International Journal of Theoretical Physics, Vol. 21, No. 12, 1982

Causal Nets or What Is a Deterministic Computation?

Péter Gács[1]

Computer Science Department, University of Rochester, Rochester, New York 14627

and Leonid A. Levin[2]

Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139

Received May 8, 1981

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.

1. 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

[1]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.

[2]The research of this author was partially supported by NSF grants MCS 77-19754 and MCS-8104211.

0020-7748/82/1200-0961$03.00/0

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[3]) 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 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 (1958) made the first significant steps toward a model general

[3]N. V. Petri (1972) is different from the inventor of Petri nets—which have no essential relation to our causal nets.

enough so that most other models could be considered as its restricted forms. Their machine has a graphlike 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 (1973). L. A. Levin originated the concept of causal nets, P. Gács proved the result on the symmetric inputs.

2. BASIC DEFINITIONS

2.1. 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 present 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 $\leqslant$ 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 single-labeled alphabet, x can be identified with $\lfloor x \rfloor$. Then the causality of a net X can be written as $(|X| \supseteq x \cong y \in |X|) \Rightarrow x \in |X|$.

2.2. Programming. A causal net can in general be described much shorter than by listing the entire matrix θ. It is already uniquely determined by its base and the types of neighborhoods that can occur 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 $\mathfrak{B}$ and causal program $\mathcal{P}$, a unique (possibly infinite) causal net $\mathcal{P}(\mathfrak{B})$ with base $\mathfrak{B}$ exists whose causal structure is the maximal one consistent with $\mathcal{P}$. $\mathcal{P}$ is said to *generate* $\mathcal{P}(\mathfrak{B})$ from $\mathfrak{B}$. 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).

2.3. 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 $c(t) \equiv t \pmod 2$, 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.

3. COMPLEXITY OF COMPUTATIONS

3.1. 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 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 $\Phi(x) \leqslant t$ and $\Phi(y) \geqslant t$. 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 Cooke, 1973). 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.

3.2. 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 (1972) 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 $\mathfrak{P}$, 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 $\mathfrak{P}$ 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.

3.3. 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 (Kolmogorov and Uspenskii, 1958). This differs from the "storage modification machine" proposed later by Schönhage (1980) and called "pointer machine" (PM) 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 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 $T(\Theta)$.

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 $|X| \cap |Y| = \varnothing$. 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 $T(\Theta_I)$ 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 $Z' = f(X)$, $D(Y) = O(u(X))$, $s_d(Y) = O(v(X))$.

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 treelike storage structure). The second complexity may exceed the first one only by a logarithmic factor.

3.4. 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.

4. 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 (mod 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 (mod 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 if we restrict our input nets to be *marked*, 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 necessary and sufficient conditions 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 $A \subseteq B$ 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, $B_2 \subseteq B_0 \cap B_1$ 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 $f(A_0; A) \supseteq B$. (Notice that this notion is defined only by the functor F, without causal nets.) It follows from (i) that $f(f^{-1}(B; A); A) \supseteq B$. In a causal net X, of course, a node a of the input A is the ancestor of a subset $C \subseteq |X|$ if $a \leqslant y$ for some $y \in C$. Notice that since the image of ι^F is always an ideal, $a < b$ implies $f^{-1}(\{a\}; A) \subseteq f^{-1}(\{b\}; A)$.

(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 factor-group 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).

Remarks. (1) The recursiveness of the functor in (i) cannot be replaced by the weaker requirement of the recursiveness of the function f. There is an example of a 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 (Luks, 1980). Notice also that the automorphism group of a connected graph of bounded valence is smooth if one of its orbits is small.

REFERENCES

Babai, L., and Lovász, L. (1973). "Permutation Groups and Almost Regular Graphs," *Studia Scientarium Mathematica Hungarica*, **8**, 141–150.

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

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

Kolmogorov, A. N., and Uspenskii, V. A. (1958). "On the Definition of an Algorithm," *Upsekhi Mathematicheskikh Nauk*, **13**, 3–28 (1963). *AMS Transl.* 2nd ser., 217–245.

Luks, E. M. (1980). "Isomorphism of Graphs of Bounded Valence Can Be Tested in Polynomial Time," *Proc. of the 21st Symp. on FOCS*, Syracuse 1980.

Petri, N. V. (1972). Personal communication.

Schönhage, A. (1980). "Storage Modification Machines," *SIAM J. on Computing*, **9** (3), 490–508.