicated for the simple form of Theorem 1. It seems therefore important to look for conceptual simplification. The problem of quantitative simplification is no less pressing. The medium described here has millions of states. In order to lend the medium physical credibility, one must reduce the number of states (while retaining nearest-neighbor interaction). A great amount of reduction is possible within the present framework: e.g., most variables could be shared by a few neighbor cells, the "dovetailing" can be done much more economically, and many actions can be performed in parallel. This was avoided only for reasons of carity. Further attempts at reduction bring up principal problems since "forced self-simulation," a crucial feature of the medium, is expensive.

Positive memory capacity. Here, we need $K \log^c(KL)$ cells to store K bits of information for L steps. The time delay is $\log^c(KL)$ per step. A memory used for a fixed number L of steps is a special communication channel of the type considered in information theory, and one can ask whether it has a positive capacity C in the sense that only $KC/\log s$ cells are needed to store K bits for L steps (at least if K is not smaller than $\log L$), if s is the number of cell-states. This problem has positive solution but several details need to be elaborated. Besides the positive capacity, the time delay will also be decreased to $\log^{1+\varepsilon}(TK)$ with ε arbitary small.

Here are some of the changes to be made for positive capacity. A block of level k has now $k^2 2^{k-1}$ subblocks of level $k-1$. (The new proof of Toom's result also uses these block sizes.) In the inductive definition of k-sparsity we allow 2^{k-1} exceptional subrectangles instead of just one. The simple idea of self-simulation must be abandoned for a more direct definition of the work of a $k+1$-block in terms of the k-blocks comprising it. The spatial repetition must be replaced with an algebraic code of rate $1-1/k^2$. Thus some P_k/k^2 checkbits are added to every k-block of size P_k. The control, mailbox, and other service variables are shared by several cells to cut their contribution to the redundancy. The temporal repetition is about 2^k-fold on level k.

Continuous time. Most interactive particle systems considered by probabilists and physicists work in continuous time as a Markov process. Since such systems are physically more realistic the question arises whether reliable computation is

achievable in them. As Bennett remarked in 1983, continuous time brings in the additional problem of *synchronization.* Still, this problem seems to be solvable. The hierarchical organization used for error-correction can also be used for synchronization. Each block of cells is supposed to be synchronized to a certain tolerance and is periodically resynchronized. Higher order blocks are synchronized to progressively looser tolerances. The details have to be worked out.

Self-organization. The one-dimensional cellular space described in the previous section works reliably only if the starting configuration contains already the (input-independent) hierarchical organization. It would sound more natural that if the input is one bit then the starting configuration should consist just of the repetition of this one bit. If error-correction requires structure then the medium should be able to build up this structure, out of "nothing." In probabilistic terms, this would correspond to the existence of several invariant measures that are also *translation-invariant.*

This goal is the *most intriguing* among the ones proposed. We run into several new problems. First, without errors, no structure can arise, since all cells have the same state and the same rule. Hence whatever structures may arise will be random (e.g., if there are blocks, their position will be random). Second, killing a small organized island surrounded by inconsistencies was one of the main principles of the present contruction. Self-organization requires that we permit these island to grow. A possible new rule replacing the old one could be that islands die if they are prevented from growth for longer time. Assigning different speeds to different kinds of growth) making "self-organizing" growth the slowest) is another trick that might work.

APPENDIX: NOTATION

This is a list of the notation used throughout the paper and the section is given where the notation is introduced:

$[a \cdots b)$, Section 2.
$|E|$, Section 2.
$H(t, +)$, $H(t, -)$, Section 13.
Z^+, Z^-, Section 5.
$\phi \circ \gamma$, Section 3.
ϕ_*, ϕ^*, Section 3.
A, Section 2.
B, Section 10.
c_i, Section 13.
Cn, Section 6.
Comp, Section 9.
Conform'(j), Section 6.
Conform(j), Section 9.
consis'(x_0, x_1), Section 6.
consis(x_0, x_1), Section 7.
Cons'(j), Section 6.
Cons(j), Section 7.
Core, Section 5.
d_i for $i = 0, 1$, Section 13.
δ, Section 9.
Dead, Section 6.
dead, Section 9.
$E(t)$, $E_0(t)$, $E_1(t)$, Section 12.
e^j for $j = 0, 1$, Section 5.
ϕ, Section 3.
γ, Section 3.
$G^i(t)$, Section 13.
Grow, Section 9.
Γ^k, Γ_0^k, Γ_1^k, Section 10.
Heal', Section 6.
Heal, Section 9.
Input$_i$, for $i = 1, 2, 3$, Section 5.

$J = J' \times J''$, Section 12.
$\mathscr{K}$, Section 5.
$L_k(t; a, b)$, $L^k(t; a, b)$, Section 10.
$L'_k(t; a, b)$, Section 15.
m, m', Section 2.
$m_k(h, i)$, Section 10.
M, Section 2.
M_1, Section 5.
M_2, Section 6.
$M_{q,w}$,Section 9.
$Mail_i$ for $i = \pm 1$, Section 5.
Main, Section 9.
Maj, Section 5.
Misc, Section 9.
neut, Section 9.
v, Section 5.
$Output_i$ for $i = 1, 2, 3$, Section 5.
P, Section 3.
P_0, Section 5.
$\mathscr{P}[i]$, $\mathscr{P}^k[i]$, Section 4.
$\mathscr{P}_0[i]$, $\mathscr{P}^k[i]$, Section 5.
$\mathscr{P}^{(j)}[i]$, $\mathscr{P}'[i]$, $\mathscr{P}''[i]$, $\mathscr{P}'$, $\mathscr{P}''$ for $j = 0, 1, 2$, Section 10.
p_0, p_1, p_2, Section 12.
π, Section 5.
Postp, *Prep*, Section 9.
Purge', Section 5.
Purge, Section 7.
ψ, Section 3.
q_0, q_1, Section 12.
r, Section 3.
r_i for $i = 1, 2, 3$, Section 9.
ρ, Section 2.
Readin', Section 5.
Readin, Section 9.
s, Section 3.
Shrink(i) for $i = 0, 1, 2$, Section 9.
$\mathrm{Sim}(q, P_0)$, Section 8.
single, Section 3.
Single, Section 9.
s_i for $i = 0, 1, 2$, Section 7 and 13.
s_c, Section 13.
s_r, Section 15.
S_D, Section 2.
T, Section 3.
$\mathscr{T}[h]$, $\mathscr{T}^k[h]$, Section 4.
T_0, $\mathscr{T}_0[h]$, $\mathscr{T}_0^k[h]$, Section 5.
τ, Section 5.
Temp, Section 7.
U, Section 3.
$V[h, i]$, $V^k[h, i]$, Section 4.
w, Section 4.
$W_j^k[h, i]$ for $j = 0, 1$, Section 11.
x, Section 2.
X, Section 5.
$x^k[h, i; a, b]$, Section 10.
Y, Section 5.
y, Section 2.
$\boldsymbol{Z}$, $\boldsymbol{Z}_m$, Section 2.

References

1. R. L. Dobrushin and S. I. Ortyukov, Lower bound for the redundancy of self-correcting arrangements of unreliable functional elements, *Problems Inform. Transmission* **13** No. 1 (1977), 59–65; Upper bounds on the redundancy of self-correcting arrangements of unreliable elements, *Problems Inform. Transmission* **13** No. 3 (1977), 201–218.
2. P. Gács, L. A. Levin, and G. L. Kurdyumov, One dimensional homogeneous media dissolving finite islands, *Problems Inform. Transmission* **14** No. 3 (1978), 92–96.
3. P. Gács and J. Reif, A simple three-dimensional real-time reliable cellular array, *in* "Proc of the 17th ACM Symp. Theory Computing," 1985, pp. 388–395.
4. R. G. Gallager, "Low-Density Parity-Check Codes," MIT Press Cambridge, Mass., 1963.
5. L. Gray and D. Griffeath, A stability criterion for attractive nearest neighbor spin systems on $\mathbf{Z}$, *Ann. Probab.* **10** (1982), 67–85.
6. M. Harao and Sh. Noguchi, Fault tolerant cellular automata, *J. Comput. System Sci.* **11** (1975), 171–185.
7. G. L. Kurdyumov, An example of a nonergodic homogeneous one-dimensional random medium with positive transition probabilities, *Soviet Math. Dokl.* **19** (1978/1), 211–214.
8. A. V. Kuznietsov, Information storage in a memory assembled from unreliable components, *Problems Inform. Transmission* **9** No. 3 (1973), 254–264.

9. T. M. LIGGETT, "The Stochastic Evolution of Infinite Systems of Interacting Particles," Lecture Notes on Math. Vol. 598, Springer-Verlag, New York/Berlin, 1976.
10. T. F. LEIGHTON AND C. E. LEISERSON, Wafer-scale integration of systolic arrays (extended abstract), *in* "Proc. of the 23rd Annual Symposium on the Foundations of Computer Science Chicago, 1982, pp. 297–311.
11. H. NISHIO AND Y. KOBUCHI, Fault tolerant cellular spaces, *J. Comput. System Sci.* **11** (1975), 150–170.
12. L. SNELL, Personal communication.
13. M. C. TAYLOR, Reliable information storage in memories designed from unreliable components, *Bell System Tech. J.* **47** No. 10 (1968), 2299–2337; Reliable computation in computing systems designed from unreliable components, *Bell System Tech. J.* **47**, No. 10 (1968), 2339–2366.
14. A. L. TOOM, Nonergodic multidimensional systems of automata, *Problems Inform. Transmission* **10** (1974), 239–246.
15. A. L. TOOM, Stable and attractive trajectories in multicomponent systems, *in* "Multicomponent Random Systems" (R. L. Dobrushin and Y. G. Sinai, Eds.), Advances in Probability Vol. 6, pp. 549–575, Dekker, New York, 1980.
16. A. L. TOOM, Estimates for the measures describing the behavior of stochastic systems with local interaction, *in* "Interactive Markov Processes and their Application to the Mathematical Modelling of Biological Systems" (R. Dobrushin, Kryukov, and A. Toom, Eds.), Acad. Sci. USSR, Pushchino, 1982. [Russian]
17. B. S. TSIREL'SON, Reliable information storage in a system of locally interacting unreliable elements, *in* "Interacting Markov Processes in Biology," Lecture Notes in Math. Vol. 653, Springer-Verlag, New York/Berlin, 1978.
18. J. VON NEUMANN, Probabilistic logics and the synthesis of reliable organisms from unreliable components, *in* "Automata Studies" (Shannon and McCarthy, Eds.), Princeton Univ. Press, Princeton, N.J., 1956.