Causal Nets or What Is a Deterministic Computation?[*]

Péter Gács[†]
Computer Science Department
University of Rochester
Rochester, NY 14627

Leonid A. Levin[‡]
150-3 Kenrick st.
Boston, MA 02135

Abstract

The network approach to computation is more direct and "physical" than the one based on some specific computing devices (like Turing machines). However, the size of a usual—e.g., Boolean—network does not reflect the complexity of computing the corresponding function, since a small network may be very hard to find even if it exists. A history of the work of a particular computing device can be described as a network satisfying some restrictions. The size of this network reflects the complexity of the problem, but the restrictions are usually somewhat arbitrary and even awkward.

Causal nets are restricted only by determinism (causality) and locality of interaction. Their geometrical characteristics do reflect computational complexities. And various imaginary computer devices are easy to express in their terms. The elementarity of this concept may help bringing geometrical and algebraic (and maybe even physical) methods into the theory of computations. This hope is supported by the group-theoretical criterion given in this paper for computability from symmetrical initial configurations.

0 Introduction

In this work, we propose a framework unifying various aspects of the theory of complexities of information processing—also providing a language for some new problems. Presently, many results below the level of abstraction provided by Blum's axiomatic theory are seemingly dependent on specific machine models (Turing machine, RAM, iterative network, etc.) or formulated in such models with some comment on the measure of independence of the model. This leads to unnecessary specification and to awkward formal constructions unusual in traditional mathematics.

We take the notion of computation itself as a primitive (causal net[1]) instead of considering the work of a device performing this computation. Such an approach is less detailed since the same computation can be implemented in various ways: on different devices, sequentially or parallelly, varying the order of the operations and their distribution over parts of the device. Because of its potential for the avoidance of details, we hope to set up a more unified framework providing simpler definitions and still preserving concreteness and elementarity. A causal net can be interpreted as the time-space history of all elementary operations accomplished in the computing process, with their mutual dependencies indicated. As an additional advantage of this approach, a computation on each input is regarded as a separate finite object independently of the context of a function over an infinite domain. In this way, we hope to facilitate the

[*] *Information and Control* 51(1), 1981.

[†] Part of this work was done while this author was visiting Johann Wolfgang Goethe University, Frankfurt am Main in 1978 and Stanford University in 1979.

[‡] The research of this author was supported by Boston University, Massachusetts Institute of Technology and NSF grants MCS 77-19754 and MCS-8104211.

[1] N. V. Petri in [2] is different from the inventor of Petri nets— which have no essential relation to our causal nets.

application of geometric and algebraic methods in complexity theory, and to preserve the advantages of the theory of Boolean circuits.

Unlike other types of nets (e.g., Boolean circuits) the causal net constructs its logical structure in the process of computation and thus it can be reconstructed from its input and the structure of the possible neighborhoods in it (causal structure). All operations needed for this are taken into account. At a fixed causal structure (playing the role of the program of the algorithm) the input nets can be arbitrarily large. At a given size of the input, the size of the causal nets is a complexity of computation in the usual sense (most similar to the product of time and space) in contrast to the size of the Boolean circuits which is bounded (by $2^n/n$). The closeness of the definition of causal nets to some physical ideas gives hope of finding a connection between the geometrical characteristics of these nets and the physical characteristics of computations, as, e.g., the size of the net and the *entropy increase* caused by the computation in question.

The last years witnessed a large number of *ad hoc* models for parallel computation addressing special problems like synchrony. Some of them, as also the classical Boolean circuits, are very different in nature from Turing-machine-style sequential models. For sequential machines, Kolmogorov and Uspenskii [1] made the first significant steps toward a model general enough so that most other models could be considered as its restricted forms. Their machine has a graph-like storage structure undergoing gradual local changes in time, by the work of a constant number of active units.

In the next sections, we introduce the concept of causal nets and compare it with a more traditional model of computations: Kolmogorov machines in parallel mode. We also consider the problem of computability when input nets with arbitrary symmetries are allowed. This problem seems to be new because it does not arise but for sufficiently general concepts of parallel computations like the ones presented here. We give a complete characterization of the functions computable in these models —in terms of the automorphism group of the input. The result can be considered as some "Church Thesis" for symmetry-preserving computations and is related to some combinatorial theorems of Babai and Lovász [5]. L. A. Levin originated the concept of causal nets, P.Gács proved the result on the symmetric inputs.

1 Basic definitions

Causal nets

A *net* X is a directed labeled graph, i.e., a matrix $\theta : |X|^2 \mapsto \Theta$ defining the label $\theta(x, y)$ of the edge (or the symbol ∞ of its absence) between any two nodes. The set of *ancestors* of any node x (i.e., of nodes connected to x by a directed path) is assumed to be finite. A *subnet* is the restriction of θ to a subset of the nodes. The *input* subnet is the union of all oriented cycles. The *cause* $\lfloor x \rfloor$ of a node x is the subnet of nodes y for which (y, x) is an edge. The *immediate consequence* A^+ of a subnet A is A extended by all nodes (with ingoing edges) the entire cause of which is contained in A.

A net represents the whole space-time history of a computation rather than its state at some time moment. A node of the net corresponds to an "elementary event" in the course of the computation, the edges to "causal relations" between them. We can (and will) use multiple edges—simulated by adjusting the set Θ, and states for the nodes—simulated by the states of the preceding edges.

Definition 1 *A net is called* local *if the cause of each node is* weakly connected *(i.e., connected as an undirected graph). A local net is called* causal *if any isomorphism between its two subnets A and B can be uniquely extended to an isomorphism between A^+ and B^+.*

The requirement of uniqueness is not essential and is imposed only for convenience. To check for causality and locality, only subnets isomorphic to causes of nodes should be considered. This is easy since all such subnets are small and connected.

The requirement of causality is the way we represent physical determinism: the past uniquely determines the future. Another important physical principle, that of the locality of interaction requires that the immediate cause of an elementary event should consist of events closely related to each other. The evidence for this close relation is usually present in a chain connecting these events and should be considered as part of the cause. Thus, nodes of the cause of a node have causal interconnection themselves and therefore correspond to close but different moments of time (in some analogy to the formalism in mechanics where the future positions of a system are determined by its present position and a position in the near past—giving a speed).

The noninput nodes and the strongly connected components (packets) of the input form an acyclic graph with a natural partial order $\leq$ on it. The *base subnet* consists of the input and all preceding nodes. The *output subnet* consists of the noninput nodes adjacent to edges labeled by a distinguished output subalphabet Θ_O. Any graph can be converted to an input net by adding a loop to each node. These are the usual bases for nets. Other types of base may be used to simulate fancy things, e.g., the use of "oracles" (input nodes whose cause contains noninput nodes).

The nodes of a net can be objects of any kind. But a noninput node x can be naturally identified with the function mapping $y \in \lfloor x \rfloor$ to $\theta(y, x)$. In the case of a single-label alphabet, x can be identified with $\lfloor x \rfloor$. Then the causality of a net X can be expressed as $|X| \supset x \underline{\simeq} y \in |X| \;\Rightarrow\; x \in |X|$.

Programming

A causal net can be usually described much more concisely than by listing the entire matrix θ. It is already uniquely determined by its base and the types of neighborhoods occurring in it (unlike the Boolean circuits). The *neighborhood* $V(x)$ of a node x (its *center*) is the subnet consisting of x and all nodes connected to x. The *causal neighborhood* $C(x)$ contains x and $\lfloor x \rfloor$. *The local [causal] structure (also called program)* of a net is the set of its [causal] neighborhoods or "commands" (up to isomorphism). A net X is said to be *consistent* with any local [causal] structure containing the one of X.

For any $\mathcal{B}$ and causal program $\mathcal{P}$, a unique (possibly infinite) causal net $\mathcal{P}(\mathcal{B})$ with base $\mathcal{B}$ exists whose causal structure is the maximal one consistent with $\mathcal{P}$. $\mathcal{P}$ is said to *generate* $\mathcal{P}(\mathcal{B})$ from $\mathcal{B}$. If the net is finite and the output exists in all connected components, we say that the *output* is *computed* from the base by the program. Thus to implement computations by these concepts, take a finite causal program $\mathcal{P}$ and input A, let the program start generating a causal net from it by subsequent extensions and take the output as the result.

The requirement of consistency with some fixed local structure is a useful way to impose various local restrictions on the net, e.g., boundedness of the degree of the nodes. The computation by a causal net is *monotone*: from a part of the input, always a part of the output will be computed. To eliminate this effect, one can always confine oneself to functions in whose domain no input net is a proper part of an other one. Such a domain is, for example, the set of all nets consistent with a *closed* local structure as defined below. Also, in a closed net, we can easily recognize the last moment when a node was used in generating other nodes.

Definition 2 *A* net *is* locally asymmetric [closed] *if none of its neighborhoods has a nontrivial isomorphism to itself [to a proper part of another one]. A closed locally asymmetric net with one distinguished central node in each (weakly) connected component is called* marked.

The nodes of a connected marked net can easily be numbered in a canonical way: we construct a spanning tree with the central node as the root, proceeding on the edges of X from the root, e.g., in a breadth-first manner. In the theory of information processing, we practically never encounter nonmarked nets, and the permission of symmetric nets gives rise to serious special problems (like the problem to find an algorithm deciding whether two given graphs are isomorphic).

Example: representation of a Turing machine

A Turing machine has a tape—a finite succession of cells numbered by subsequent integers, and a head observing the cell with number $c(t)$ at time t. A finite set of states is fixed and each cell k as well as the head is at each moment t in one of these states $p(t,k)$ and q_t. The terminal cells have the distinguished states R and L. The program of the machine is a finite function λ ordering certain actions to pairs of states. Thus, $\lambda(q_t, p(t, c(t)))$ determines q_{t+1}, $p(t+1, c(t))$, $c(t+1) - c(t) = \pm 1$ and $p(t+1,k) = p(t,k)$ for all $k \neq c(t)$. If the cell $c(t+1)$ does not exist yet, it will be created. If the head was at one of the ends it also determines whether the cell $c(t)$ has to be removed. The sequence $p(0,k)$ is the input and $c(0) = 0$. Thus always $c(t) \equiv t \bmod 2$, and since the state of a cell cannot change in steps of different parity, we can exclude these from consideration. Let us agree that at the end of the computation, the head assumes a special state τ, and going from one end of the tape to the other one, erases it. (This prevents the representing causal net from being infinite.)

To represent the computations of this machine by causal nets, let $s(t,k)$ denote $(p(t,k),x)$, where x is q_t if $c(t) = k$, special symbol otherwise. Let the set V of nodes of the causal net be the set of time-cell pairs (t,k) of equal parity where the cell k exists at time t. The edges run between nodes $(t,k \pm 1)$ and $(t+1,k)$. Their label reflects the states $s(t,k)$ of their adjacent nodes. Other edges, with some constant label, run between $(t-1,k)$ and $(t+1,k)$. If the cell k does not exist at moment $t-1$, this edge connects $(t+1,k)$ to the terminal cell or forms a loop when $t+1$ is 0 or 1. The output subalphabet contains the labels with states $s(t,k)$ having $x = \tau$.

It can be easily checked that the above defined net is causal and local.

2 Complexity of computations

Time and space

One of the differences between the more traditional models and the computations as modeled by the causal nets is that on the latter the elementary operations are *not* necessarily *synchronized*. Only the relative order of those events is determined which are in a causal relation to each other. What results is a certain vagueness in the definition of the storage requirement of a causal net.

Let us define the *height* $d(x)$ of a *node* x of a causal net as the maximum length of a decreasing sequence of nodes starting with x. The height of a whole connected net X is $D(X) = \max_{x \in V} d(x)$. The height can be considered as the time required for the computation. Let $\Phi(x)$ be a *monotone mapping* of $|X|$ to the axis of time. (An example is $d(X)$.)

Definition 3 *The* storage size $s_\Phi(t,X)$ *at moment t is the number of edges (x,y) with $\Phi(x) \leq t$ and $\Phi(y) \geq t$. Denote* $s_\Phi(X) = \max_t s_\Phi(t,X)$. *For an unconnected net, height and storage are defined componentwise, as a* family of numbers *indexed by the connected components of X.*

It seems to be unnatural to define the storage used at one moment in a way independent from the time function $\Phi(x)$; apparently by the same considerations that in the theory of relativity show that there is no invariant way to define the notion of two events occurring at the same time. (Note that any imaginable relativistic computer is representable by a causal net.)

Minimizing the storage size over all possible monotone mappings we obtain the value $s_0 = \min_\Phi s_\Phi(X)$ that is similar to the number of stones needed to "pebble" the net (see [6]). However, s_0 is not a realistic measure of storage requirement. It seems to be reasonable to require that a timing be realized by the height function of some net "implementing" X in some formal sense. And the minimizing timing may be hard to compute and not implementable.

Time-space trade-off

Machines that actually build up a causal net of size n from its program and input cannot require less storage than n. The situation changes if we are content with a machine that does not necessarily store a representation of the net, only gives $\theta(i,j)$ for any two nodes (their numbers) i,j on request. (The machine *weakly represents* the net.) This may require only storage $O(\log n)$ instead of n (that it never requires more is another formulation of the hypothesis of logarithmic time-space tradeoff). The next theorem was originally proved by N. V. Petri [2] in terms of some concrete types of machines, but causal nets are the most natural setting for formulating it. It says that the storage size for weak representation can be minimized (no speedups).

Theorem 1 *For any causal structure $\mathcal{P}$, there is a Turing machine T with the following property. For each input net X, using a weak representation of X (by an oracle), it weakly represents a causal net Y generated by $\mathcal{P}$ from X. Any other Turing machine M doing this (even only) for X will use storage no less than by a constant C_M times the storage used by T.*

Sketch of proof. The optimal Turing machine T works as follows. It sets forth a certain amount of storage s, then considers all possible other Turing machines M with a description shorter than s. Running over all k-tuples of numbers less than 2^s, where k is the maximum size of the elements of the causal structure $\mathcal{P}$, T is able to decide whether M while working within storage s weakly represents a net generated from X by $\mathcal{P}$. If it does not find any M doing that, it increases s. When it finally finds a fitting M, it uses M for answering the questions it was asked.

Example: Characterization of Pointer Machine Complexity

Various models of computation with only one finitary operation at each step can be considered as essentially a special case of Kolmogorov's graph machine [1]. This differs from the "storage modification machine" proposed later by Schönhage [3] and called "Pointer Machine" by Knuth only in that Schönhage works with directed, Kolmogorov and Uspenskii with undirected graphs (forcing thereby both bounded in- and outdegree). The storage structure, called *pointer graph* of the *Pointer Machine (PM)* is a directed labeled graph with constant outdegree.

The program prescribes how the central node transforms its 2-neighborhood step-by-step, modifying thereby gradually the whole graph. The initial graph is the input, the graph at halting is the output. They are labeled by the disjoint alphabets Θ_I, Θ_O.

Barzdin and Kalnins generalized the model of Kolmogorov and Schönhage by introducing parallelism. A program for the Parallel Pointer Machine (PPM) will be similar to the program of a PM but its meaning is different: the local transformations must be simultaneously carried out by all nodes. A node x changes only its outgoing edges, or disappears if they all loop. A common new node may be created by a maximal clique formed by edges with a distinguished label ϵ. In determining the next action, edges with output labels do not count. The computation is finished when all edges have output labels. A PPM is a *parallel Kolmogorov machine* (PKM) if its pointer graphs are *undirected* at each step (i.e., their matrix is symmetric) and each node has a loop with a special constant label. The set of nonempty undirected pointer graphs is denoted by $\mathbf{T}(\Theta)$.

The functions defined on undirected connected marked input graphs computable by the PM and PPM are exactly the recursive functions. With respect to computing time, the PPM is a powerful generalization of the PM and is able to solve, e.g., any NP problem in polynomial time (but possibly with exponential space). This model can claim to be able to efficiently simulate any other model of parallel computation.

A function f computable by a PPM—just as the complexities in Definition 3—is *componentwise*, i.e., it commutes with *disconnected union*: $f(X \cup Y) = f(X) \cup f(Y)$ if $|X| \cap |Y| = \emptyset$. We associate a pointer graph Z' with a (possibly acyclic) net Z by identifying all nodes connected by edges with a special label η.

Note. The above version of the PPM is more general than usual in order to extend Theorem 2 to symmetric inputs. For usual computations, the inputs should be assumed marked.

Theorem 2 *For componentwise functions f, u, v over $\mathbf{T}(\Theta_I)$ these properties are equivalent.*

(a) A PKM exists computing $f(X)$ for each X in time $O(u(X))$ and storage $O(v(X))$.

(b) For each X a closed causal net Y exists with bounded degrees of nodes, with input X, output Z with $Z' = f(X)$, $D(Y) = O(u(X))$, $s_d(Y) = O(v(X))$.

The proof will be given in the Appendix.

Open problem. Find out which traditional complexity corresponds to the size of causal nets. It is known that the size of the smallest causal net computing a function is between the time required on a PM and the time required on an "address-machine" (a PM with a tree-like storage structure). The second complexity may exceed the first one only by a logarithmic factor.

3 Symmetric inputs

In this section, we will characterize the functions computable by causal nets. Of course, every such function is partial recursive. But it turns out that partial recursive functions that are defined on certain very symmetric inputs are not computable in models preserving this symmetry.

Let us try, e.g., to compute $n \bmod 2$ from a "circle X of length n": some net with the automorphism group Z_n (the cyclic group of order n). We ask for a program generating a one-edge output z from X with state equal to $n \bmod 2$. Thinking in terms of parallel pointer machines, we can imagine the input as a circular array of identical automata—capable of unlimited local organization and creation—trying to merge into a single node. There is no leader among them to organize the process. Since all have similar initial neighborhood, the first merge can divide them only into small groups of identical size—which is impossible if their number is prime. Indeed, it turns out that the existence of such a program implies that n *cannot have any large prime divisors.* (Such numbers are sometimes called "smooth", in reference to smooth sand containing only fine grains.)

The functions computable on the Pointer Machine are exactly the partial recursive functions. However, the input to a PM must always be a *marked* pointer graph. Theorem 2 sets up a correspondence between functions computable by causal nets and those computable by the Parallel Pointer Machine. Hence for *marked* inputs, the functions computable by causal nets are just the partial recursive functions. On the other hand, functions that are not computable by causal nets will therefore be not computable by the Parallel Pointer Machine (a version of Theorem 2 holds also without the restriction that the PPM be a Kolmogorov machine). We now proceed to formulate the criterion for a recursive function without the markedness requirement to be computable by a causal net. We assume the nodes of nets to be constructive objects (say integers).

A partial componentwise function f from nets with a loop-edge at each node to output nets with uniformly bounded indegree will be called *standard*.

Let $\mathcal{P}$ be a causal structure generating causal nets X_0, X_1 with outputs B_0, B_1 from input nets A_0, A_1. Suppose further that there exists an embedding ι of A_0 into A_1. By causality, this embedding will generate an embedding of the whole causal net X_0 into X_1 and thereby an embedding $\iota^{\mathcal{P}}$ of B_0 into B_1. Notice that the image of B_0 will be an *ideal* C of B_1 ($y < x \in |C|$ implies $y \in |C|$). For different causal structures $\mathcal{P}$ computing f this correspondence of embeddings on the outputs to embeddings on the inputs can be different, but its existence is a serious restriction implying among others the *monotonicity* of f. Hence the first condition on the standard partial function f is the following. Let $\mathrm{id}_{A,B}$ be the identical embedding of $A \subset B$ into B.

(i) There exists a recursive correspondence F which orders an isomorphism ι^F of $f(A_0)$ onto an ideal of $f(A_1)$ to each embedding $\iota \ : \ A_0 \mapsto A_1$. F is a *functor*, i.e., $(\iota_0 \circ \iota_1)^F = \iota_0^F \circ \iota_1^F$. Let A_0, A_1 be subnets of net C, $A_2 = A_0 \cap A_1$, $B_j = \mathrm{id}^F_{A_j,C}(f(A_j))$. Then $B_2 = B_0 \cap B_1$.

This intersection property of the functor F reflects the fact that the net B_2 computed by a program $\mathcal{P}$ from the intersection A_2 of two nets A_0, A_1 is the intersection of the nets B_0, B_1 computed from A_0 and A_1, respectively. Indeed, $B_2 \subset B_0 \cap B_1$ is evident from monotonicity. But the ancestors in the input of each node of $B_0 \cap B_1$ are all both in A_0 and A_1, hence also in A_2. This proves $B_2 = B_0 \cap B_1$.

The above property implies that for a subnet B of an output net $f(A)$ we can find the smallest part of A still producing B. For any subnet A_0 of A define $f(A_0; A) = \mathrm{id}^F_{A_0,A}(f(A_0))$; this is the subnet of $f(A)$ computed from the subnet A_0 of A. The *set of ancestors* $f^{-1}(B; A)$ *of* B is the intersection of all subsets A_0 of A with $f(A_0; A) \supset B$. (Notice that this notion is defined only by the functor F, without causal nets.) It follows from (i) that $f(f^{-1}(B; A); A) \supset B$. In a causal net X, of course, a node a of the input A is the ancestor of a subset $C \subset |X|$ if $a \leq y$ for some $y \in C$. Notice that since the image of ι^F is always an ideal, $a < b$ implies $f^{-1}(\{a\}; A) \subset f^{-1}(\{b\}; A)$.

(ii) For each input A, the set of ancestors of each node of $f(A)$ is connected.

The most interesting property F must have is connected with possible symmetries of the inputs. The functor $\lambda \mapsto \lambda^F$ is a homomorphism from the group of automorphisms of A to that of $f(A)$. For any node x of $f(A)$, let us denote by $G(x, A, F)$ the factorgroup of the group of all automorphisms λ that leave x invariant (i.e., for which $\lambda^F(x) = x$) by the normal subgroup of the automorphisms that fix all elements of $f^{-1}(\{x\}; A)$. (This divisor is the unity if x depends on the whole input.)

For any finite group G, let $a(G)$ be the minimum of the indices of proper subgroups in G, $b(G)$ the maximum of $a(H)$ over all subgroups of G. $b(G)$ is sometimes called the *smoothness* of G in analogy to the above-mentioned notion of smoothness of natural numbers.

(iii) $b(G(x, A, F))$ is bounded for all inputs A.

Theorem 3 *For a standard partial function f, the following two conditions are equivalent.*

(a) For all nets A in the domain of f, there are finite causal nets with input A, output $f(A)$, and with bounded indegrees of noninput nodes.

(b) f satisfies (i-iii).

The proof will be given in the Appendix.

Remarks. (1) The recursiveness of the functor in (i) cannot be replaced by the weaker requirement of the recursiveness of the function f. In the Appendix, we give an example of a recursive function f with a nonrecursive functor satisfying the rest of (i-iii) which has no recursive functor (even without the rest of (i-iii)).

(2) Of most interest are functions in whose domain no net is a proper part of an other one, and which are invariant, i.e., their functor F maps any automorphism of the input into the identity on the output. In this case, (iii) requires the automorphism groups of inputs to be uniformly smooth.

(3) The smooth groups play an important role in the newly discovered isomorphism-testing algorithms of graphs of bounded valence [7]. We plan to follow up the consequences of [7] in a following work. Notice also that the automorphism group of a connected graph of bounded valence is smooth if one of its orbits is small.

4 Conclusion

Causal nets might become a simple and universal concept in the theory of computation: they provide an easy and natural way to describe the work of different real and imaginary computing devices since nothing occurs in their definition but the most general physical ideas concerning the processes going on in the machines. The causal net can be constructed already on the basis of the computation to be accomplished without specifying the type of machine used. Different characteristics of the computing resources correspond to simple geometric characteristics of the causal nets. Besides their universality, the causal nets have the advantage of a simple definition and give the possibility of considering each computation as a single finite object independently from the context of all possible computations of the same algorithm on different inputs. This makes common geometrical and algebraic methods available for the study of computations. At the same time, the theory of causal (in contrast to the Boolean) nets is equivalent to the theory of algorithms via the fact that a net is uniquely reconstructable from its input net if its local structure is known.

Proof of Theorem 2

The computation of the PPM is a series of pointer graphs $X = X_1, \dots, X_u = f(X)$. Let $p_t(x, y)$ be the label of the edge (x, y) in X_t (∞ if this edge does not exist). We construct a causal net A over the nodes (t, x) for all $x \in X_t$ having at time $t-1$ nonoutput outgoing edges. Put $\theta((t, x), (t', y)) = p_t(y, x)$ for all x, y in X_t where $t' = t+1$ or $t' = t = 0$. Connect also, by edges having some new constant label, all pairs $(s, y), (t, x)$ where y is at time $t-2$ in the 2-neighborhood of x—or of a node that created x if x is new and $s = t-1, t-2$. If x has an outgoing output edge, connect (x, t) and $(x, t+1)$ with an η-edge. A can be seen to be causal and closed. Its input graph is X. Its output graph Z contains a path of η-edges for each node of $f(X)$. After the contraction of these paths, we get $Z' = f(X)$.

It remains to prove $(b) \Rightarrow (a)$. For a closed local structure $\mathcal{Q}$, let us call its c-domain the set of all input graphs X from which $\mathcal{Q}$ computes a net Y with output Z satisfying $Z' = f(X)$, $D(Y) \leq cu(Y)$, $s_d(Y) \leq cv(Y)$. By the assumption of the theorem, any X is in the domain of some $\mathcal{Q}$ with maximal degrees of nodes bounded by some natural number k. Notice also that if $X_1 \cup X_2$ is in the domain of $\mathcal{Q}$ then so is X_1. It follows that a local structure $\mathcal{Q}$ exists whose domain is the set of all nets.

We can therefore suppose that $Y(X)$ is generated from X by $\mathcal{Q}$ applying subsequent extensions. We have to show that Y can be built up by a PPM (within the required time and storage bounds) level-by-level. In this construction, we will first use some temporary output labels $\overline{\alpha}$ when some edge of the net should occur with output label α. η-edges will be contracted as soon as possible.

Let A_t be the subnet of nodes of height $\leq t$ in Y. Let us omit from A_t all nonoutput nodes which are *closed*: whose neighborhood is isomorphic to an element of $\mathcal{Q}$. (These nodes cannot occur in the cause of any new node, so they are no longer needed.) The resulting subnet is B_t. The program for the PPM computes B_{t+1} from B_t in a constant number of steps in the following stages.

Stage 1. For all nodes x, the machine looks up all copies U of the cause of some command Z of $\mathcal{Q}$ containing x. This needs only constantly many steps since the the degree of the nodes of B_t is bounded by k. For each such U and Z, a new auxiliary node $v(x, Z, U)$ is added, with pointers having the same values as the pointers at x.

Stage 2. Each $v(x, Z, U)$ and $v(y, Z, U)$ is connected by a pointer with label ϵ in both directions forming thereby ϵ-cliques $M(Z, U)$ for the third stage.

Stage 3. The ϵ-cliques are replaced by single nodes with the corresponding pointers.

Stage 4. If a node x is closed do the following. If x is not an output node and has no adjacent η-edge then delete it. If x is connected to an other closed node by an η-edge then merge them. Thereafter, if x is an output node, convert all temporary output edges leaving x to the corresponding final ones (which are also output edges of the PKM we are just defining). ■

Do chips need wires?

A physical device (like a chip) realizing a parallel Kolmogorov pointer machine should have its active elements (nodes) attached to a 2-dimensional surface, for the purposes of energy exchange. Thus, we assume the device to be a plane square, and the nodes to be subsquares with integer corners. Each node x at each moment has k links (d_x is the maximum of their lengths) to other nodes (the *partners* of x). We do not care, for the moment, about the physical realization of the links, and do not assume that they occupy any separate place on the chip. We require, however, that a node occupies at least $k \log d_x$ in area (to store the relative addresses of the partners), and its elementary operation takes time proportional to d_x (the speed of communication is bounded). Moreover, this time is $d_x \log^c d_x$ for devices called $c - chips$, where $c > 1$ is a constant. A chip is called *primitive* if all numbers d_x are bounded by a constant. One can prove the following.

Note. Every c-chip can be simulated in the same time by a primitive chip of the same size.

This result is in contrast to current chips where wires occupy most of the area and to the theorems that for most graphs of bounded valence, in any realization, the average link length and the diameter of the chip is proportional to the amount of nodes.

Proof of Theorem 3

1. To prove that (a) implies (b) it is enough to show that (a) implies (iii): the rest has been shown already. Let Y be any command of a causal structure $\mathcal{P}$. It can have automorphisms of its own, which divide the cause of Y into orbits (transitivity classes). Denote by $k(\mathcal{P})$ the maximum size of orbits in all commands in the program $\mathcal{P}$.

Suppose that (a) holds, i.e., that some causal structure $\mathcal{P}$ generates from every input net A a causal net X with output $B = f(A)$. Let x be a node of B. Let F be the functor naturally provided by the net. We will show that $G(x, A, F)$ has a $k(\mathcal{P})$-bounded smoothness.

Let C be any subset of X. Let $G(C)$ be the group of automorphisms of A leaving each node of C fixed. Let C^0 be the set of ancestors of C in A. We will show that $b(G(C)/G(C^0)) \leq k$. We will use induction over the following partial ordering $\prec$ of sets of nodes of X. $C_1 \prec C_2$ if $C_1 \supset C_2$ and every element x of C_1 is majorized by an element $y \geq x$ of C_2. A set C is minimal in this order only if it contains C^0. In this case, $G(C)$ is the one-element group. Suppose that the assertion is true for all $C' \prec C$. If the cause of every node of C is in C then C contains C^0. Suppose that C contains a node x for which $\lfloor x \rfloor \not\subset C$. Let $y \in \lfloor x \rfloor - C$. Put $C' = C \cup \{y\}$. By our inductive assumption, $b(G(C')) \leq k$, since $C' \prec C$. We show that $|G(C) : G(C')| \leq k$. $G(C')$ consists of all elements of $G(C)$ that leave y fixed. To each coset of $G(C')$ in $G(C)$, a different node of $\lfloor x \rfloor$ will correspond which is, moreover, in the orbit of y. Therefore the number $|G(C) : G(C')|$ of cosets is bounded by the maximum of the sizes of orbits in $\lfloor x \rfloor$— which is bounded by $k(\mathcal{P})$. This completes the proof that (a) of the theorem implies (b).

2. To prove the positive part, we will describe the way a causal structure generates the causal net X from any input net A to get output net $B = f(A)$. This description will make it clear how to formally define the actual causal structure. Besides properties *(i)-(iii)*, the only property of our functor F we can use is that it is (partial) recursive. However, the way a recursive function is computed does not help us immediately to construct the causal net, because it also uses some knowledge about the individuality of the nodes of the

input (we may assume that each node is a natural number), i.e., some numbering of the nodes. The causal nets, on the other hand, work in an invariant way from the beginning, without knowing about anything but the structural properties of the input. Our way to solve this difficulty (certainly not the most effective way) is to generate *all possible* numberings of a certain sort for the input, use them to compute the function value and then get rid of them. We need property *(iii)* for the third step. Also, it will be seen that $k(\mathcal{P})$ for the causal structure can be made as small as the maximum of $b(G(x, A, F))$ over all inputs A and output nodes x.

Put $N_k = \{1, \dots, k\}$. Let A be a connected net with n elements. Any one-to-one function $u : N_n \mapsto A$ will be called a *numbering*. Let our label alphabet Θ be ordered in some fixed way: $\Theta = \{\theta_1, \dots, \theta_r, \infty\}$. We also fix some pairing function $\langle i, j\rangle$ with inverse $\langle k\rangle_1, \langle k\rangle_2$, with the property that for each k, $N_{k^2} = \{\langle i, j\rangle : 1 \le i, j \le k\}$. We order the matrices with elements from $\{\theta_1, \dots, \theta_r, \infty\}$ lexicographically: $X = (x_{ij}) < Y = (y_{ij})$ if the sequence $\{x_{\langle k\rangle_1\langle k\rangle_2} : 1 \le k \le n^2\}$ is lexicographically smaller than $\{y_{\langle k\rangle_1\langle k\rangle_2} : 1 \le k \le n^2\}$. Each numbering u of an n-node net A orders to A a matrix $\theta(u(i), u(j))$. This is the matrix of the net A_u over N_n in which the connection of i and j is the same as of $u(i)$ and $u(j)$ in A. The numberings for which the corresponding matrix is lexicographically smallest will be called *frames*. The net $A^* = A_u$ will be the same for each frame u. By restricting ourselves to frames we can reduce the set of numberings that we have to consider. Frames can be considered as coordinate systems: a transition to a different frame is always accompanied by an automorphism of A. Let, namely, σ be a permutation of N_n, u a frame. Then $u\sigma$ is a frame again if and only if the transformation $u(i) \mapsto u(\sigma i)$ of A is an automorphism. Therefore fixing any frame u will establish an isomorphism $\phi = u \circ \sigma \circ u^{-1}$ between the automorphism group G of A and the group G^* of permutations carrying frames into frames (the dual group). We can suppose w.l.o.g. that $A = A^*$ (remember that on one hand, our nodes are numbers, on the other hand, a causal structure does not use these numbers anyway). In this case, the identical mapping is a frame and $G = G^*$.

Let now A be some net with possibly more than n nodes. An n-frame of A is a frame for some connected subnet C of A with n nodes. Let u be an n-frame. For any $k < n$, we denote by $u|k$ the restriction of the function $u : N_n \mapsto A$ to N_k. It is easy to see that if u is an n-frame then $u|k$ is a k-frame for each $k < n$. This is due to our special lexicographical ordering of the matrices: if a matrix is minimal then so are all its upper left corner submatrices. We define now a sequence of nets

$$A = C_0 \subset C_1 \subset \cdots$$

representing the k-frames for each $k \le n$. Suppose that C_{n-1} is defined. To get C_n we add a new node $\overline{u}$ for each n-frame u, together with two new edges: an α-edge from $u(n)$ to $\overline{u}$ and a β-edge from $\overline{u|n-1}$ to $\overline{u}$ where α, β are labels not used for other purposes.

Remark 1. Suppose that some encoding $E(\sigma)$ of permutations σ of N_n by nets is given. An appropriate program will be able to do the following. Whenever a code $E(\sigma)$ of some permutation is brought into a certain connection with a node $\overline{u}$ of C_n representing an n-frame, a new node v will be generated and connected by some new edges (labeled by two new symbols used only for this purpose) to $\overline{u}$ and $\overline{u\sigma}$. In other words, it is possible to go from $\overline{u}$ to $\overline{u\sigma}$ effectively. Moreover, it is possible to do this simultaneously for all n-frames u.

The subnet of nodes of B with n or less ancestors will be called the *n-th floor* B_n of B. The construction of the output proceeds in many stages. In stage n, the n-th floor will be constructed.

Let X_n be the part of the causal net built up through the n-th stage. It will contain the following parts (besides, possibly, many auxiliary nodes, from which these are distinguishable):

The input A.

The first n floors of the output B.

The net C_n.

Our objective is to find a causal structure constructing X_n from X_{n-1}. The first step is to construct C_n from C_{n-1} which does not present any difficulties. $B_n - B_{n-1}$ consists of all nodes which have exactly n ancestors. For an n-element subset L of A, let $B(L)$ be the set of all nodes of B whose set of ancestors is L. Then $B_n - B_{n-1} = \cup_L B(L)$ and this union is disjoint. If $L_1 \ne L_2$ then no edge goes between $B(L_1)$ and $B(L_2)$, otherwise the upper node on the edge would have more than n ancestors. (Here we used the fact that ι^F is an embedding into ideals.) Therefore for a given n, all $B(L)$'s can be generated independently from each other. In this way we reduced the problem to the case where

$$\#A = n, \quad B = B_n, \quad B_n - B_{n-1} = B(A)$$

This case is considered further. We also have already the structure C_n of frames of A.

The group $G^F = \{\sigma^F \ : \ \sigma \in G\}$ is a group of automorphisms of B. Two nodes x and y of B are called equivalent if $\sigma x = y$ for some $\sigma \in G^F$. The equivalence classes are called *orbits* in B. Let U and V be two orbits. We write $U < V$ if $x < y$ for some $x \in U,\ y \in V$.

Lemma 1 *The relation $<$ is a strict partial ordering of the orbits and (hence) the orbits are independent sets of nodes.*

Proof. First we show that $U < V$ and $V < W$ implies $U < W$. Let x, y_0, y_1, z be elements of U, V, V, W, respectively, with $x < y_0$, $y_1 < z$ and $\sigma y_0 = y_1$. Then $\sigma x < \sigma y_0 = y_1 < z$ which proves $U < V$. Now we show that $U < U$ does not hold for any orbit U. Indeed: if $x < \sigma x$ held then for some i we would have in our acyclic graph a cycle $x < \sigma x < \sigma^2 x < \cdots < \sigma^i x = x$. ■

Remark 2. The set $B_n - B_{n-1}$ consists, of course, of whole orbits.

Now we introduce an invariant numbering for the orbits of B. Remember that we supposed that $A = A^*$. The nodes of B are natural numbers, therefore the orbits of B can be lexicographically ordered as sets of natural numbers. Let us use this order together with the partial order $U < V$ defined above to generate a complete order $B_{n1}, \ldots, B_{np}$ of orbits of $B(A)$ for which if $B_{ni} < B_{nj}$ then $i < j$. The orbits B_{nk} of $B(A)$ will be constructed one-by-one: we construct a sequence of nets

$$X_{n-1} = X_{n0} \subset \cdots \subset X_{np} = X_n$$

where $B_{nk} \subset X_{nk}$. Suppose that $X_{n\,k-1}$ has already been constructed. Our goal is to construct the nodes of $D = B_{nk}$ and to connect them to the previously constructed nodes of B as required in B_n.

Let b be the node of D that is the smallest as a number. Let H be the group of automorphisms h of A with $h^F b = b$. We call two frames u and v equivalent if $v = uh$ with some $h \in H$. The equivalence classes uH thus defined are our candidates for the elements of D: we will construct single nodes $\overline{uH}$ to represent them. The node $\overline{uH}$ will represent the node $u^F b$: for any previously constructed element y of B, the connection of y and $\overline{uH}$ will be the same as that of y and $u^F b$ in B.

Now we show how to connect all nodes uh to all previously constructed nodes y of B with an edge expected between y and $u^F b$. An agent sitting at node $\overline{u}$ has his own view of the net constructed until now. He represents $x \in A$ by the number $u^{-1}x$ and y by the number $(u^{-1})^F y$. He connects y therefore to a new version of $\overline{u}$ in the way $(u^{-1})^F y$ should be connected to b in B. This is the same connection as between y and $u^F b$. Thus we created a new net $Y_{n\,k-1}$ that essentially looks like $X_{n\,k-1}$ except that a new copy of every node $\overline{u}$ has been created (we denote the new copy by the same symbol) with the connections to the previous parts of the net that $\overline{uH}$ should have. It remains to "merge" the nodes in $\{\overline{uh} \ : \ h \in H\}$ into a single node $\overline{uH}$ for each class uH.

Now we must use condition *(ii)*. It says that for some number k that is constant for our function f that we want to compute, the group H has a k-bounded chain

$$e = G_1 \subset \cdots \subset G_r = H$$

where $|G_{i+1} : G_i| \leq k$. Let $G_1, \ldots, G_r$ be the first (in some lexicographical order) among those chains of H with the smallest possible bound. We will construct a sequence of nets

$$Y_{n\,k-1} = D_1 \subset \cdots \subset D_r = X_{nk}$$

D_i will contain, besides D_{i-1} and some auxiliary nodes, a node $\overline{uG_i}$ for each equivalence class uG_i which has the same connections to nodes in $B_{n\,k-1}$ as $\overline{u}$ in $Y_{n\,k-1}$. We also have an edge with some special label from each node $\overline{uG_{i-1}}$ to $\overline{uG_i}$. Suppose that D_{i-1} has already been constructed.

Two nodes $\overline{uG_{i-1}}$ and $\overline{vG_{i-1}}$ are considered equivalent if $v = uh$ for some $h \in G_i$. An equivalence class will be of the form $\{\overline{uhG_{i-1}} \ : \ h \in G_i\}$. Thus the elements in an equivalence class will correspond to the cosets of G_{i-1} in G_i. Let $e = h_1, \ldots, h_{k_i}$ be some canonical representatives of these cosets (e.g., let each be the least in its coset in a lexicographical order of the permutations).

To construct D_i, we build up a sequence

$$D_{i-1} = D_i(1) \subset \cdots \subset D_i(k_i)$$

of nets. $D_i(j)$ contains, in addition to D_{i-1}, for each permutation $h_p \quad (p \le j)$ and each class uG_{i-1} a new node z which is connected by edges to $\overline{uG_{i-1}}$ and $\overline{uh_pG_{i-1}}$. These edges are labeled by a symbol λ used only for this purpose. The node z together with the two λ-edges will be called a λ-connection. If we have $D_i(k_i)$ the construction of D_i takes only one step: each set of nodes $\{\overline{uhG_{i-1}} \ : \ h \in G_i\}$, together with their causes and the nodes added to get $D_i(k_i)$ will form the cause of one new node, $\overline{uG_i}$. In this same step, this new node can be made to have the same connections to $B_{n\,k-1}$ as $\overline{uG_{i-1}}$.

Our only remaining task is therefore to construct $D_i(j)$ from $D_i(j-1)$. This will happen through a sequence of nets

$$D_i(j-1) \subset E_1 \subset \cdots \subset E_{i-1} = D_i(j)$$

E_m will have, in addition to $D_i(j-1)$, a λ-connection between each node $\overline{uG_{i-1}}$ and $\overline{uh_jG_m}$. To construct E_1 from $D_i(j-1)$ we must first construct for each frame u a new node representing u (we denote it also by $\overline{u}$) together with a λ-connection from node $\overline{uG_{i-1}}$ to this new node. Then we make a λ- connection from the node $\overline{u}$ to $\overline{uh_j}$ using Remark 1. This connection can be used to generate a λ-connection between $\overline{uG_{i-1}}$ and $\overline{uh_j}$. It is easy to see that the construction of E_m from E_{m-1} will take only one step for each m. ■

A recursive function with no recursive functor

The domain of our function f will be the set of certain 0-1 sequences. There is an obvious encoding of these sequences into nets. We will suppose that, e.g., the sequences 01 and 10 are isomorphic (reversal is an isomorphism). f is defined over all $\{a_{nk} = 101^n0^k \ : \ n,k > 0\}$ and $\{b_{nk} = 1101^n0^k \ : \ n,k > 0\}$. We will use the μ-operator from recursion theory. If for some function g, $g(k) = 0$ for all $k < n$ then we put $\mu_{k<n}(g(k) \neq 0) = n$. Let $g(n,k)$ be a number-theoretical function for which the predicate $P(n) \Leftrightarrow g(n, \mu_k g(n,k) \neq 0) = 1$ is undecidable. Put $G(n,k) = g(n, \mu_{j<k} g(n,j) \neq 0)$. Put

$$f(a_{nk}) = \begin{cases} 001 & \text{if } G(n,k) = 0 \\ 0001 & \text{otherwise} \end{cases}$$

$$f(b_{nk}) = \begin{cases} c = 00101100 & \text{if } G(n,k) = 0 \\ 0c & \text{if } G(n,k) = 1 \\ c0 & \text{otherwise} \end{cases}$$

Obviously, the only embeddings in this domain are the unique embeddings of a_{nk} to a_{nk+1}, b_{nk} to b_{nk+1}, a_{nk} to b_{nk} and the combinations of these. The functor F must correspond an embedding from $f(a_{nk})$ to $f(b_{nk})$ to the last type mentioned. If $G(n,k) = 0$, the functor has two possible values: we can embed 001 to the front or the end of c. But if $G(n,k) \neq 0$, there is only one embedding: either to the front or to the back. Notice that a_{n1} can also be embedded to a_{nk}. Therefore, the functor property implies that the embedding of $f(a_{nk})$ to $f(b_{nk})$ must be a continuation of the embedding of $f(a_{n1})$ to $f(b_{n1})$, and $f(a_{n1})$ will be embedded to the front or back of $f(b_{n1})$ depending on $P(n)$, i.e., in a nonrecursive way.

References

[1] A. N. Kolmogorov, V. A. Uspenskii: On the Definition of an Algorithm, *Uspekhi Mat. Nauk* **13**(1958) 3–28; *AMS Transl.* 2nd ser. **29** (1963) 217–245.

[2] N. V. Petri: Personal communication (1972).

[3] A. Schönhage: Storage Modification Machines, *SIAM J. on Computing* **9**/3 (Aug. 1980) 490-508.

[4] Ja. M. Barzdin', Ja. Ja. Kalnin's: A Universal Automaton with Variable Structure, *Automatic Control and Computing Sciences* **8**(2) (1974) pp. 6-12.

[5] L. Babai, L. Lovász: Permutation Groups and Almost Regular Graphs, *Studia Sci. Math. Hung.* **8** (1973) 141-150.

[6] S. A. Cook: An Observation on Time-Storage Trade-Off, *Proc. Fifth Ann. ACM Symp. on the Theory of Computing,* (1973) 29-33.

[7] E. M. Luks: Isomorphism of Graphs of Bounded Valence Can Be Tested in Polynomial Time, *Proc. of the 21th Symp. on FOCS*, Syracuse 1980.