Reliable computation with cellular automata

Péter Gács
Department of Computer Science
University of Rochester
TR132

ABSTRACT:

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 mechanics, 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. To compute reliably 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*.

Author's current address: Department of Computer Science, Boston University, Boston, Massachusetts.

This work was supported in part by the National Science Foundation grants No. MCS 8110430, MCS 8104008, and MCS 8302874, and in part by DARPA grant No. N00014-82-K0193, monitored by the ONR.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: TR 132	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): Reliable Computation with Cellular Automata	5. TYPE OF REPORT & PERIOD COVERED: technical report
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Peter Gacs	8. CONTRACT OR GRANT NUMBER(s): N00014-82-K0193
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Computer Science Department, University of Rochester, Rochester, NY 14627	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Defense Advanced Research Projects Agency, 1400 Wilson Blvd., Arlington, VA 22209	12. REPORT DATE
	13. NUMBER OF PAGES: 54
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Office of Naval Research, Information Systems, Arlington, VA 22217	15. SECURITY CLASS. (of this report): unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

none

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

We construct a one-dimensional array of cellular automata on which arbitrarily large computations can be implemented reliably, even though each automation at each step makes an error with some constant probability. In statistical mechanics, 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. To compute reliably with unreliable components, von Neumann proposed Boolean

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Abstract (cont.)

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? A partial positive answer was given in [vN 52]. (It was subsequently sharpened in [Ta 68]. See also [Kz73] and [D 77]). For any Boolean circuit A of some size N working with reliable components, one can construct a circuit B of size $O(N \log N)$ from components which can make (independent) errors with probability not exceeding some known ρ such that B computes the same Boolean function as A with error probability $O(\rho)$.

Von Neumann's formal solution does not address the reliability problem in sufficient generality. The intricate connectivity pattern of his reliable network B is unrealizable with constant-length connections in any finite-dimensional space. Increasing the length of connections strongly exposes the assumption that errors are confined to the logic elements while their connection pattern is reliable. The information storage devices described in [Ta 68] and [Kz 73], however efficient they are, suffer from the same problem. A reliable 2-dimensional locally connected information-storage device was described in [Ts 76]. Probably, it could be used to implement a 2-dimensional version of von Neumann's reliable circuits. However, the size of this device is proportional to the working time, and the type of automata varies from position to position.

Is reliable computation (or just information storage) feasible in a finite-dimensional array of locally interacting automata (cellular automata, iterative array)? Such a simple connection pattern is already not necessarily subject to errors since it (or some analogous variant) may be enforced by physical law (e.g. the automata may be molecules in a crystal structure), or may even be just a geometrical framework for the description of physical phenomena. Such devices are also the easiest to manufacture (using e.g. VLSI) and assemble in large quantities. Work has been done on fault-tolerant cellular automata e.g. in [H 75, N 75]. However, these papers make very strong assumptions on the pattern of errors. In the terminology of our Section 3, they assume that the errors occur on a 1-sparse set (i.e. never come too close to each other). Hovever, if the errors occur independently with constant probability, we can only assume e.g. that they occur on a k-sparse set where k depends on the size of the space-time area we are concerned with.

The problem of reliable information storage in a cellular structure arises naturally in statistical physics. Let us call *medium* an array of identical cellular automata where the state of every automaton depends stochastically on the states of its nearest neighbors. A medium can be the model of magnetic spins in a crystal, certain states of cells in a tissue, voting behavior, etc., see [L 76, Gr

82]. The subsequent states in time of the whole medium form a Markov process, and the first thing a probabilist askes about such a process is whether it is *ergodic*. If a process is ergodic it eventually loses every single bit of information about its initial state.

We are interested in media where all local transition probabilities are positive. It required great ingenuity to show that *not* all such media are ergodic. Toom constructed in [T 74] several examples of nonergodic media of dimension 2 or higher. The one-dimensional case seemed harder. Toom's media accomplish a sort of local voting and preserve only a few bits of information, using even the whole infinite medium. In [K 78], G.Kurdyumov proposed some ideas for the construction of one-dimensional nonergodic media, using an infinite hierarchy of Turing machine-like media simulating each other. The presentation is so vague that the problem is still considered unsolved by most specialists ([S 80]). Rigorous but more modest results about simpler media are proved in [G 78]. If his ideas are realizable, Kurdyumov can also use his media to implement reliable computation. It seems now that because of the restrictions of the one-dimensional medium (local voting does not work), one cannot solve the problem of information storage (even of one bit) without solving the general problem of reliable computation: an unsolved problem even in higher-dimensional media.

Here, I construct a one-dimensional nonergodic medium M, solving thereby the above problems. Since the construction is partly based on Kurdyumov's ideas, medium M is also capable of reliable computation. Section 2 states the result, and the next sections outline the proof. Refinements of the result will be given in subsequent papers.

The present paper benefited from conversations with L.Levin and G.Kurdyumov, coauthors of [G 78], and C.H.Bennett, whose work on algorithmic depth further stimulated my interest in the question whether deep sequences can arise in nature.

2. Statement of the result

2.1 Media and Markov systems

A (one-dimensional homogenous deterministic) medium is a uniform chain of locally interacting automata, working in discrete time $t = 0, 1, \ldots$. The medium is defined by the finite set S of automata states and the transition function $D : S^3 \to S$ which we will also use to name the medium. Let $\mathbf{Z}$ be the set of integers. For a partial function $x[t, n]$ over $\mathbf{Z}^2$, we will say that x *agrees* with D if the relation

$$x[t+1, n] = D(x[t, n-1], x[t, n], x[t, n+1])$$

holds whenever both sides are defined. (We will generally write the time and space variables as "array indices" in square brackets.)

For a set $E \subset \mathbf{Z}$, let $\xi = (\xi[t, n] : (t, n) \in E)$ be a system of random variables with values in S. For any function $v : E \to S$, we denote by $C(v, t)$ the event that $\xi[i, n] = v[i, n]$ for all $i \leq t, n \in \mathbf{Z}$. In this paper, we will use an ad hoc terminology and call ξ a *Markov system* if for each v, t, the random variables $(\xi[t+1, n] : n \in \mathbf{Z})$ are conditionally independent under condition $C(v, t)$. A Markov system ξ is a *ρ-perturbation* of a medium D if for all v, t the conditional probability under condition $C(v, t)$ of the relation

$$\xi[t+1, n] = D(v[t, n-1, v[t, n], v[t, n+1])$$

is greater than $1 - \rho$ whenever it is defined. We will say that an *error* occurred at (t, n) if

$$\xi[t+1, n] \neq D(\xi[t, n-1], \xi[t, n], \xi[t, n+1]).$$

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

$$[a \ldots b] = [a, b] \cap \mathbf{Z} = \{ n \in \mathbf{Z} : a \leq n \leq b \},$$
$$[a \ldots b) = [a, b] \cap \mathbf{Z} = \{ n \in \mathbf{Z} : a \leq n < b \},$$

etc. For an interval $I \subset \mathbf{Z}$ and a partial function $x[n]$ over $\mathbf{Z}$, we denote the sequence $(x[n] : n \in I)$ as $x|I$. Similarly, for a function $x[t, i]$ over $\mathbf{Z}^2$, we denote the sequence $(x[t, i] : m \leq i \leq n)$ by $x[t] \mid [m \ldots n]$.

We will always suppose that an *ordering* $S = \{s_0, s_1, \dots\}$ is given on the set S of automata states. This permits us to speak of a distinguished state s_0 automatically.

We imagine our space-time as a plane with a left-right space axis and an upward 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 $R = [k \dots k+h) \times [m \dots m+n)$ in $\mathbf{Z}^2$, we call n its *width*, and h its *height*.

Let ξ be a Markov system on $[k \dots k+h] \times [m \dots m+n]$. For strings u, v, w we say that ξ satisfies the *input condition* u and the *border conditions* v, w if

$$\begin{aligned} \xi[k] \mid (m \dots m+n) &= u, \\ \xi[k+t, m] &= v[t], \\ \xi[k+t, m+n] &= w[t] \end{aligned}$$

for $t \in [0 \dots h)$. The border conditions are *standard* if $v[t] = w[t] = s_0$ for all t. The string $\xi[k+h] \mid (m \dots m+n)$ is called the *output* of the rectangle R. If ξ is a 0-perturbation of a medium D then the "contents" of the whole rectangle, and thus its output, are completely determined by the strings u, v, w. We will denote this output by $D^h(u; v, w)$. For standard border conditions, we will write $D^h(u)$.

2.2 Coding and simulation

Our purpose is to find a medium M and a positive constant ρ such that M can simulate the work of any medium D (and thus perform any computation, e.g. just information storage) *reliably*: we get the desired results with high probability even if M is subject to ρ-perturbation. A stable simulation must receive its input and deliver its ouput in some *encoded form*. Otherwise, it loses significant information already in the first or last step.

Let S_0 and S_1 be two state sets. A (P_1, P_0)-*code* is given by a pair (f, ϕ) where the encoding function $f : S_1^{P_1} \to S_0^{P_0}$ and the partial decoding function ϕ are connected by the property

$$\phi(f(x)) = x.$$

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 putting

$$f(u_1 \dots u_m) = f(u_1) \dots f(u_m).$$

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

A *simulation* (or "block simulation") of a medium D_1 by a medium D_0 is given by the *work period* T, *blocklength* P and the *code* (f, ϕ) with $f : S_1 \to S_0^P$. We require for any symbols $s_1, s_2, s_3 \in S_1$ and strings v, w with $f(s_i) = u_i$ that the following simulation relation holds:

$$D_0^T(u_1u_2u_3; v, w) = f(D_1(s_1, s_2, s_3)).$$

Thus the medium D_0 computes in T steps the code of $D_1(s_1, s_2, s_3)$ from $u_1u_2u_3$ under any border conditions. A medium is *universal* if it can simulate any other medium.

To make the reliable medium "universal" it is enough to make sure it can simulate reliably a medium U which is universal in the above sense. In Section 4, we will find a universal medium U. For the purpose of the following theorems, let U be an arbitrary but fixed universal medium.

We want to construct a medium M with the following property. For any string x there is a code (F, Φ) such that if we input $F(x)$ to M with standard border conditions, wait some number of steps and get the output y then $\Phi(y) = U^t(x)$ with large probability. Let us discuss some vague points in this formulation.

Is it not unnatural to assume that coding is error-free? No, because the process of encoding and decoding is only there 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 space factor (redundancy) must our code have. 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 may happen that the redundancy of the code must be increased or decreased during the computation. This also can be done with unreliable elements, but involves some additional problems, hence in this publication we will work with fixed redundancy. We imagine that input and output strings are "padded" to include all memory space needed during the computation.

Is it not possible to cheat, hiding all computation into the coding process? We will give the code explicitely, and it will be clear that no cheating is involved. But cheating is unlikely already given the following properties of our code: decoding is inverse to encoding, and the code is *simple to compute*. It takes only linear time to compute F and Φ on a serial machine, and only logarithmic time on a suitable parallel machine. (This machine must be able to produce n copies of a symbol in $\log n$ time, therefore must be more powerful than cellular automata.)

To define our code we introduce the notion of *concatenation* of codes. For $i = 0, 1$, let (f_i, ϕ_i) be (Q_i, P_i)- codes. Suppose that Q_1 divides P_0. Then the code $(f_1 \circ f_0, \phi_0 \circ \phi_1)$ is defined as follows:

$$(f_0 \circ f_1)(u) = f_0(f_1(u)).$$

If the string u has length Q_0 then the string $(f_0(f_1(u))$ has length P_1P_0/Q_1. The decoding is applied, of course, in reverse order. Example: if $f(0) = 000$ and $f(1) = 101$ then $f(f(1)) = 101000101$. The code $f \circ g$ is called the *concatenation* of f and g. The k-th *iteration* of f is $f^k = f \circ \cdots \circ f$ (k times). It is defined for a code $f : S^Q \to S^P$ if Q divides P. In this paper, we will consider codes which are essentially iterations of some fixed $(1, P)$- code f. Since P is constant, the code f^k is computable in $O(k)$ steps on a suitable parallel machine.

We will suppose that the first and last symbols λ and μ of the word $f(s)$ are independent of s (view them as delimiters). Then the strings

$$\lambda, f(\lambda), f^2(\lambda), \ldots$$

form a sequence where each previous string is a prefix of the following one. The limit is an infinite string $f^\infty_+ = v_0v_1\ldots$. Similarly, the strings $\mu, f(\mu), f^2(\mu)\ldots$ form a sequence where each string is a postfix of the next one. The limit of this sequence is a string

$$f^\infty_- = \ldots v_{-3}v_{-2}v_{-1}$$

infinite toward the left. If we join these two strings we get a doubly infinite string $f^\infty = f^\infty_- f^\infty_+$.

The encoding we use will involve another step. An appropriate function $g : S_M \times S_U \to S_M$ will be used with a partial inverse γ such that $\gamma(g(a, b)) = a$. The function g will be extended to strings as follows.

$$g(a_1 \ldots a_m, b_1 \ldots b_n) = g(a_1, b_1) \ldots g(a_k, b_k)$$

where $k = \min(m, n)$. We view the operation g as entering the "data" string b into the "software" string a. The function g is called the *entering operation.*

The code (F, Φ) of the theorem uses a *base code* (f, ϕ) and an entering operation (g, γ). With their help, we define

$$F_k(u) = f^k(g(f^\infty_+, u))$$

for an appropriate number k. Thus to obtain $F_k(u)$ we enter u into a large enough segment of our standard "software" string f^∞_+, and then apply k more times the encoding f. Of course, the inverse is $\Phi_k = \gamma \circ \phi^k$. We will say that any base code f and entering operation g determine in this way a *standard code* (F, Φ). The *space factor* of F is the same as the space factor of f.

THEOREM 1 *There is a universal reliable medium M, constants ρ, P, T, c and a standard code (F, Φ) with space factor P, with the following property. For any n, t, ϵ, any string $u \in S_U^n$, putting $N = nt/\epsilon$, $k = \lceil c \log\log N \rceil$, for any ρ-perturbation of M on a rectangle with input $F_k(u)$, height $T^k t$, standard border conditions and output η, we have*

$$\text{Prob}[\Phi_k(\eta) = U^t(u)] > 1 - \epsilon.$$

The present paper is essentially devoted to the proof of this theorem. The remainder of this section is devoted to its discussion, and a related theorem which is applicable to statistical mechanics.

The simulation promised in the theorem has a space factor $P^k = \log^\beta N$ and time factor $T^k = \log^\alpha N$ for some constants α, β. For the time being, the role of ϵ is not important. We want to avoid that the output be garbage with high probability as a result of the accumulation of errors. For this, even $\epsilon = 1/3$ would do. Thus, the space and time redundancy we pay for reliability is $\log^\alpha$ of the number of elementary operations to be performed in the original computation. In a subsequent publication, we will show how to improve the time factor to $\log N o(\log N)$. The space factor can be made almost constant as long as $\log t = O(n)$.

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 form an ergodic Markov chain whose state converges this fast to a unique equilibrium state. The product of these time and space factors comes close to von Neumann's factor $\log N$, which is shown in [D 77] to be in some sense optimal. I would like to emphasize that the present paper answers not only the question what are the optimal time and space factors of reliable computation, but also whether reliable computation (in the sense defined) is possible at all.

We now formulate a theorem which immediately implies the existence of a nonergodic one-dimensional stochastic medium. For any symbol s, let s^∞ be the doubly infinite string each element of which is s.

THEOREM 2 *There is a medium M, constant ρ and a standard code with base code (f, ϕ) and entering operation (g_0, γ_0), with the following property. For any ρ-perturbation ξ of M on $[0 \ldots \infty) \times \mathbf{Z}$ with input $g_0(f^\infty, s^\infty)$, for any integers n, t we have*

$$\text{Prob}[\gamma_0(\xi[t, n]) = s] > 2/3.$$

We refer to the work [T 74] for the definition of ergodicity, and to the simple proof of how the nonergodicity of any ρ-perturbation of M follows from this theorem. But without any technical definition of ergodicity, it is evident that after we used the sequence $g(f^\infty, s^\infty)$ as initial configuration of our Markov system ξ, the theorem states that ξ "remembers" s forever.

3. The sparsity of errors.

The error-correcting structure we choose is an infinite hierarchy of simulations. Medium M_1 simulates (whenever it works correctly) some medium M_2, which simulates some medium M_3, etc. In the present paper, all these media will be the same appropriately chosen medium M, and all the simulations will use the same code (f, ϕ), work period T and blocklength P.

These simulations form a structure strong enough to carry the load of an arbitrary miscellaneous computation, the one we really want to perform. The input of this miscellaneous computation will be injected using the entering operation g. The iterated simulation will give rise to higher order blocks. Put

$$T^k = [0 \dots T^k),\ P^k = [0 \dots P^k),\ V^k = T^k \times P^k.$$

We omit the superscript k for $k = 1$. For any subset B of $\mathbf{R}^2$ and numbers a, b, put

$$(a,b) + B = \{(x+a, y+b) \ : \ (x,y) \in B\},$$
$$aB = \{(ax, ay) \ : \ (x,y) \in B\}.$$

Similar definitions apply in $\mathbf{R}$. The first of these sets (the shift) will also be called a *copy* of B.

The cells in the k-th order block P^k would, under error-free conditions, perform a coordinated activity over the working period P^k. Of course, they will make errors, but they will be designed to work satisfactorily as long as the set of errors in the rectangle V^k and a few of its neighbors is *k-sparse*.

Let c_0 be an arbitrary positive constant. In this paper, we set

$$c_0 = 3.$$

A set $E \subset \mathbf{Z}^2$ is *0-sparse* if it is empty. It is *k-sparse*, if for every copy I of $c_0 V^k$ there is a copy J of $3c_0 V^{k-1}$ such that $E \cap I - J$ is $(k-1)$-sparse. The concept of k-sparsity is similarly defined for sets in $\mathbf{Z}$. Ignoring the multipliers in the definition, we could say that a 1-sparse set is one whose elements are far enough from each other so that no two of them belong to the same copy of V. With a two-sparse set, it may happen that more than one element occurs in some copy J of V but such events J are so rare that no two of them happens in the same copy of V^2.

Let p_k denote the probability that the set of errors in $2c_0 V^k$ is not k-sparse, i.e. that a "k-error" occurs. It turns out that

$$p_{k+1} = O(p_k^2),$$

since a $k+1$-error means approximately the occurrence of two k-errors in V^{k+1}. The following lemma gives thus 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 1 *There is a $c > 0$ such that for every $\rho < c$, every k, n, t, every ρ-perturbation of a medium over a rectangle of height $2c_0tT^k$ and width $2c_0nP^k$, the probability that the set of errors is not k-sparse is less than*

$$nt\rho^{2^{k-1}+\frac{1}{2}}.$$

Notice that for $k = 0$ the lemma gives the obviously valid estimate $nt\rho$.

Proof: Let $\mathcal{E}$ be some set of subsets of $\mathbf{Z}^2$, and E the set of errors in some ρ-perturbation of our medium. It is easy to see that for any rectangle R we have

$$\text{Prob}[E \cap R \subset \mathcal{E}] \leq \sum_{R \supset F \in \mathcal{E}} \rho^{-|F|}.$$

We can therefore assume that the errors occur independently with probability ρ, since in this case equality is achieved in the above estimate.

Notice that a set E is k-sparse if and only if its intersections with every copy of c_0V^k are k-sparse.

We prove the lemma by induction. If the errors occur independently, it is convenient to deal with a lattice consisting of disjoint rectangles. Since the lemma speaks about arbitrary copies of V^k and V^{k-1}, we will use two overlapping lattices of rectangles. Let R be a rectangle of height $2c_0tT^k$ and width $2c_0nP^k$. We define two partitions P_0 and P_1 of R into copies of $2c_0V^k$ as follows. Partition P_j consists of the intersections with R of all rectangles of the form

$$((2h+j)c_0T^k, (2i+j)c_0P^k) + 2c_0V^k.$$

Thus, the corners of the rectangles in partition P_1 are in the centers of the rectangles of P_0 (and vice versa). The number of elements in any of these partitions is at most

$$(n+2)(t+2) \leq 9nt.$$

Suppose that R contains a copy I of c_0V^k such that $I \cap E$ is not k-sparse. Then I is contained in an element K of $P_0 \cup P_1$, hence $E \cap K$ is not k-sparse. Hence $18nt\rho_k$ bounds the probability that $R \cap E$ is not k-sparse.

We now estimate p_k. Let us subdivide K in the manner described above, into two partitions $\mathcal{R}_0$ and $\mathcal{R}_1$ consisting of copies of $2c_0V^{k-1}$. The number of elements in $\mathcal{R}_j$ is at most $(T+2)(P+2)$. Let U be the event that there are two disjoint elements J_1, J_2 of $\mathcal{R}_0 \cup \mathcal{R}_1$ such that $J_i \cap E$ is not $k-1$-sparse. We prove that $p_k \leq \text{Prob}[U]$. Indeed, suppose that U does not occur, and J_0 is a rectangle of $\mathcal{R}_0 \cup \mathcal{R}_1$ such that $J_0 \cap E$ is not sparse. Then all other such rectangles J_1 must intersect with J_0 and each other. It is easy to see that there can be at most one such J_1, and $J_0 \cup J_1$ is contained in a copy J of $3c_0V^{k-1}$. Then the set $K - J \cap E$ is $k-1$- sparse.

Thus we have

$$\begin{aligned} p_k \leq \text{Prob}[U] &\leq \binom{2(T+2)(P+2)}{2} p_{k-1}^2 \\ &\leq 2(T+2)^2(P+2)^2 p_{k-1}^2. \end{aligned}$$

Using the inductive assumption on $2c_0V^{k-1}$, we have

$$p_{k-1}^2 \leq \rho^{2^{k-1}+1}.$$

Hence the total probability is estimated by

$$18ntp_k \leq antp^{2^{k-1}+1}$$

with $a = 36(T+2)^2(P+2)^2$. Putting $c = a^{-2}$ makes the induction work. ■

Our purpose is to design the medium M and the simulation f in such a way that the work of a small group of k-rectangles is essentially undisturbed by a k-sparse set of errors. Let us reflect on this requirement. Put

$$\begin{aligned} V^k[h,i] &= (hT^k, iP^k) + V^k, \\ \mathcal{T}^k[a] &= aT^k + \mathcal{T}^k, \\ \mathcal{P}^k[a] &= aP^k + \mathcal{P}^k, \\ x^k[h,i] &= \phi^k(x[hT^k] \mid \mathcal{P}^k[i]). \end{aligned}$$

(The definition of x^k will be slightly changed later.) Thus $x^k[h,i]$ is the element of S_M represented, via the encoding f^k, by the segment $\mathcal{P}^k[i]$ at the time hT^k, i.e. in the starting row of the rectangle $V^k[h,i]$.

If no errors occurred we would have the relation

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

Our purpose is to maintain this relation despite a k-sparse set of errors. The rectangles directly involved in this computation are $V^k[h, i+p]$ for $p = -1, 0, 1$. It turns out that a few neighbor rectangles may be indirectly involved. We will suppose the set of errors to be k-sparse over these rectangles. The concept of k-sparsity is defined in a way facilitating proof by induction. Let I be a rectangle containing all rectangles involved in the computation of $x^k[h, i]$ from $x^k[h, i+p]$ for $p = -1, 0, 1$. Then k-sparsity will guarantee that there is a rectangle J whose size is of the order of V^{k-1} such that the set of errors is $k-1$ on $I - J$. Therefore two issues need only concern us.

First, that since we cannot suppose anything about the nature or errors within the rectangle J, all structure can be destroyed within J or, what is worse, replaced by some "malignant growth". Second, even if we are able to reestablish the structure of hierarchical simulation on the heirs of J, all information contained in J is lost or altered. The second problem is less serious: redundancy takes care of it. But the problem of reestablishment of structure is new, and is in some sense the central problem of this paper.

Originally by "structure" we mean the hierarchy of simulations described above, and by "information" just the data connected with the miscellaneous computation it carries. But fortunately, on any level of our structure, the structure of the higher levels appears to be just like any other kind of information. And the structure of the lower levels need not concern us since the present level would not even exist without the proper functioning of the lower levels. It follows that it is enough to care about the structure of one level (or maybe two to achieve a rippling effect of certain operations over all levels).

Let the rectangle J be the product of the time segment J_0 (we will later define J_0 somewhat differently) and space segment J_1. Since in the rectangle J anything could happen, we are concerned with

Preserving structure around J_1 in the time segment J_0;

Rebuilding structure on J_1 after J_0.

The procedures described in Section 5 serve mainly the second goal, but they are designed with extra care to achieve the first goal too. The "normalization" of J_1 will happen in the following three steps.

1. We conclude by induction that a few steps of size T^{k-1} after J_0, the segment J_1 is covered with a few islands which are structured up to level $k-1$. The structures of the different islands may be inconsistent with the original one (e.g. their $k-1$ -blocks may start at a place which is not a multiple of P^k) or each other.

The area between the islands is "dead". Being dead is one of the possible states of a cell of our medium. A peculiar property of our simulation is that dead cells are encoded essentially by an array of dead cells. Thus if a k-cell dies it dies on all levels. All of its structure decays, freeing the place for any new structure.

2. The islands inconsistent with the neighborhood of J_1 will recognize this inconsistency in their normal $k-1$ -level activity and commit suicide.

3. The healthy neighborhood of J_1 reoccupies the dead areas and reestablishes the $k-1$ -structure over them.

We define M in Section 5. In Section 6, we introduce a certain property (called k-organizedness) implying that the sequence of events 1-3 takes place and formulate the Main Lemma which asserts that k-sparsity implies k-organizedness. In Section 7, we use the Main Lemma to prove Theorems 1 and 2. We prove the Main Lemma in Section 8.

4. A universal medium

The literature contains examples of universal media with a very small number of states. I propose the following medium, which is though not minimal but easy to program and simulate. Let $z = \langle y_0, y_1, y_2 \rangle$ be a "pairing" operation with $y_i = \langle z \rangle_i$ denoting the inverse. Let $\mathcal{T}$ be a universal machine (Turing or other). Let $\mathcal{T}_b(p, x, y, z)$ be the output of $\mathcal{T}$ after b steps of computation, with program p, arguments x, y, z all of which are strings of length $\leq b$ of integers with absolute value $\leq b$. We put

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

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. Here is the outline of a simulation of an arbitrary medium D by U_b. Each cell of D is represented by a group of consecutive U_b-cells delimited by markers. A group divides into a subgroup of length $O(\log|D|)$ to store the current state of the D-cell, a working area of the same length, and a subgroup 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 groups 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 $\mathcal{T}_b$ 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 is not obliged to carry out the simulation in the way outlined in the previous paragraph. In fact, if there is a "small" medium E "efficiently" simulating D then we get an efficient simulation of D by U combining the simulation of D by E and that of E by U.

For a U-cell in state x, let us write $\mathrm{Prog}(x) = \langle x \rangle_0$, $\mathrm{Rep}(x) = \langle x \rangle_1$, and call these parts of the state the *program* of the cell and the value *represented by it.* For a string $u = u[1] \ldots u[k]$ we will write

$$\mathrm{Prog}(u) = \mathrm{Prog}(u[1]) \ldots \mathrm{Prog}(u[k]),$$

and we will proceed similarly with other functions of states. Let $\beta(R)$ be the binary representation of a number R.

Let us restrict a little the simulations we will talk about. We can assume that for every medium D there is a simulation $\mathrm{Sim}(p) = (f, \phi)$ of D by U with

time period R_0 and space period R_1, with the following properties. We have $R_1 < R_0$. For any $s \in S_D$ and $u = f(s)$, the string Prog(u) is

$$p = p_1 * \beta(R_0) * \beta(R_1) * p_2$$

possibly followed by zeroes. Here p_1 does not contain the symbol $*$ and the strings p_i do not depend on s. Also, $\langle u \rangle_2 = 0$. Thus the program p of the simulation determines R_0, R_1, the decoding $\phi(u)$ depends only on Rep(u), and the strings $p, \text{Rep}(u)$ determine u. We will call any string u with $\langle u \rangle_2 = 0$ the *starting configuration* of a simulation. This name reminds us that during the simulation period, we can get $\langle u \rangle_2 \neq 0$.

We can often view the states s of the medium D to be simulated as r-tuples $(s_1, \ldots, s_r)$. Then we can require from the U-simulation that for any $u = f(s)$, the string Rep(u) have the form

$$v_1 * v_2 * \ldots * v_r$$

where v_i does not contain $*$, has fixed length and depends only on s_i. We are also free to choose the encoding of the symbols s_i by whatever strings v_i we want.

5. The medium M

We define a medium M which, besides making many error-correcting efforts, "simulates" itself. The trick to do self-simulation is well-known: it is closely related to the proof of the recursion theorem. First we define a medium M_p. A block of medium M_p will, besides many error-correcting efforts, follow the work of a block of the universal medium U when the latter is performing the simulation $\mathrm{Sim}(pp)$. (The repetition in pp is no misprint. The program of the U-simulation we want to follow is the string pp.) It will turn out that there is a string q such that $\mathrm{Sim}(qp)$ is a simulation of the medium M_p for all p. Choosing $M = M_q$ gives the desired self-simulation.

Suppose that the working period and the blocklength of the simulation $\mathrm{Sim}(pp)$ are R_0 and R_1 respectively. The blocklength of the simulation M_p is

$$P = 3R_1 + 2.$$

The working period is $T = O(R_0 + R_1)$. The block P is divided into the two *endcells*

$$e^j = \frac{j-1}{2} \pmod P$$

and three subintervals K_1, K_2, K_3 of length R_1. Their union is denoted by K. Thus we have

$$K = K_1 \cup K_2 \cup K_3, \; P = \{e^-\} \cup K \cup \{e^+\}.$$

A block of M_p will contain essentially three copies of the contents of a block of $\mathrm{Sim}(pp)$. During a working period of M_p, the work of a working period of $\mathrm{Sim}(pp)$ will be performed three times. We will see in detail, how to organize this activity. Tripling in space and time would sufficiently protect us from a sparse set of errors if these errors were confined to the "information" contained in our cells, and each cell remembered at each step of the working period, what to do with whatever information it has.

To distinguish the different sorts of information present in a cell, the cell states $x[t,n]$ of M_p are determined by a collection of *variables* $Z_1[t,n], \ldots, Z_r[t,n]$. To represent the word of values of a variable Z over an interval I, we will write $Z|I$ for $(Z[n] : n \in I)$. We will write $Z^j[t,n]$ for $Z[t, n+j]$. When speaking about the present step and a fixed cell, we may omit t, n and write Z^-, Z, Z^+ for the value of the Z variable of the left neighbor, the current cell and the right neighbor respectively (writing sometimes, as here,

$+, -$ instead of $1, -1$) . We list some of the main variables, leaving the rest to the procedures which use them. Each variable has a *default value.*

Every cell of M_p knows what kind of step to perform at a certain stage by looking at its variables τ, π (let us call them "counters"). The variable $\tau[n] \in \mathcal{T}$ shows which step of the working period is now being performed by cell n, while $\pi[n] \in P$ shows the place of cell n in its block. To distinguish newly "occupied" cells (see later) from the rest we let the variable π vary in the enlarged range $[-1.1P \ldots 2.1P)$. The default operation is, of course, to increase τ in each step by 1 modulo T, and to leave π unchanged.

We will have several variables "of type U":

$$X, Y, Input_i, Output_i, Misc_k, Mail_j$$

for $i = 1,2,3$, $k = 0,1$, $j = -1,1$. These take values from the set $S_U \cup \{Dead, Out\}$. The variable X contains the value "represented" by the cell in the simulation. In general, it is changed only in the last step of the working period.

If $X = Dead$ or Out we are in a unique state and no other variables matter. If $X = Out$ the cell is in the distinguished state s_0 of its state set $S_M = \{s_0, s_1, \ldots\}$. This state is used only to delimit the whole working area, and no cell is supposed to assume it. Thus

(5.1.1) If in a cell we have $X = Out$ then in the next step we put $X \leftarrow Dead$.

We will say that the cell is *live* if $X \in S_U$. When X is *Dead*, we will say that the whole state x is dead. The default value of a variable of type U is some distinguished element α of S_U. Ideally, all three words $X|K_s$ for $s = 1,2,3$ are equal to the state of the same block in the simulation $\mathrm{Sim}(pp)$, and $X = \alpha$ in the endcells e^j.

(5.1.2) If $X = \alpha$ and $X^- = X^+ = Out$ then $X \leftarrow \alpha$.

This rule keeps the cell alive which on its lower levels contains our whole configuration.

If a cell is live it always has τ, π-values. If it is dead then it uses the τ, π-values of one of its live neighbors for orientation. (If it has two inconsistent live neighbors it remains dead). Thus, a cell n defines its *home block* as the interval $n - \pi[n] + P)$. It divides the line into *blocks* which are shifts of the home block by multiples of P.

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 variables $Output_i$ containing the results of the three imitations are only altered when they receive the new values computed.

These devices limit the consequences of a small group of errors. An error is dangerous if the values of the counters are changed. But such a change will be recognized and corrected since it creates inconsistency between neighbor cells. If the counters are restored the block continues to function according to the program and the effects of the error on the other variables remain localized to a third (in space or time) of a work rectangle.

The variables $Misc_0, Misc_1$, of type U, are used to perform a miscellaneous computation. Error-correcting steps will keep $Misc_0$ constant. If $X = Out$ then $Misc_1$ assumes the distinguished symbol s_0 of S_U. Further, we put

$$Misc_1 \leftarrow U(Misc_1^-, Misc_1, Misc_1^+). \tag{5.1.3}$$

In regions where there is even one error the $Misc_1$ values are meaningless. Thus their usefulness is confined to the highest level where errors are improbable.

We describe the function M_p in terms of *procedures*, which are then combined at the end. The block will perform two functions essentially simultaneously: simulating a cell of M and maintaining consistency. The two functions manipulate different variables, therefore they will not conflict with each other. We begin with the simulation.

5.1. Computation.

The procedure *Readin* reads in the information found in the variable X in the current block K and the two neighbor blocks $K \pm P$. Since in these blocks all information is repeated 3 times, we write the three supposedly identical thirds

$$X|K_s - P \text{ for } s = 1, 2, 3$$

of the left neighbor block into the corresponding variables $Input_s|K_1$ of the left third of the current block, and proceed similarly with the other two blocks. All transfer operations use the *mailboxes* $Mail_-$, $Mail_+$. Normally, a cell would put $Mail_j \leftarrow Mail_j^j$ for $j = \pm 1$ in every step.

(5.1.4) *If we read from a neighbor whose state is Dead or Out, we write Dead or Out respectively.*

But at times determined by our goal, a cell will put $Mail_j \leftarrow X$ or $Input_k \leftarrow Mail_j$ for the appropriate j and k. Finally a majority vote decides among the three pieces of information read in over each other.

Procedure Readin

for $k = 1,2,3$, $j = -1,0,1$ do

$Input_k | K_{j+2} \leftarrow X \mid (jP + K_k)$;

$Y \leftarrow$ the majority of $Input_k$ for $k = 1,2,3$;

After we read in the information we do some initialization, using the procedure *Init*. This step is crucial, and makes the work of our block different from ordinary simulation. The block rewrites the part of the information which is not really information since it is known by the definition of the medium. (Despite the voting, it does not trust what was read in. This is an important step to avoid "cancers" or "viruses", i.e. working blocks with a wrong program.) Especially, knowing p it knows the format of a starting configuration of a block of Sim(pp). Therefore it imposes this format on $Y|K_s$. Such a precaution guarantees that the *only activity* a consistent block of M_p is capable of is the imitation of Sim(pp).

Let us recall the format of a standard simulation Sim(pp). If the states of the simulated medium consist of several variables $Z_1, \ldots, Z_r$, then in the configuration u of the simulating group, $\mathrm{Rep}(u) = v_1 * \ldots * v_r$. Let I_m denote the interval occupied by v_m in $[0 \ldots R_1)$ for $m \in [1 \ldots r]$. Let $I_1(s), \ldots, I_r(s)$ denote the subintervals of K_s corresponding to $I_1, \ldots, I_r$.

Now we remember that the medium whose simulation we are imitating, will turn out to be M_p itself. Hence its state is given by a collection of variables. Without loss of generality, let the first of these be X. Suppose that in Sim(pp), the values *Dead*, *Out* are represented by the numbers 0,1.

Procedure Init

for $s = 1,2,3$ do (

if $Y =$ *Out* on K_s then $\mathrm{Rep}(Y) \mid I_1(s) \leftarrow 1$;

else if *Dead* or *Out* occurs in $Y|K_s$ then put 0 in the same place;

(Together with (5.1.4), this step achieves that e.g. the left input is *Out* if we are at the left border and *Dead* if the left neighbor block is partial.)

if $Y \notin S_U$ then $Y \leftarrow \alpha$;

Separate by $*$ the intervals $I_1(s), \ldots, I_r(s)$ in $\mathrm{Rep}(Y) \mid K_s$;

$\mathrm{Prog}(Y) \mid K_s \leftarrow pp$;

$\langle Y \rangle_2 \mid K_s \leftarrow 0$;

)

In the last two steps, we do not alter $\mathrm{Rep}(Y)$.

After initialization, the procedure *Core*(i) performs the actual simulation. In this, we pretend that the three blocks $Y \mid K_s$ for $s = 1,2,3$ are just three

consecutive blocks of length R_1 of the medium U and iterate on them R_0 times the transition rules of U. The result will be found in $Y \mid K_2$. Then it sends the result out in three identical copies into $Output_i \mid K_s$ for $s = 1, 2, 3$.

Finally, if the result represents a dead cell, we write *Dead* everywhere in Y, preparing the whole block to die. This is another essential element of the construction. If our simulation says that the cell represented by our block becomes a dead cell, then the whole block will die at the end of the working period. Thus if according to the nested simulation, the cell represented by some higher-order block dies, it will die on all lower levels, and become an interval of dead cells.

Procedure Core(i)

Perform the R_0 steps of a job of Sim(pp) with $Y|K$.
if you read *Dead* or *Out* use α instead;
if Rep(Y)$|I_1(2) = 0$ then $Output_i|K \leftarrow Dead$ ·
else for $s = 1, 2, 3$ do $Output_i|K_s \leftarrow Y|K_2$;

Here is the whole computational side of the program.

Procedure Comp

Idle $5.5P$ steps;
(This parallels the repetitions of *Ocp*, to be defined later on.)
for $i = 1, 2, 3$ do(
Idle P steps;
(To separate the three thirds from each other in time.)
Readin; *Init*; *Core*(i);
)
Idle a few steps for divisibility by c_5 (see later);

5.2. Consistency

In this subsection, we define the part of the program designed to maintain consistency. It runs parallelly with the computational part.

The variable $Cons_j$ is 0 if the cell is inconsistent with its neighbor in direction j, i.e. the following does not hold:

$$\tau = \tau^j,\ Misc_0 = Misc_0^j,\ X^j \in S_U,$$
$$\pi^j \equiv \pi^i + j - i \pmod{P}.$$

If $Cons_j \neq 0$ then $Cons_j = 2$ if $\pi^j = \pi^i + j - i$, and 1 otherwise. As we see, we do not really need $Cons_j$ since its value is computable from other variables. The places where $Cons = 1$ are usually the meeting places of neighbor blocks, except during times when a block is overtaken by two neighbor blocks. In this case, the occupying blocks reach beyond their boundary. We have $Cons = 2$ at the old block boundary and 1 at the meeting place of the left and right occupying arms.

The auxiliary procedure *Conform* makes a cell consistent with its left of right neighbor.

Procedure Conform(j)
If (
$X = Dead,\ X^j \in S_U,$
$(X^{-j} \notin S_U$ or $\tau^- = \tau^+)$
(Due to this condition, a dead cell can determine which neighbor it has to conform to.)
$\pi^j - j \in [-1.1P \ldots 2.1P)$
)
then (
$\tau \leftarrow \tau^j;\ \pi \leftarrow \pi^j - j;$
$X \leftarrow d[\pi \pmod{P}];\ Misc_0 \leftarrow Misc_0^j;\ H \leftarrow 0;$
All other variables get their default values;
)

There are certain procedures which we want to start again and again. Let us introduce the following notation. The length of a procedure P is $|P|$. If P and Q are two procedures and c a positive interger then we get the procedure

$$P[c]Q$$

in the following way. First we perform P. Then we perform c steps of Q. Then we interrupt Q and again perform P. Then we again perform c steps of Q, etc. until there are any steps of Q left. We have

$$P[c](Q[cd]R) = (P[c]Q)[(c + |P|)d](P[c]R). \tag{5.2.1}$$

whenever c divides $|Q|$.

The maintaining program is

$$Purge[c_2](Recover[c_5]Maintain), \tag{5.2.2}$$

where the procedures *Purge*, *Recover* and *Maintain* and the constants c_2 and c_5 will be defined below. To have (5.2.1), we will make both c_5 and $|Recover|$ divisible by c_2.

The procedure *Maintain* consists of some repetitions of a procedure *Ocp* (which has length $5c_5$) followed by some repetitions of the procedure *Integrity* (length c_5).

Procedure Maintain

repeat $1.1P/c_5$ times *Ocp*;

($H \leftarrow 1$; $\pi \leftarrow \pi \pmod P$);

(Middle)

repeat until the end of *Comp Integrity*;

if $H = 0$ then $X \leftarrow$ *Dead*

else ($X \leftarrow$ the majority of *Output*(i) for $i = 1, 2, 3$;)

(End)

The role of the variable H will be explained later. Let us agree that the steps Middle and End are performed simultaneously with the preceding step (to preserve divisibility).

In the procedure *Ocp*, a healthy block tries gradually to impose its own structure on a neighbor block of dead cells, by extending an "occupying arm". The occupying arm will be partially withdrawn (in the "retreating" part of *Ocp*), to make sure that a spurious block does not kill a good one due to an error.

The block we created by applications of *Ocp* will *represent* a dead cell, and only actual computation decides whether we will really convert it to a block representing a live cell, or kill it. Therefore *Ocp* assigns the X-variables the values they would get in *Init*, with $\mathrm{Rep}(X)|I_1(s) = 0$ for $s = 1, 2, 3$. Let us denote the sequence of X-values thus obtained by $d[i]$ for $i \in P$.

Procedure Ocp;

(Attack)

repeat $2c_5$ times

for $j = -1, 1$ do

if $\pi^j - j \not\in P$ then *Conform*(j);

(Retreat)

repeat c_5 times

for $j = -1, 1$ do

if $(j(\pi - e^j) > 0$, $Cons_j = 0$,

$(X \neq Out$ or $\pi \not\equiv e^j \pmod P)$

(This condition makes sure we do not retreat from the border of the whole computing area.)

)

then $X \leftarrow Dead$;

(Wait for a possibly damaged good block to recover.)

Idle $2c_5$ steps;

The procedure *Integrity* is the block's way of checking its own integrity. The variable H is 0 if the cell thinks its block is a partial one and 1 otherwise. Initially, it is 1, but the procedure *Integrity* propagates $H = 0$ from any discontinuities with the speed c_2/c_5. To achieve unanimity in the final decision, there will be a final time T_3 after which no new signal $H = 0$ does arise, only the old ones propagate. We do not consider a block partial if it ends with *Out*.

Procedure Integrity

repeat c_2 times

for $j = -1, 1$ do

(The step that follows is called a *marking step*.)

if ($\pi \not\equiv e^j \pmod P$), $(H^j = 0$ or $(Cons(j) = 0, \tau < T_3))$)

then $H \leftarrow 0$;

Idle $c_5 - c_2$ steps;

The constant T_3 is defined by the requirement that it is the last time in the program after which still $1.1P$ marking steps remain.

As seen above, the procedure *Purge* will be performed after every c_2 steps of anything else. It kills small groups of cells inconsistent with their neighborhood

or sets $H = 0$ in them. How large can such groups be? An error rectangle in a 1-sparse set of errors has the size $3c_0 \times 3c_0$. It will turn out that the created damage can extend 3 more cells on both sides, bringing it to

$$c_1 = 3c_0 + 6$$

cells. Before the time influenced by the error ends, the damage can grow c_1 units on both sides. Since c_2 steps can pass before a next application of *Purge*, the size of the damage can already be

$$c_3 = 3c_1 + 2c_2.$$

The procedure *Purge* consists of two parts. In the first part, from any place of inconsistency, a message gets propagated to the right about the kind of inconsistency found there. The second part sets $H = 0$ in the marked cells or kills them if the message met some inconsistency on the right.

The intermediate cells are marked using the variable *Cn* which remembers the consistency problem found on the left. The values 0,1,2 correspond to these values of $Cons_-$, while 3 means $H^- = 0$. We kill a small homogenous group if it has $Cn \in \{0,1\}$ on both sides. We write $H = 0$ in the group if $H = 0$ on one side and $Cn \in \{0,3\}$ on the other side.

The details of the organization, here as well as in *Recover*, 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 revive dead cells.

Procedure Purge
  $Cn \leftarrow 2$;
  repeat c_3 times
    if ($Cn^- \neq 2$, $Cons_- \neq 2$) then $Cn \leftarrow 0$
    else if $Cn^- \neq 2$ then $Cn \leftarrow Cn^-$
    else if $Cons_- \neq 2$ then $Cn \leftarrow Cons^-$
    else if $H^- = 0$ then $Cn \leftarrow 3$;
  repeat c_3 times
    if $\{Cons^+, Cn\} \subset \{0,1\}$ then $X \leftarrow Dead$
    else if (($Cn = 3$, $Cons^+ \in \{0,3\}$) or ($Cn \in \{0,3\}$, $H^+ = 0$))
    then $H \leftarrow 0$;

The program before the interjecting of *Purge* has already the procedure *Recover* interjected after every c_5 steps. This procedure tries to resurrect the cells in a small gap. The procedure *Conform* used for this marks all new cells by $H = 0$. Then *Recover* tries to restore $H = 1$. The variable $Temp_0$ marks newly created cells, and $Temp_1$ marks the ones in which we wrote $H = 1$. The temporary changes will be repealed if they do not close a gap.

Procedure Recover

$Temp_0 \leftarrow 0$; $Temp_1 \leftarrow 0$;
Repeat c_4 times
for $j = -1, 1$ do (
  $Conform(j)$; $Temp_0 \leftarrow 1$
  if ($H = 0$, $H^j = 1$, $Cons_j = 2$)
  then ($H \leftarrow 1$; $Temp_1 \leftarrow 1$);
  );
(Repeal the changes if they did not close a gap.)
Repeat c_4 times
for $j = -1, 1$ do
if $\pi \neq e^j$
then (
  if ($Temp_0 = 1$, $Cons_j = 0$) then $X \leftarrow Dead$;
  if $Temp_1 = 1$
  then if (($Cons_j = 0$, $\tau < T_3$) or ($Cons_j = 2$, $H^j = 0$))
  then $H \leftarrow 0$;
);

How large is c_4? If the distance of the damage from the left end of the block is not more than c_3 then *Purge* can kill the cells between the damage and the end of the block creating a gap of size $c_1 + c_3$. It will turn out that before *Recover* can really reclaim these cells, the message $H = 0$ may be carried to the right to a distance of $6c_2$. Hence the total number of cells to restore may be

$$c_3 + c_1 + 6c_2 = 2c_1 + 8c_2.$$

If we put

$$c_2 = 2c_1,$$

this area can be recovered in c_4 steps where we put

$$c_4 = 9c_2.$$

We will find that

$$c_5 = 6c_4$$

is also an appropriate choice. This completes the definition of our procedures.

5.3. The program.

The program is thus a parallel performance of *Comp* and (5.2.2). Let us remind that the variable τ is updated continuously and we had the special rules (5.1.1-3) for *Out* and $Misc_1$.

Some turning points of our program have names.

T_1 is the time when the retreating part of the last application of *Ocp* starts.

T_2 is the start of the first application of *Integrity*.

T_3 (defined earlier) is the time after which the marking steps do not start a message $H = 0$ anymore from a place of inconsistency.

To be comfortable with the structure of the program and see that it is not too sensitive to small modifications, let us note that the choice of the earlier members in our sequence

$$c_0 < c_1 < c_2 < c_3 < c_4 < c_5 < P < T.$$

of constants imposed only lower bounds on the later ones.

Let us show now that the string p can really be chosen so as to make M_p self-simulating. It can be seen without difficulty that there is a program q and constants C_i such that for all p,

$$r_1 > C_0(|p| + \log R_0),\ r_0 > C_1 r_1,$$

the simulation

$$\mathrm{Sim}(q * \beta(r_0) * \beta(r_1) * p)$$

is a simulation of M_p on U with periods r_0, r_1. We choose now an R_1, R_0 permitting to make $r_i = R_i$ and put

$$p = q * \beta(r_0) * \beta(r_1) * .$$

Put $M = M_p$. Let (h, χ) denote the code belonging to the simulation $\mathrm{Sim}(pp)$.

Thus by the simulation Sim(pp), the medium U simulates one operation of a cell of M with R_0 operations of a block of R_1 cells. Medium M imitates a working period of an above simulating block by T steps of a group of P cells, giving rise to a code $f : S_M \to S_M^P$. To obtain $f(s)$ for a state s, we put

$$u[0]\ldots u[P-1] = \alpha h(s)h(s)h(s)\alpha,$$

and form the string $f(s) = x = x[0]\ldots x[P-1]$ as follows.

The value of each $x[n]$ will be determined if we give value to all the variables

$$X, Y, Input_i, Output_i, Mail_j, Misc_j, \tau, \pi, H, Cn, Temp_j.$$

We put $\tau[n] = 0$, $\pi[n] = n \pmod P$, $Misc_0[n] =$ something independent of n, $Misc_1[n] =$ arbitrary, $Cn = 2$, $Temp_j = 0$, $X[n] = u[n]$, and

$$Input_i[n] = Y[n] = Mail_j[n] = Output_i[n] = Dead.$$

To decode a string $x \in S_M^P$ into $s = \phi(x)$, we first try to reverse the above process, finding $h(s)$ from the three candidates by majority decoding in every symbol, then apply χ.

The entering operation $g_i(a, b)$ for $i = 0, 1$ is the operation of writing b into the $Misc_i$ variable of a, and the inverse γ_i is reading it out from there.

6. Traces

Informally, Lemma 2 asserts that if in a space-time rectangle B the set of errors is k-sparse then the activity of the cells within the "interior" of B is "organized" up to the k-th level of simulation. Of course, by "organized" we can only mean a condition which is not affected by a k-sparse set of errors. Moreover, we cannot assert that all space-time points in the interior will belong to the same organization, only that the interior can be broken up into disjoint organized islands swimming in a see of dead cells. These islands will be called "traces". They are not necessarily connected, rather they are defined by the type of their organization.

We want to incorporate into the lemma the case when B contains the edges of the domain of definition (with standard border conditions) and the case when B is not a rectangle because some bad parts were cut out from it. Therefore, more generally, we will be concerned with a triple (R, B, x) where B is a set of lattice points in space-time $\mathbf{Z}^2$, which is contained in a rectangle

$$R = [m_0 \ldots m_1] \times [n_0 \ldots n_1].$$

The function $x : \mathbf{Z}^2 \to S_M$ is defined on R with standard border conditions. The shifted triple $(a, b) + (R, B, x)$ is defined as

$$(R + (a, b), B + (a, b), z)$$

where $z[t, n] = x[t - a, n - b]$.

Informally, a *canonical k-trace* L^k is a subset of B such that within it, the function x agrees with the work of the medium M satisfactorily up to the k-th simulation level. If we shift a canonical k-trace by any nonzero vector in space-time, we get some other k-trace $\overline{L}^k$ inconsistent with the canonical one. But even canonical k-traces can be inconsistent between each other, since consistency depends not only on the values of the counter variables but also on the variable $Misc_0$. Therefore to completely specify a canonical k-trace, we have to fix a sequence

$$\Pi = \Pi_0, \ldots, \Pi_k$$

where $\Pi_i \in S_U$ is the value of $Misc_0$ required on the i-th level.

A *k-frame* $F = (a, b, \Pi)$ consists of a *base vector* (a, b) with $a \in \mathcal{T}^{k+1}$, $b \in \mathcal{P}^{k+1}$, and the *program sequence* Π. For any k-frame $F = (a, b, \Pi)$, we

define the $k-1$-frame $F' = (a', b', \Pi')$ by

$$a' = a \pmod{T}^k,$$
$$b' = b \pmod{P}^k,$$
$$\Pi'_i = \Pi_i \text{ for } i = 0, \ldots, k-1.$$

Two k-frames F and $\overline{F}$ are called *locally consistent* if $F' = \overline{F}'$. The set C_k of *canonical k-cells* is the set of all n such that

$$P^k[n] \subset (n_0 \ldots n_1).$$

We will call any n not in C_k a *partial cell.* A partial cell i is *proper* if $P^k[i] \cap (n_0 \ldots n_1) = \emptyset$.

For a k-frame $F = (a_0, b_0, \Pi)$ and a vector (a, b) we define the *shifted* k-frame

$$F + (a, b) = (a_0 + a \pmod{T}^{k+1}, b_0 + b \pmod{P}^{k+1}, \Pi).$$

A k-frame (a, b, Π) is *canonical* if $a = b = 0$. The notions of canonicity and local consistency will also be applied to the k-traces arising from the corresponding k-frames. From now on, we suppose that some canonical k-frame F has been fixed. For a noncanonical k-frame $\overline{F} = (a, b, \overline{\Pi})$, we define the working rectangle

$$\overline{V}^k[t, n] = (a, b) + V^k[t, n].$$

The sets $\overline{C}_k, \overline{L}_k[h]$ are obtained just like the set $L_k[h]$, but from the shifted triple $(R, B, x) - (a, b)$ and the canonical k-frame $\overline{F} - (a, b)$. When we speak of the work of the (canonical) k-cell n at (the period) t, we mean the work rectangle $V^k[t, n]$. If it cannot lead to confusion then we may simply speak of a canonical k-cell as a *cell.*

Each canonical k-cell i *represents* at each period h a value $x^k[h, i]$. As an element of S_M, this value is the collection of variable values $X(x^k)[h, i], \tau(x^k)[h, i], \ldots$. The function x^k will be defined recursively and simultaneously with the set $L_k[h]$ of *live* cells. We put

$$L^k = \cup\{ V^k[h, i] \ : \ i \in L_k[h] \}.$$

Put $x^0 = x$. For $k > 0$, suppose that x^{k-1}, L_{k-1} are defined already. We say that cell i is *formatted* at period h if there is an interval I of length

$c_1 + 2c_3$ such that

$$(K + iP) \setminus L_{k-1}[h] \subset I,$$

and the sequence

$$X(x^{k-1})[hT] \mid K + iP$$

can be changed on I to have the form uuu for a string u in $S_U^{R_1}$. For cells i we put $x^k[h,i] = Dead$ if i is not formatted at h, and

$$\phi(X(x^{k-1})[hT] \mid P_k[i])$$

otherwise.

Let us extend the definition of $x^k[h,i]$ to be Out for proper partial cells i, and $Dead$ for improper partial cells as well as cells i with $hT^k < m_0$. Put

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

For $k > 0$, that the cell i is *protected* in period h if the rectangle

$$[(h-1)T^k \ldots (h+1)T^k) \times [(i-2.1)P^k \ldots (i+3.1)P^k) \cap R$$

is contained in B. The 0-cell i is protected at h if (h,i) is in B. The set of k-cells protected at period h is denoted by $B_k[h]$.

The cell i is in $L_k[h]$, or it is a *live cell of* F if it is protected, formatted and

$$\begin{aligned} X(x^k)[h,i] &\in S_U, \\ \tau(x^k)[h,i] &\equiv h \pmod{T}, \\ \pi(x^k)[h,i] &\equiv i \pmod{P}, \\ Misc_0(x^k)[h,i] &= \Pi_k. \end{aligned}$$

We say that cell or partial cell i is *proper dead* at h if $P[i] \cap L_{k-1}[hT]$ can be covered by an interval of length c_3. Let $\overline{F}$ be a k-frame locally inconsistent with F. We say that the cell or partial cell n of $\overline{F}$ at t *disturbs* the cell i at h if n is not a proper dead k-cell of $\overline{F}$ at t and the set

$$T^k[h] \times [iP^k - 0.5c_5P^{k-1} \ldots (i+1)P^k + 0.5c_5P^{k-1})$$

intersects with $\overline{V}^k[t,n]$. A cell i is *proper* at h if it is either proper dead, or formatted and not disturbed (by any cell of any other frame). The behavior

of x^k is not predictable for unprotected cells and less predictable for improper cells. We will see that improper cells are a transient phenomenon.

From now on, we suppose about every triple (R, B, x) we deal with that the bottom row of R is contained in L^l for all $l \leq k$. This condition implies that the time projection of R begins at a point of the form hT^k, and the space projection is the union of whole copies of P^k.

We say that the triple (R, B, x) is *k-organized* if for any canonical k-frame F and any shift of the triple (R, B, x), the conditions (O1-O3) hold. Suppose that i is a protected k-cell in both the periods h and $h+1$.

(O1) (Regularity.)

1. Each protected cell is either proper dead or formatted.
2. The k-traces corresponding to different k-frames are disjoint.
3. If both $i-1$ and $i+1$ are protected and formatted at h then i is proper at h.
4. If an inner cell i is proper dead or undisturbed at h or $m_k[h,i] = Dead$ then it is proper at $h+1$.

(O2) (Computation.)

$x^k[h+1,i]$ is either $m_k[h,i]$ or is *Dead*. We have the former case if i is proper live at h.

The last condition tells when a k-trace is able to "advance into a no man's land". It is formulated for advance to the right, but we assume that it holds for left advance as well.

(O3) (Advance.) If $k > 0$ and $i-1 \ldots i+2$ are protected and undisturbed at h, then $x^k[h+1,i] = m_k[h,i]$.

Over a k-organized domain, we can make many assertions about the function x^k. For each h, the live k-cells form intervals. New intervals do not arise out of nothing, except possibly at the left and right ends of B. The old intervals can grow with time, shrink or break up. The inner points or new endpoints of any of these intervals are proper. 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 $m_k[h,i]$. 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 live cell i can can die at period h only if it is an improper endpoint or $m_k[h,i] = Dead$. The condition (O3) asserts that an interval can increase at a proper endpoint if it is in the appropriate phase of the working period, and other traces are not too close to block the growth.

LEMMA 2 (MAIN LEMMA) *If the set of errors is k-sparse over B then our triple is k-organized.*

The lemma is obviously true for $k = 0$. Section 8 is devoted to the proof of the Main Lemma by induction.

7. Proof of Theorems 1 and 2.

We use the medium M, the code f and the entering operations g_i defined in Section 5.

Proof of Theorem 1: Let the standard code (F, Φ) use the entering operation (g_1, γ_1) defined at the end of Section 5. Let u be a string in S_U^n. We can suppose without loss of generality that $n = P^r$ for some r. (We can "pad" the input string u if this does not hold.)

$$v = g_1(f_+^\infty, u).$$

Let ξ be a ρ-perturbation of M defined over a rectangle

$$R = [0 \dots tT^k] \times [-1 \dots nP^k],$$

with input $f_k(v)$ and standard border conditions.

It follows from Lemma 1 that for any ϵ, we can choose

$$k = O(\log\log nt/\epsilon)$$

such that the set of errors is k-sparse with probability $1 - \epsilon$. Let us thus suppose that the set of errors in our sample realization x of ξ is k-sparse. Given the input string u, the bottom of R obviously belongs to L^k. We have to prove that

$$\Phi_k(x[tT^k]) = U^t(u). \tag{7.1}$$

The Main Lemma implies that the triple (R, R, x) is k-organized. If also $R \subset L^k$ then the values $\gamma_1(x^k[h,i])$ will be the record of the work of the universal medium U with input u. Therefore (7.1) holds.

We will prove $R \subset L^k$. Suppose that, on the contrary, there is a h such that not the whole row $[0 \dots n)$ belongs to $L_k[h]$. Let h_k be the first such h. Then the cases (O1),(O2) apply to all (h,i) with $h < h_k$. If $m_k[h,i]$ is live then $i \in L_k[h]$ and $x^k[h+1,i] = m_k[h,i]$. Therefore if $h_k \le t$ then $m_k[h_k - 1, i]$ is dead for some i.

To prove that $m_k[h,i]$ is always live, we must convince ourselves that there are only two cases when our program kills a cell. First, this can happen if there is an inconsistency. But the only way the program can *introduce* an inconsistency is the same as the second way to kill a cell: namely in the last step of *Comp*, if in some cells we have $Y = Dead$. Without inconsistencies, this will

happen only if the last step of *Core* finds that in the computation *simulated* by x^k, a cell gets killed. Thus a cell can get killed in the computation x^k only in a step of the form $h_k = h_{k+1}T$, and only if a cell gets killed in the computation x^{k+1} by step h_{k+1}.

The input to our computation was the string $f^k(v)$ where $v = g_1(f^\infty_+, u)$. Therefore the strings $x^l[0]$ are consistent for all natural numbers l. Hence the above argument applies to all $l \geq k$: a cell of x^l can get first killed at step h_l only if a cell of x^l gets first killed at step h_{l+1}. There is only one $k+r$-cell in our computation, namely 0. According to our definition of f^∞, and our program, we have $X(x^{k+r})[w, 0] = \alpha$ for all w (since 0 is an endcell for the $k+r+1$ -block 0). Therefore on level $k+r$, no inconsistency ever arises. Concluding back from $l+1$ to l for $k \leq l < k+r$ we find that $m_k[h, i]$ is always live. ■

For Theorem 2, we will deal with triples (R, B, x) where $R = [0 \ldots \infty) \times \mathbf{Z}$, and the bottom of R is contained in L^l for all l. We need another lemma, which comes as a side result of the proof of the main lemma.

We define the following rectangles.

$$W_0^k = [0 \ldots 2T^k) \times [-3P^k \times 3P^k),$$
$$W_1^k = [T^k/2 \ldots 2T^k) \times [-P^k \times P^k)$$
$$W_j^k[h, i] = W_j^k + (hT^k, iP^k).$$

For Lemma 3, let (R, B, x) be triple, $k > 0$. Suppose that the set of errors is k-sparse over B and $k-1$ -sparse over the set $W_0^k[h, i] \subset B$.

LEMMA 3 *Suppose that $W_0^k[h, i] \subset L^k$, and the values $m_k[h, i-1]$, $m_k[h, i]$ are live. Then $W_1^k[h, i] \subset L^{k-1}$.*

This lemma, together with the previous one, says that if our input differs in something like a k-sparse set of errors from a string of the form $f^k(u)$, 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 errors in the input will soon be corrected. For example, the information Π_0 can be read out from every copy of V^{k-1} which is well inside the domain.

Proof of Theorem 2: Let the entering operation (g_0, γ_0) be the one defined in Section 5. 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.

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

Let x be a realization of ξ for which U holds. It remains to show that

$$Misc_0(x)[t,n] = s. \tag{7.2}$$

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 (7.2).

Let us first show that $R_{r-1} \subset L^{r-1}$. Without loss of generality, we can suppose that R_r is either W_0^r or $W_0^k[-1,0]$.

First Case: $R_r = W_0^r$. Then the bottom row of R_r is the starting line, hence it is contained in L^k for any k. We conclude from Lemma 3 that W_1^k is contained in L^{r-1}, hence also R_{r-1} is contained in L^{r-1}.

Second Case: $R_r = W_0^k[-1,0]$. Let us look at the triple (R,B,x)

$$B = R_r \cap R = [0 \ldots T^r) \times [-3P^r \ldots 3P^r).$$

The set of errors is $r-1$-sparse over B, and the input to R is contained in L^k for any k; especially, it is contained in L^{r-1}. We conclude from Lemma 2 that the triple is $r-1$ -organized.

Let us show that the rectangle

$$A = [0 \ldots T^r) \times [-2P^r \ldots 2P^r).$$

is contained in L^{r-1}. We already know that $[-2P \ldots 2P]$ consists of proper cells of $L_{r-1}[0]$. Just as in the proof of Theorem 1, we can conclude that if row $(h+1)T^{r-1}$ is the first one not contained it L^{r-1} then $m_{r-1}[h,i] = Dead$ for some i in $[-2P \ldots 2P)$. But since $0 \le h \le T-2$, the only cause of killing a cell during this time is an earlier inconsistency within the block, which we did not have. Thus we proved $A \subset L^{r-1}$, and with it, $R_{r-1} \subset L^{r-1}$.

Now we can prove that for all $k \in [1 \ldots r-1]$, if $R_k \subset L^k$ then $R_{k-1} \subset L^{k-1}$, using Lemma 3 in exactly the same way as in First Case above. Therefore by induction, we proved $R_k \subset L^k$ for all $k < r$. ∎

8. Proof of Lemmas 2 and 3

Throughout this section, we are given a triple (R, B, x) satisfying the conditions of Lemma 3. In particular, the set of errors is k-sparse over B. By induction, we assume that the lemma holds for $k-1$. We will always assume that the constant P is as large as needed with respect to the constants $c_0, c_1, \ldots$, and T as large as needed with respect to P.

It follows from k-sparsity that in any copy of $c_0 V^k$ there is a copy J of $3c_0 V^{k-1}$ such that the set of errors is $k-1$ -sparse outside J. We will call J the *error rectangle*. We will generally not explicitely specify the enclosing copy of $c_0 V^k$, but it will always be clear from the context that it can be easily chosen. Instead of J, it is more convenient to work with the intervals J_0, D_0 which we now define. Let D_0 be the interval of all those i for which the strip

$$\mathbf{Z} \times [(i-2.1)P^{k-1} \ldots (i+3.1)P^{k-1})$$

has a nonempty intersection with J. Then the length of D_0 is at most $c_1 = 3c_0 + 6$. Similarly, let J_0 be the interval of all those h for which the strip

$$((h-1)T^{k-1} \ldots (h+1)T^{k-1}] \times \mathbf{Z}$$

intersects with J. Its length is at most $3c_0 + 2$. Any interval disjoint from J_0 is called *error-free*.

8.1 The effect of *Purge* and *Recover*.

It is simple to show that the procedure *Purge* kills small islands. But for *Recover* to work, the gap it has to close must be purged of all remainders of $k-1$ -traces. This will happen since sooner or later each of these $k-1$ -traces uses its own *Purge*; but it is convenient to formalize this observation in Lemma 4.

Let a, b, t, n be integers. Let $E \subset B$ be a rectangle

$$(a, b) + [0 \ldots t] \times [0 \ldots n]$$

not intersecting with J.

LEMMA 4 *Suppose that $t > c_2 + 4c_3$. Suppose further that the left and right edges rectangle of E do not intersect with L^{k-1}, and the intersection of its bottom with L^{k-1} can be covered by an interval of length $3c_1 P^{k-1}$. Then its top does not intersect with L^{k-1}.*

Proof: Put $h = \lfloor a/T^{k-1} \rfloor$, $i = \lfloor b/P^{k-1} \rfloor$,

$$\mathcal{L}[s] = L^{k-1}[s] \cap [i \ldots i+n].$$

The intersection of L^{k-1} with the bottom of E is

$$\{a\} \times \cup\{ P^{k-1}[q] \ : \ q \in \mathcal{L}[h] \}.$$

Thus $\mathcal{L}[h]$ is contained in an interval of length $3c_1$. Let t_0 be the first $t \geq h$ where an instance of *Purge* starts. Then

$$t_0 < i + c_2 + 2c_3,$$

since a *Purge* starts after every c_2 steps and its length is $2c_3$.

The elements of $\mathcal{L}[t_0]$ can be covered by an interval of length c_3. It follows from the $k-1$ -organizedness of x over our rectangle E that after the $2c_3$ steps of *Purge*, the $k-1$ -cells in this interval, being isolated from other live k-cells, will be killed. After it, no element of $\mathcal{L}$ can arise. ■

Due to the general position of the rectangle $E + (a, b)$, Lemma 4 is applicable to any $k-1$ -trace, not only the one arising from the frame F; and it is typically applied simultaneously to *all other traces* . Put

$$c_6 = |Purge[c_2]Recover|.$$

LEMMA 5 *Let us use the notation of the previous lemma, with* $a \equiv 0 \pmod{T}^{k-1}$, $b \equiv 0 \pmod{P}^{k-1}$. *Suppose that*

$$n \leq c_4, \ t \geq c_5 + 2c_6,$$

and $a/T^{k-1} + t$ *is the end of an application of Recover. Suppose further that the edges of the rectangle* E *are contained in* L^{k-1}, *and no* $k-1$ *-trace* $\overline{L}^{k-1}$ *locally inconsistent with* F *has an intersection with the bottom of* E *longer than* $3c_1P^{k-1}$. *Then the top of* E *is contained in* L^{k-1}.

Proof: The part of E above the level

$$t_1 = a + (c_2 + 4c_3)T^{k-1}$$

does not intersect with $\overline{L}^{k-1}$ for any k-frame $\overline{F}$ locally inconsistent with F. Indeed, let $\overline{F} = (a_1, b_1, \overline{\Pi})$ be such a frame. It follows from the $k-1$

-organizedness that L^{k-1} does not intersect with $\overline{L}^{k-1}$, hence the left and right edges of E are disjoint from $\overline{L}^{k-1}$. Therefore we can apply Lemma 4 to the k-frame $\overline{F}$ and the set $\overline{E} = E - (a_1, b_1)$. Any horizontal segment through E above t_2 can play the role of the top of $\overline{E}$ for the k-frame $\overline{F}$.

After t_1, the next application of *Recover* recovers all cells between the two edges of E to L_{k-1}. Indeed, since it now finds undisturbed cells in the gap, it can rely on (O3) of Lemma 2. This application of *Recover* starts certainly before $a + c_5 + c_6$. ■

8.2. The integrity of blocks.

The goal of Lemmas 6-10 is to show that each protected cell is either proper dead or formatted. We will assume that the k-cell 0 is not proper dead at 1. Lemma 6 shows that then its $k-1$ -cells occupy the whole block for most of the working period 0. Lemma 8 says that most of the $k-1$ -cells in the block 0 have $H = 1$ at time $T-1$, i.e. will not be killed for $H = 0$ in the last step of the working period.

We will say that the rectangle $[u \dots v] \times [i \dots j]$ is *regular* if for all t in $[uT \dots (v+1)T]$ we have

$$[(i - 0.1)P \dots (j + 1.1)P) \cap C_k \subset B_{k-1}[t].$$

From now on, put

$$\mathcal{L}^j[t] = P[j] \cap L_{k-1}[t].$$

As usual, we write $\mathcal{L} = \mathcal{L}^0$. Let further $\mathcal{H}^j[t]$ denote the set of those i in $\mathcal{L}^j[t]$ for which

$$H(x^{k-1})[t, i] = 1.$$

Let us remember the definition of a *marking step* given in the procedure *Integrity*. The last $1.1P$ marking steps of the program are called *concluding*. We know that they will already only propagate a message $H = 0$ but not create one.

LEMMA 6 *Suppose that* $(0,0)$ *is regular, and* 0 *is a* k*-cell which is not proper dead at 1. Then for all but* P *values of* t *in* $[T_2 \dots T_3]$ *we have* $\mathcal{H}[t] = P \cap (n_0 \dots n_1)$.

Proof: A *gap* at time t is any contiguous interval of $P \setminus \mathcal{H}[t]$. An interval in $P \setminus \mathcal{L}[t]$ will be called a *dead gap*. The part of space-time outside the error rectangle J is $k-1$ -sparse, hence we can apply the inductive assumption there. We will say that an *endgap* opens if an improper cell dies at an end of

C. When an endgap is closed and we are outside the error rectangle then the endcell becomes proper, by (O1).

Any error-free marking step can only widen existing gaps. In an error-free application of *Recover*, all newly created cells will be repealed if they did not close a dead gap. Similarly, only *Recover* can unmark the marked cells, and it does so only if it closes a gap. Thus after T_2 and outside J_0, gaps are never diminished unless closed completely.

Put $m = c_1 + 2c_4 + 1$. Let us call a *free run* any m consecutive error-free nonconcluding marking steps. We will show that any gap disappears in the first free run after its appearance. Suppose that a gap persists throughout some free run. Then the size of this gap will be at least m by the end of this run. Any error-free marking step coming after the appearence of this large gap increases it by at least 1. The effect of the error rectangle can decrease it by at most $c_1 + c_4$ (changing a piece of size c_1 and cutting off by it a piece of size $3c_2$ which can be recovered). Since this still leaves a gap larger than c_4, it will not be recovered completely. Therefore since there are at least $1.1P$ more marking steps after our free run, at least P of them widen all possible gaps, hence the gap will eventually cover the whole P. An error coming after this time can create an island of size c_3 but not larger, because the first application of *Purge* erases this island again. The last step of the program kills all cells in $\mathcal{L}[t] - \mathcal{H}[t]$. Thus if a gap persisted in our free run then 0 would become a proper dead k-cell at 1.

Suppose now that no gap persists in a free run. It is easy to see that there are only a few kinds of gap, and since each quickly disappears after its appearance, the lemma will be proved. The original inner gaps must disappear in the first free run. A further inner gap can be opened by the error rectangle, but it will also be closed by the next free run. A left endgap may also show up. When it disappears, it is replaced by a proper endcell, hence it can reappear only due to the error rectangle. But after this, it disappears forever. The same applies to the right end.

Thus, apart from the error interval plus a few contaminated free runs, gaps may appear only before T_2 and after T_3. ■

Notice that this proof is also applicable to the case when 0 is an improper partial k-cell. We can conclude that then 0 is *always proper dead at 1.*

For the next two lemmas, suppose that the conditions of Lemma 6 hold. Lemma 7 is a preparation for Lemma 8. Let $t > T_2$ be such that J_0 does not occur between T_2 and t. Suppose that t is u marking steps away from the end of the program.

LEMMA 7 *The set $\mathcal{L}[t]$ is an interval containing $[1 \ldots P-2]$. Further, $\mathcal{H}[t]$ is an interval which either contains $\mathcal{L}[t]$ or is at most $u - 0.1P$ long.*

Proof: From the proof of the previous lemma we know that there is a time before $T_2 + 0.1P$ at which $\mathcal{H} = P$. After this time and until the error, a gap can arise only if an improper endcell dies. Suppose that this happens on the left end. If it happens before T_3 then the marking on the cells begins to be propagated from the left end by every marking step. Until the marking travels farther than c_4, two events can interrupt it. First if the endcell gets restored. Second, if τ becomes larger than T_3. In both cases, *Recover* unmarks the marked cells. Of course, similar events may occur on the right side. If the marking travels farther than c_4 then these events do not change it anymore, and the first v concluding marking steps mark at least v cells (until there is any unmarked cell left). If an improper endcell dies after T_3 this event does not create any new marked cells. ■

LEMMA 8 *Under the assumptions of Lemma 6, there is an interval D of length*

$$c_1 + c_2 + c_3$$

containing D_0 such that

$$[1 \ldots P-2] \setminus D \subset \mathcal{H}[T-1].$$

If $\{0\} \times \{0,1\}$ is regular and 1 is not a proper dead at 1 then $P-2$ can be replaced by $P-1$ here.

Proof: Let t_0 be the time when J_0 occurs, t_1 the time of the first application of *Purge* after J_0. Then

$$t_1 < t_0 + c_1 + c_2.$$

Let t_2 be the time of the first application of *Recover* after t_1, and

$$t_3 = t_2 + c_6$$

be the end of this *Recover*.

Let us first prove that there is an interval D_1 of size $c_1 + c_3$ such that

$$K \setminus D_1 \subset \mathcal{L}[T-1]. \tag{8.2.1}$$

By the previous lemma we have $[1 \ldots P-2] \subset \mathcal{L}[t_0]$. The effect of J_0 can kill cells within D_0. Further cells can be killed only by *Purge*, and this happens

only if there are c_3 or less cells between D_0 and one of the endcells. Then *Purge* may kill these intermediate cells. If the endcell is improper and dies then the joint effect of these two events can kill cells within an interval of maximal size $c_1 + c_3$ in K. This proves (8.2.1).

Suppose that $\mathcal{H}[t_0]$ is empty. Then new cells of $\mathcal{H}$ can only arise in D_0. Before the next *Purge*, these can be propagated within an interval of size

$$c_1 + 2(c_1 + c_2) = c_3.$$

No further growth is possible. Therefore 0 is a proper dead k-cell at 1.

Suppose that $\mathcal{L}[t_0] = \mathcal{H}[t_0]$ and $u < 1.1P$. Then cells of $\mathcal{H}$ can be erased only either by the death of an improper endcell, or by the error. The former event now does not lead to further decrease of $\mathcal{H}$ since our marking steps are the concluding ones. The error can erase the elements of $\mathcal{H}$ in D_0. If this happened closer than c_3 steps to a perished endcell then the intermediate cells can be killed by *Purge*. At the same time, markings can be propagated into the block to c_2 further cells before the arrival of the next *Recover*, making the maximum size of the damage in K as large as

$$c_1 + c_2 + c_3.$$

Since *Recover* unmarks all marked cells that belong to a large interval of $\mathcal{H}$, the damage does not grow any further.

Suppose that $T < t_2$. Then since *Recover* gets applied every c_5 steps and is c_6 steps long, we have $t_0 > T - c_5 - c_6$. By the previous lemma, then either $\mathcal{H}[t_0]$ or $\mathcal{L}[t_0] - \mathcal{H}[t_0]$ is empty. We have considered these two cases in the previous paragraphs.

Now we can suppose that $t_2 < T$, i.e. that an application of *Recover* is left between t_0 and T.

We first show that $\mathcal{L}[t_3]$ is an interval. The set $\mathcal{L}[t_0] - D_0$ consists of two (possibly empty) intervals I_0 and I_1. We can suppose without loss of generality that I_1 is longer. Only *Purge* can erase cells outside D_0, and if this happens then it erases the whole of I_0 or I_1.

Suppose that both I_0 and I_1 exist by the time t_3. Since the gap between I_0 and I_1 is D_0 and thus short, it will be closed, as shown in Lemma 5. Thus $\mathcal{L}[t_3]$ contains K.

If I_0 disappears by t_3 then it is shorter than $c_3 + 1$ cells, hence the number of cells outside I_1 is at most $c_1 + c_3 + 2$. It follows that after *Purge*, the cells of $\mathcal{L}$ outside I_1, if there are any, will be either contiguous with I_1 or form one

long interval I. If I survives until t_3 then by Lemma 5, it will be joined to I_1. We thus proved that $\mathcal{L}[t_3]$ is always an interval.

The gap between the two (possibly empty) intervals of $\mathcal{H}[t_0] - D_0$ can by the time t_2 increase to the size c_3. Nevertheless, it is easy to show by an argument similar to the one used for $\mathcal{L}$, that $\mathcal{H}[t_3]$ is an interval. Obviously, if P is large enough then the size of the difference between $\mathcal{H}[t_0]$ and $\mathcal{H}[t_3]$ is bounded by $0.05P$.

Suppose that $\mathcal{H}[t_0] \neq \mathcal{L}[t_0]$. Then by Lemma 7 we know that the number of cells in $\mathcal{H}[t_0]$ is at most $u - 0.1P$. If $u \leq 0.1P$ then $\mathcal{H}[t_0]$ is empty. This case was considered in a previous paragraph. Suppose that $u > 0.1P$. Then the size of $\mathcal{H}[t_3]$ is at most $u - 0.5P$. Since after t_3 there are still at least $u - 0.5P$ marking steps left, by the time $T - 1$ they mark all cells of I, and $\mathcal{H}[T-1]$ is empty.

Suppose that $\mathcal{H}[t_0] = \mathcal{L}[t_0]$. The case $u < 1.1P$ was discussed in a previous paragraph. On the other hand, if $u \geq 1.1P$ then the discussion of the preceding lemma can be applied to the events after t_3, and we can conclude that $\mathcal{H}[T-1]$ is either empty or contains $\mathcal{L}[T-1]$. ■

8.3. Occupation and computation.

Let us examine the work of the procedure *Ocp*. For $j \in [-2 \ldots 2]$, let $G^j[t]$ denote the set of elements n of $L_{k-1}[t]$ with the property that

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

The set $G^j[t]$ is an extension of the proper elements of the k-cell j by its "occupying arms".

For Lemma 9, suppose that $\{0\} \times \{0, 1\}$ is regular and 0 is a proper dead k-cell at 0.

LEMMA 9 *For $j = -1, 1$, $t \notin [t_0 \ldots t_3]$ the set $G^j[t]$ is an interval. If also $t < T_2$ then $G^0[t]$ is empty.*

The size of $\mathcal{L}[t_3]$ differs from the size of $\mathcal{L}[t_0]$ by at most $0.05P$ cells. If $t_3 < T_4$ then the size of the interval $G^j[t_3]$ differs from the size of $G^j[t_0]$ by at most $0.05P$.

The proof of Lemma 9 is similar to the proof of Lemma 8. This lemma enables us to reason about the growth intervals directly in terms of the program, knowing that the the intervention of the error will not change the situation greatly.

The first property in (O1) says: "Each protected cell is either proper dead or formatted." We prove this in the next lemma.

LEMMA 10 *Suppose that the set* $\{-1,0\} \times \{-1,0,1\}$ *is regular. Then 0 is either a proper dead or a formatted k-cell at 1.*

Proof: It follows from the proof of Lemma 6 that if 0 is not proper dead at 1 then that there is a $t_4 < T_2 + 0.1P$ such that $\mathcal{L}[t_4] = P$.

Suppose first that J_0 occurs before t_4. Then the three identical computational parts of the program in its application in $V[1,0]$ (on x^{k-1}) are undisturbed by any error within K. Thus in applications $s = 1,2,3$, three copies of the sequence u_s are written to $Output_s$ in the three thirds K_1, K_2, K_3. If any element of u_s is *Dead* then all of its elements are *Dead*, since an errorless computation produces such outputs. We cannot claim any relation among the three sequences u_1, u_2, u_3, only that $Output_s|K$ will be $u_s^3 = u_s u_s u_s$. The final step of the program will leave us with dead cells wherever $H = 0$ and with the result of cell-for-cell voting among the three strings u_1^3, u_2^3, u_3^3. The result of this voting may be a string which does not "code" anything, but its three parts will be equal, and it is either all dead or all live. As shown in Lemma 7, if an improper endcell dies and this does not kill the whole block then it does not kill any other cell.

Let us suppose now that J_0 does not occur earlier than t_4. Then the previous paragraph can be applied to the work of each of the k-cells among $-1,0,1$ at the period -1. Hence each of these k-cells is either proper dead or formatted at 0.

Let us look at the three blocks $P[j]$ at time T_1. If j is a cell not proper dead at 0 then $X(x^{k-1})[T_1]$ has the desired triple structure over $K + jP$. If j is proper dead at 0 then its live $k-1$ -cells all belong to two occupying arms of the neighbor blocks, and we have

$$X(x^{k-1})[T_1, n] = d[n]$$

in cell n. If these occupying parts do not meet by T_1 then the gap between them will be widened so much by the retreating part of the last *Ocp* that *Integrity* will kill the block by the time $2T$. This cannot happen for $j = 0$.

Thus at time T_1, each k-cell among $-1,0,1$ either has the information in $X(x^{k-1})$ in the desired triply redundant form or has the (triply redundant) code of a dead cell in each of its live $k-1$ -cells. Moreover, we have $P \subset \mathcal{L}[T_1]$.

To make sure that the computation can make proper use of this information, let us notice that if 1 is formatted at 0 then

$$[1 \ldots 2P-2] \subset \mathcal{L}[T_1].$$

Indeed, if 0 is also formatted at 0 then according to Lemma 6, already $L^{k-1}[T]$ contains $[1 \ldots 2P-2]$ and this situation does not change until the appearence of the error. If 0 is proper dead then the right occupying arm in P must be a continuation of $\mathcal{L}^+$.

Similar reasoning shows that if 1 is a proper dead k-cell at 0 then

$$[P \ldots 2P-2] \cap L_{k-1}[T_1]$$

consists of two intervals. Thus $L_{k-1}[T_1]$ contains an interval containing K, which we can call the *input interval.* It covers the live neighbors of the block 0.

Having the desired input to the computational part of the program, it is not difficult to see that it comes up with the desired form of 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 $c_1 + c_3$ cells of the input interval. If these cells are well inside the input interval then Lemma 5 implies that they will be restored to L_{k-1} in at most $c_5 + 2c_6$ steps. The error may also affect the $X(x^{k-1})$ values in a short interval. But due to the triple redundancy, these errors will be suppressed by voting. 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 of the program, the input coming from the input interval is therefore restorable by voting. This is true even if e.g. the k-cell 1 was proper dead at 0 (e.g. because it is an improper partial cell). In this case its block may not be covered by the input interval. If the error-free reading part encounters a discontinutiy in block 1 it will record *Dead*, and *Init* implies from this correctly that 1 is dead. If no discontinuity is encountered then all but a very small interval of the block 1 has $X[n] = d[n]$, from which it will again be concluded that 1 is dead.

Thus in the error-free thirds, the *Output* values computed will be equal to the same triply redundant string (with the possible exception of the interval D_1). The final voting produces the desired result. ■

Examining the previous proof yields us some additional facts.

LEMMA 11 *Suppose that* $[-2 \ldots 1] \times \{-1, 0, 1\}$ *is regular and 0 is not a proper dead k-cell at 1. Then there is an interval D of length* $2.5c_4$ *containing* D_0 *such that*

$$K - D \subset \mathcal{L}[t] \tag{8.3.1}$$

for t in $[T_1 \ldots 2T-1]$.

Proof: By Lemma 10, the k-cell 0 is either proper dead or formatted at 0. In the first case, (8.3.1) holds for t in $[T_1 \dots T)$. Indeed, if the two occupying arms were further apart than $2.5c_4$ by this time, they would no longer grow to meet later, and the gap is too big to be closed by *Recover*. This would cause block 0 to die by Lemma 6.

In the second case, (8.3.1) holds for all t in $\mathcal{T}$; indeed, nothing diminishes $\mathcal{L}$ in this interval (or causes a large gap in $\mathcal{L}[0]$) but the error rectangle or an improper endcell. This argument also extends the validity of (8.3.1) to $[T_1 \dots 2T-1]$. ■

LEMMA 12 *Suppose that* $[-2 \dots 1] \times [-2 \dots 2]$ *is regular. Then* $x^k[1,0]$ *is either* $m_k[0,0]$ *or Dead. In the latter case, the k-cell 0 is proper dead at 1.*

Proof: It follows from Lemma 10 that each of the k-cells $-1, 0, 1$ is either proper dead or formatted at 0. Now we can follow the part of the proof of Lemma 10 which concludes all the assertions of our lemma from this assumption. True, in that proof we also knew that J_0 does not occur until T_2. However, we did not use this fact in a significant way. If J_0 occurs earlier then the error can either open a gap in the set L_{k-1} which, by Lemma 5, will be closed in due time, or change the length of an interval of L_{k-1} by a constant amount. None of these will affect the input to the computation significantly. ■

8.4. The disjointness of k-traces.

The following lemma proves that the occupation procedure never brings a k-trace into contact with a k-trace with which it is locally inconsistent. Let $\overline{F}$ be a k-frame locally inconsistent with F. Suppose that $\mathcal{T}^k$ has a nonempty intersection with $\overline{\mathcal{T}}^k[t]$ and the intervals $\mathcal{P}^k$ and $\overline{\mathcal{P}}^k[n]$ are less than $c_5/2$ steps apart (i.e. the rectangles V^k and $\overline{V}^k[t,n]$ "disturb" each other).

LEMMA 13 *Suppose that either 0 is a proper dead cell of* F *at 0 or* n *is a proper dead cell of* $\overline{F}$ *at* t. *Then either 0 is a proper dead cell for* F *at 1 or* n *is a proper dead cell of* $\overline{F}$ *at* $t+1$.

Proof: We can suppose without loss of generality that $t = n = 0$, and

$$\overline{\mathcal{T}}^k = a + \mathcal{T}^k,\ \overline{\mathcal{P}}^k = b + \mathcal{P}^k$$

where $a \in \mathcal{T}^k$ and $b \in [0 \dots P^k + 0.5c_5P^{k-1})$. Since we want to prove the contrary, we can also suppose that 0 is formatted at 1 both in F and $\overline{F}$.

In this case, it follows from Lemma 11 that for some intervals D, $\overline{D}$ of length $2.5c_4$ the set

$$\{u\} \times P^{k-1}(K \setminus D) \tag{8.4.1}$$

is contained in L^{k-1} for all u in $T^{k-1}[T_1 \ldots 2T)$, and

$$\{u\} \times b + P^{k-1}(K \setminus \overline{D}) \tag{8.4.2}$$

is contained in $\overline{L}^{k-1}$, for all u in

$$a + T^{k-1}[T_1 \ldots 2T).$$

In the case of $b \leq P^k/2$ we can immediately arrive at a contradiction from here. Indeed, put $u = a + T_1 T^{k-1}$. Then the two sets (8.4.1) and (8.4.2) have a large intersection, which contradicts the disjointness of L^{k-1} and $\overline{L}^{k-1}$. Hence b is in $(P^k/2 \ldots P^k + 0.5c_5 P^{k-1})$.

Let us suppose first that 0 is a proper dead cell for $\overline{F}$ at 0. Then cell 0 of $\overline{F}$ can come to life in the period 0 only by the procedure *Ocp* of its right neighbor. Indeed, -1 is a proper dead cell of $\overline{F}$ by a reasoning analogous to the one in the preceding paragraph. We can now apply Lemma 9 to the interval consisting of the block $\overline{P}^k$ and its two neighbor blocks. We get that if 1 is also a proper dead cell of $\overline{F}$ at 0 then the occupying arms $\overline{G}^{-2}$ and $\overline{G}^2$ intruding from the ends of the interval $b + [-P^k \ldots 2P^k]$ never meet. Thus 1 is a formatted cell of $\overline{F}$ at 0 and the originator of the growth interval $\overline{G}^+$ which eventually overtakes $\overline{P}^k$.

However, $\overline{G}^+$ can grow only about $P + c_5/2$ steps to the left, because then it meets the set (8.3.1). The last retraction part of *Ocp* will therefore retract the growth interval to a size $P - c_5/2$. The set $\overline{L}$ will no more be able to cover $\overline{P}$, hence *Integrity* marks all occupied cells. The error rectangle can only change the length of the intervals encountered here by an amount less than $c_5/2$, hence does not change the validity of this reasoning.

Let us now suppose that 0 is not a proper dead k-cell at 0. By our assumption, 0 is not a proper dead $\overline{F}$-cell at 1. Therefore we have (8.3.1) for all u in $a + [0 \ldots 2T)$. Since we assumed that 0 is a proper dead cell at 0 for either F or $\overline{F}$, it follows that 0 is a proper dead cell at 0 for F. But then we can repeat the argument of the preceding paragraph showing that the occupation procedure which would bring the canonical cell 0 to life will balk at the sets (8.4.2). ■

The second property listed in (O1) is proved in the following lemma.

LEMMA 14 *Any two different k-traces are disjoint.*

Proof: Suppose that the different k-traces are not disjoint. Then we can suppose without loss of generality that there is a k-frame $\overline{F} = ((a,b),\overline{\Pi})$ such that our canonical k-trace L^k has a nonempty intersection with the k-trace $\overline{L}^k$.

If $a' = b' = 0$ then the k-frames F and $\overline{F}$ differ only in values which can be computed from the function $x[h]|P^k[i]$ whenever i is formatted for F. This makes L^k and $\overline{L}^k$ disjoint by definition. Suppose therefore that either a' or b' differs from 0.

We can assume without loss of generality that L^k and $\overline{L}^k$ intersect in such a way that $V^k[1,0]$ is contained in L^k while $\overline{V}^k[1,0] = (a',b') + V^k[1,0]$ is contained in $\overline{L}^k$ and intersects with $V^k[1,0]$. We will arrive at a contradiction from this assumption.

We assumed that 0 is in both $L_k[1]$ and $\overline{L}_k[1]$. It follows from Lemma 13 that 0 is not a proper dead k-cell at 0 for either F or $\overline{F}$. It follows from Lemma 6 that

$$\{u\} \times P^k \subset L^{k-1}$$

for most elements u of the set $[0 \ldots 2T^k]$, while

$$\{u\} \times (b' + P^k) \subset \overline{L}^{k-1}$$

for most elements u of the time segment

$$a' + [T_2T^{k-1} \ldots T_3T^{k-1}).$$

Since P^k and $b' + P^k$ have a nonempty intersection and the latter time segment is contained in the former one, we obtain a contradiction with the disjointness of the sets L^{k-1} and $\overline{L}^{k-1}$. ■

8.5. Proper cells.

The next unproved statement of (O1) reads: "If $i-1, i, i+1$ are in $B_k[h]$ and $i-1, i+1$ are formatted at h then i is proper at h."

If i is proper dead at h there is nothing to prove. Otherwise, all three cells in question are formatted at h. Thus a disturbing rectangle $\overline{V}^k[n,t]$ would intersect with one of the three rectangles $V^k[h,i]$ $(j = -1, 0, 1)$. But this is excluded by Lemma 14.

The next unproved statement reads: "If i is proper or undisturbed at h or $m_k[h,i] = Dead$ then it is proper at $h+1$' '. Without loss of generality, let us

take $h = i = 0$. If $m_k[0,0] = Dead$ then our statement follows from Lemma 12.

Suppose now that 0 is proper dead at 0. We can also suppose that 0 is formatted at 1, otherwise it is proper dead and we are done. We must prove that 0 is undisturbed at 1. Suppose that, on the contrary, there is a k-frame $\overline{F}$ locally inconsistent with F and t, n such that the rectangle $\overline{V}^k[t,n]$ disturbs the rectangle $V^k[1,0]$. It follows from Lemma 13 that since 0 is proper dead at 0 and formatted at 1, the cell n of $\overline{F}$ must be proper dead at t. Hence 1 is undisturbed at 1.

The case remains when 0 is undisturbed at 0. Then, using the notation of the previous paragraph, n must be a proper dead k-cell at $t-1$ for $\overline{F}$. It again follows from Lemma 8 that it must be proper dead at t too. Thus 1 is an undisturbed k-cell at 1.

The next unproved statement is (O2). From Lemma 13 we know that $x[h+1,i]$ is always either $m_k[h,i]$ or *Dead*. We must show that if i is proper live at h then $x^k[h+1,i]$ is $m_k[h,i]$. It is easy to see that all turns on the following fact.

LEMMA 15 *Suppose that* $\{-1,0\} \times [-2 \ldots 2]$ *is regular, and 0 is an undisturbed formatted k-cell at 0. Then there is an interval I of length* c_4 *such that* $\mathcal{H}[t]$ *contains* $P \setminus I$ *for all t in* $[0.1P \ldots T-1]$.

Proof: Since the k-cell 0 is formatted at time 0 there is a time $v > -0.5T$ such that $\mathcal{H}[v] = P$. In general, we want to show that for any v in $[-0.5T \ldots T-1]$, if $\mathcal{H}[v] = P$ and $\mathcal{H}[v+1] \neq P$ then we will have

$$H[v + |Ocp|] = P,$$

i.e. the set $\mathcal{H}$ soon recovers from any damage. We already know from Lemma 5 that any gap deep in the interior of $\mathcal{H}$ will be closed soon.

Here we want to see how the k-cell 0 can close a gap at the boundary, before the procedure *Integrity* widens the gap too much. The only obstacle to closing the gap may be if the $k-1$- cells near the end of P are disturbed, thus hindering the procedure *Recover*. We must thus understand what can be implied about these potential disturbances on level $k-1$ from the assumption that the k-cell 0 is undisturbed at 0.

Let $\overline{F} = (a, b, \overline{\Pi})$ be a k-frame locally inconsistent with F. Suppose, without loss of generality, that $a \in \mathcal{T}^k$ and

$$-c_5 P^{k-1}/2 \in b + P^k.$$

It follows from the assumption of the undisturbedness of the k-cell 0 of F at 0 that the k-cells 0 and 1 of $\overline{F}$ are proper dead at 0 and -1. Let us apply Lemma 9 to the cells 0 and 1 of $\overline{F}$ in periods -1 and 0. Let $\overline{t}_0, \overline{t}_1, \ldots$ denote the quantities corresponding to $t_0, t_1, \ldots$ in $\overline{F}$.

It follows from Lemma 9 that for any $t \not\in (\overline{t}_0 \ldots \overline{t}_3)$, the cells of

$$\overline{L}_{k-1}[t] \cap [b \ldots b + 2P - 1]$$

belong to a growth interval $\overline{G}^j[t]$ $(j = -1 or -2)$ on the left of $b + P$ and a similar growth interval on the right of $b + P + P$. If $\overline{G}^j[t]$ is nonempty then we say that $\overline{F}$ is *threatening* on the left at time $a + T^{k-1}t$. In this case, $G^j[t]$ is the right "occupying arm" of cell j of $\overline{F}$. Let

$$\Gamma[t] = b + G^j[t]P^{k-1}$$

denote the actual area occupied by the threatening cells.

Obviously, if $\overline{F}$ threatens on the left at time u then no k-trace locally inconsistent with $\overline{F}$ can threaten on the left until the cell j of $\overline{F}$ is alive, i.e. at least until $u + T^k/2$. The threat itself arises only in the time intervals $[0 \ldots T_1 T^{k-1}] + a$ and

$$[0 \ldots T_1 T^{k-1}] + a - T^k.$$

Hence, most of the time there is no threat. If there is no threat and $t \not\in [t_0 \ldots t_3]$ then the $k-1$- cells in P are undisturbed.

It follows from Lemma 6 that we have $\mathcal{H}[t] = P$ for most values of t in $[T_2 - T \ldots 0]$. Since also for most of these t the elements of P are undisturbed, the left and right endcells of P are proper. Hence the only way that $\mathcal{H}$ can decrease is by the occurrence of J. We can thus suppose that $\mathcal{H}[t_0] = P$ and that D_0 occurs closer than c_3 to the left end of P.

It follows from the undisturbedness of cell 0 of F at 0 that if $\overline{t}_0$ comes after the retreating steps of the last *Ocp* in the program, then $\Gamma[\overline{t}_0]$ does not reach $-0.5c_5P^{k-1}$. Let us put $u = c_5$ in this case. We put $u = i$ if $\overline{t}_0$ is the i-th retraction step of the current *Ocp*, and $u = 0$ in all other cases. Then the distance of $\Gamma[\overline{t}_0]$ from 0 is at least $\min(u, c_5/2)P^{k-1}$.

Suppose first that $u > 0$. If $\overline{t}_0 + c_1$ is a retreating step then let $\overline{t}_4$ be the last retreating step after it, otherwise $\overline{t}_4 = \overline{t}_0 + c_1$. Then it is easy to see that the distance of $\Gamma[\overline{t}_4]$ from 0 is at least $(c_5/2 - 3c_1)P^{k-1}$.

Suppose that $u = 0$. Let $\overline{t}_4$ be now the last one of the next group of retreating steps after $\overline{t}_0$. Then the distance of $\Gamma[t_4]$ and 0 is at least $c_5P^{k-1}/2$.

The retreating steps of *Ocp* are followed by $2c_5$ idling steps. Thus we have a period of length $2c_5$ (dovetailed properly with *Recover* and *Purge*) when Γ remains at a distance $(c_5/2 - 3c_1)P^{k-1}$ from 0. During this period, the frame F has an execution of *Recover*, which can recover the damaged left end of $\mathcal{L}$ undisturbedly, provided the damage has not grown too large by this time. How large can this damage be? It grows fastest if c_2 steps of *Integrity* are performed in every period of length c_5 between instances of *Recover*. Since in the worst case we had to wait $5c_5$ steps of *Ocp* in $\overline{F}$, it could mean already $6c_2$ steps of *Integrity* in F. Adding this to the instant end damage of maximum $c_1 + c_3$, we get the upper bound

$$c_1 + 6c_2 + c_3.$$

Our choice for the length of *Recover* made it possible to recover from a damage of this size. ■

From the knowledge accumulated by now, the proof of (O3) and Lemma 3 is straightforward.

9. Conclusions

Both the construction and the following analysis of the reliability of our medium are disturbingly complicated. This is especially striking in view of the fact that several possible improvements (to decrese the number of states of the medium, or the size of the working period) were sacrificed in an attempt to keep the construction transparent. The number of states could be, for example, radically decreased if we stored most of our variables in small blocks next to each other instead on top of each other. This can be done without dropping the nearest-neighbor interaction.

For me, the ergodicity problem of one-dimensional media is attractive just because despite its simple formulation, it seems to require such a monstrous solution. The question is open whether a much simpler solution exists. I consider any work toward simplification (even if it only means decreasing the number of states) very interesting. A bottleneck seems to be the necessity to simulate a universal medium, since no really simple one-dimensional universal cellular automaton is known (in contrast to 2 dimensions, where e.g. the "Game of Life", with two states and nearest-neighbor interaction, is close to the ideal.

The medium M is flexible enough to permit small changes without losing reliability. For example, if we prefer two states but permit a longer (constant) range of interaction, this can be done almost mechanically.

A less trivial change which also seems possible is to introduce continuous, instead of discrete, time as a more realistic one from a physical point of view. In such a model, the transition of each cell to the next state occurs at a random time with exponential distribution. It is not immediately clear how we can miss synchronization in our model. But it turns out that we can force just enough local synchronization on our cells, if we permit greater and greater synchronization slacks between our blocks as we rise in the hierarchy. I intend to elaborate this construction in a later paper.

To achieve logarithmic time redundancy and almost constant space redundancy, we have to write a program which resists more than one error in a working rectangle. It turns out that a working rectangle of size e.g. $r^4 \times r^3$ can cope with r errors. The occurrence of more than r errors in such a rectangle is already exponentially improbable. This permits wider spacing for the levels in the hierarchy. The sequence $M_1, M_2, \ldots$ of media where M_i simulates M_{i+1} will consist of different media, and the blocksize on level $i+1$ is an exponential function of the blocksize on level i. The small error-probability makes algebraic coding methods profitable (to replace the simple-minded tripling), and provides for dense information-packing. To minimize space-redundancy, we can trade

time for space if we let many cells share e.g. one mailbox. The details will be given in a next paper. At this point, it seems possible to have a space-factor which grows slower than any unbounded recursive function of N. The question whether it can be made constant remains open.

To me, the philosophically most challenging question is whether we can avoid the use of f^{∞}, at least in the case when the input to our computation is just a few bits. Technically, this requires a medium which creates the hierarchical simulation out of "scratch", at a random place and time. Thus, a medium which exhibits self-organization, not only the maintenance of an existing organization. Our medium M definitely lacks this property, since in it, any small group of cells not part of a consistent organization kills itself. This property must thus be changed, but cautiosly enough to still preserve error-correction. One can e.g. permit slow growth to such a group of cell, with suicide only if the growth is inhibited. The details, and especially the analysis, require much further work. A result of this kind will have the following consequence in the technical language of nonergodic media. There is a one-dimensional medium with the property that it has two different invariant measures which are also space-homogenous.

REFERENCES

[D 77] Dobrushin R.L., Ortyukov S.I.: Lower Bound for the Redundancy of Self-Correcting Arrangements of Unreliable Functional Elements. *ibid.* 13/1 (1977) 59-65.
Upper Bounds on the Redundancy of Self-correcting Arrangements of Unreliable Elements. *Problems of Inf.Transm* 13/3 (1977) 201-218.

[G 78] Gács P., Levin L.A., Kurdyumov G.L.: One-dimensional Homogeneous Media Dissolving Finite Islands. *Problems of Inf. Transm.* 14/3 (1978) 92-96.

[Gr 82] Gray L., Griffeath D.: A Stability Criterion for Attractive Nearest Neighbor Spin Systems on Z. *The Annals of Probability* 10 (1982) 67-85.

[H 75] Harao M., Noguchi Sh.: Fault Tolerant Cellular Automata. *J. of Comp. and Sys. Sci.* 11 (1975) 171-185.

[Kr 78] Kurdyumov G.L.: An Example of a Nonergodic Homogeneous One-dimensional Random Medium with Positive Transition Probabilities. *Soviet Math.Dokl.*19 (1978/1) 211-214.

[Kz 73] Kuznietsov A.V.: Information Storage in a Memory Assembled from Unreliable Components. *Problems of Information Transm.* 9/3 (1973) 254-264.

[L 76] Liggett T.M.: The Stochastic Evolution of Infinite Systems of Interacting Particles. *Lecture Notes on Math.* 598 Springer 1976.

[N 75] Nishio H., Kobuchi Y.: Fault Tolerant Cellular Spaces. *J. of Comp. and Sys. Sci.* 11 (1975) 150-170.

[S 80] Snell L.: Personal communication.

[Ta 68] Taylor M.C.: Reliable Information Storage in Memories Designed from Unreliable Components, *Bell Syst. Tech. J.* 47/10 (1968) 2299-2337.
Reliable Computation in Computing Systems Designed from Unreliable Components, *ibid.* (1968) 2339-2366.

[T 74] Toom A.L.: Nonergodic Multidimensional Systems of Automata. *Problems of Information Transm.* 10 239-246.

[Ts 76] Tsirel'son B.S.: Reliable Information Storage in a System of Locally Interacting Unreliable Elements. In "Interacting Markov Processes in Biology. *Lecture Notes on Math.* 653 Springer 1978.

[vN 52] von Neumann J.: Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components. *Automata Studies* (Shannon, McCarthy, eds.) Princeton Univ.Press NJ 1956.