Milking the Aanderaa Argument*

RAMAMOHAN PATURI

Department of Computer Science and Engineering, University of California,
San Diego, La Jolla, California 92093

JOEL I. SEIFERAS

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

JANOS SIMON

Computer Science Department, University of Chicago,
Chicago, Illinois 60637

AND

RICHARD E. NEWMAN-WOLFE

Computer and Information Science Department, University of Florida,
Gainesville, Florida 32611

INTRODUCTION

In the early 1970's, by an intricate and involved counting argument, Stål Aanderaa managed to show for the first time that every additional tape adds to the power of a Turing machine that operates in real time (Aanderaa, 1974). This was a full decade after Rabin (1963) had shown that the second tape adds power. With time, especially following the introduction of the information-theoretic approach in (Paul *et al.*, 1981), Aanderaa's argument has become better understood and even extended to new results. In this paper, we present a clean version of the argument and some new extensions.

The following four theorems summarize all the extensions we know about. All of them are lower bounds on the time required for on-line

* Part of the work was done while the first author was at Harvard University in 1985–1986, supported by the National Science Foundation under Grant MCS-83-02385. The work of the second author was supported in part by the National Science Foundation under Grant MCS-81-10430. The work of the third author was supported by the National Science Foundation under Grants CCR-87-10078 and CCR-87-16518.

0890-5401/90 $3.00

simulation of certain abstract storage units by certain other abstract storage units (Paul *et al.*, 1981). Theorems 3 and 4 are the new results, but we will prove or outline the proofs of the entire sequence, taking advantage of their common heritage.

THEOREM 1 (Paul, 1981). *Simulation of h pushdown stores using only $h-1$ single-head tapes requires time $\Omega(n(\log n)^{1/h})$.*

THEOREM 2 (Ďuriš *et al.*, 1984). *Simulation of one single-head tape and $h-2$ pushdown stores using only $h-1$ pushdown stores requires time $\Omega(n(\log n)^{1/h})$.*

THEOREM 3. *Simulation of h pushdown stores using only an arbitrarily initialized $(h-1)$-head tape, even with head-to-head jumps, requires time $\Omega(n(\log n)^{1/h})$.*

THEOREM 4. *Each of the above lower bounds holds even for probabilistic simulation.*

Except in the case of the very smallest values of h, all of these simulations *can* be performed in time $\mathcal{O}(n \log n)$, using Hennie and Stearns' (1966) *two*-tape simulation of *any* fixed larger number of tapes. One reason for the gap between this upper bound and the lower bounds we have summarized might be the failure of the Hennie–Stearns simulation to take advantage of more than two tapes when they are available.

As discussed in (Paul *et al.*, 1981), the pushdown stores and multihead tapes mentioned above do qualify as abstract storage units. For a pushdown store, each input command is either "push 0," "push 1," "test for emptiness," or "pop". The corresponding output responses are 0, 1, whether the store is empty (0 for yes, 1 for no), and what symbol (0 or 1) gets popped (0 if there is nothing to pop), respectively. For an h-head tape unit, each input command is of the form, "Write the symbol 'a' beneath tape head number i, and then shift that head rightward j positions," where $1 \leqslant i \leqslant h$ and $-1 \leqslant j \leqslant 1$. If we admit head-to-head jumps, then there are also commands of the form, "Reset head number i to the position of head number j," where $1 \leqslant i \leqslant h$ and $1 \leqslant j \leqslant h$. In either case, the corresponding output response indicates what symbol tape head number i is left scanning after the command is executed. Unless otherwise indicated, we assume that initially such a tape unit has all heads coincident and the symbol 0 ("blank") written at every tape location and that there is at least one other symbol, 1, in the tape alphabet. A combination of k pushdown stores and/or multihead tapes is also an abstract storage unit in an obvious way, with each input command or output response consisting of an ordered k-tuple of simple commands or responses.

A deterministic automaton (finite-state machine with access to some storage unit of its own) *simulates* an abstract storage unit if its input–output behavior matches that of the simulated (or "virtual") storage unit. (Because each command's response must precede the next command, such simulations are said to be *on-line.* We consider *only* on-line simulations, so we choose to shorten "on-line simulation" to just "simulation.") If the simulating automaton requires $T(n)$ steps in the worst case to handle its first n commands, then it simulates the storage unit *in time* $T(n)$, and we say that the simulating automaton's storage unit *can simulate* the first one in time $T(n)$. (For *real-time* simulation, there must be some fixed constant bound on the number of steps to handle each individual command; this is sufficient, but not necessary, for simulation in *linear* time. Aanderaa's original result was merely that the simulation of Theorem 1 could not be performed in real time.)

A *probabilistic* automaton differs from a deterministic one in that each step can depend on the outcome of a fair coin toss. Following Pippenger (1982) and Paturi (1985), we say that such an automaton correctly simulates an abstract storage unit only if its input-output behavior matches that of the simulated storage unit in *every* line of its computation (i.e., for *every* coin-toss sequence), but that the *time* spent on each particular command sequence is just the probability-weighted *average* over all lines of computation on that sequence. Continuing as for deterministic simulators, then, we say now that a simulator simulates within time $T(n)$ if it requires no more than (probability-weighted) time $T(n)$ to handle any initial sequence of n commands.

Note that a probabilistic simulator can have only finitely many computations on a finite command sequence. For, if it had infinitely many, then it would have to have at least one particular line of computation *that does not end,* and hence with input-output behavior that does *not* match that of the simulated storage unit.

Overlap Lemmas

We start with some purely combinatorial lemmas that relate the length of the sequence of "storage locations" accessed in essentially *any* on-line computation (and hence the length of the computation itself) to something called "overlap." It will follow that the only possible short computations will be sufficiently well behaved that we can get a handle on them in a somewhat natural way in the sections that follow.

The overlap lemmas do not depend at all on the *nature* of the "storage locations" referred to above. Consequently, the use of computational terminology would only obscure the formulation and proof of these lemmas, and we avoid it.

An *overlap event* in a sequence $S = l_1, ..., l_T$ (of "storage locations," in our applications) is a pair (i, j) with $1 \leqslant i < j \leqslant T$, and $l_i = l_j \notin \{l_{i+1}, ..., l_{j-1}\}$ ("visit and soonest revisit"). If $\omega_t(S)$ is the number of such overlap events "straddling" t (i.e., with $i \leqslant t < j$), then the sequence's *internal overlap*, $\omega(S)$, is $\max\{\omega_t(S) \mid 1 \leqslant t < T\}$.

Suppose we are given a sequence S that is parsed (by the reading of n distinguished input symbols, in our applications) into n subsequences $S_1, ..., S_n$, each contiguous. A *parse-respecting* subsequence of S is a contiguous subsequence that does not start or end inside an S_i. Each such subsequence can be specified by an interval of subsequence indices: Let interval $I = [i, j] \subseteq [1, n]$ denote the parse-respecting subsequence comprising S_i through S_j, and let $\|I\|$ denote the *weight*, $j + 1 - i$, of I. We can break such an interval I into optimally weight-balanced parse-respecting halves, quarters, eighths, etc., to get a subsequence collection that we denote by $\mathfrak{I}(I)$; recursively, $\mathfrak{I}(I) = \{I\}$ if I has weight 0 or 1, and $\mathfrak{I}(I) = \{I\} \cup \mathfrak{I}(I_1) \cup \mathfrak{I}(I_2)$ otherwise, where $\|I_1\| = \lfloor \frac{1}{2} \|I\| \rfloor$, $\|I_2\| = \lceil \frac{1}{2} \|I\| \rceil$, and $I = I_1 I_2$.

In each lemma below, it will suffice to consider subsequences of weight down only to n^e, where e is any fixed constant exponent properly less than 1. A choice such as $e = \frac{2}{3}$, say, will suffice for all our subsequent applications.

Overlap Lemma 1 (Aanderaa, 1974). *Let S be a sequence of length T, parsed into n subsequences. If $\omega(I) > \varepsilon \|I\|$ for each subinterval $I \subseteq [1, n]$ with $\|I\| \geqslant n^e$, then $T = \Omega(\varepsilon n \log n)$.*

Remark. Following common usage, we use the symbol Ω for "at least some constant times." (Similarly, we use the symbol $\mathcal{O}$ for "at *most* some constant times"; and we use Θ for the conjunction of the two, possibly with different constants. We use *lower case o* for "less than any fraction of" (with finitely many exceptions, of course).) The implicit constant here and the one in Overlap Lemma 2 below do not depend on S, T, or n; but $\varepsilon > 0$ *can* be a function of n.

Proof of Overlap Lemma 1. We seek many intervals I with internal overlap accounted for by distinct overlap events, planning then to use the fact that each item in S can serve as the second component of at most one overlap event. For each interval I, we can distinguish a straddle point t for which $\omega(I) = \omega_t(I)$, and then we can distinguish the $\omega(I)$ overlap events in I that straddle position t. Then the distinguished overlap events in I can be shared by another interval only if that interval includes the corresponding distinguished straddle point t. Therefore, if we consider the members of

$\mathfrak{I}([1, n])$ in decreasing order of weight, down to n^e, we can get distinct distinguished overlap from one interval I with $\|I\| = n$, one $(= 2 - 1)$ more with $\|I\| \geqslant \lfloor n/2 \rfloor$, at least two $(= 4 - 1 - 1)$ more with $\|I\| \geqslant \lfloor n/4 \rfloor$, at least four $(= 8 - 2 - 1 - 1)$ more with $\|I\| \geqslant \lfloor n/8 \rfloor$, etc. If n' is the greatest power of two less than or equal to n, then the sum of the internal overlap in these intervals exceeds

$$\begin{aligned} &\varepsilon n + \varepsilon \lfloor n/2 \rfloor + 2\varepsilon \lfloor n/4 \rfloor + 4\varepsilon \lfloor n/8 \rfloor + \cdots \\ &\quad \geqslant \varepsilon n' + \varepsilon n'/2 + 2\varepsilon n'/4 + 4\varepsilon n'/8 + \cdots \\ &\quad = m\varepsilon n' > m\varepsilon n/2, \end{aligned}$$

where $m = 1 + \frac{1}{2} + \frac{2}{4} + \frac{4}{8} + \cdots \geqslant \frac{1}{2} \lfloor \log n^{1-e} \rfloor = \Omega(\log n)$. ∎

The next lemma, Paul's version of the overlap lemma, is stronger. For the same conclusion, it requires only that a *sufficiently weighty* collection of the subsequences each have large overlap. Although Paul's own main result, our Theorem 1, does not seem really to need the stronger version, Paturi and Simon's (our Theorem 4) does seem to need it.

Overlap Lemma 2 (Paul, 1982). *Let S be a sequence of length T, parsed into n subsequences, and let I_0 be the entire parse-respecting subsequence $[1, n] = S$. If*

$$\sum_{\omega(I) > \varepsilon \|I\|} \|I\| > \frac{3}{4} \sum_I \|I\|,$$

restricting sums to intervals I satisfying both $I \in \mathfrak{I}(I_0)$ and $\|I\| \geqslant n^e$, then $T = \Omega(\varepsilon n \log n)$.

Proof. As above, restrict attention to intervals I satisfying both $I \in \mathfrak{I}(I_0)$ and $\|I\| \geqslant n^e$. Let $\mathfrak{I}$ be the set of such intervals selected as in the preceding proof to have distinct internal overlap. As seen there,

$$\sum_{I \in \mathfrak{I}} \|I\| > \frac{1}{2} \sum_I \|I\|.$$

From this and the current hypothesis, we conclude that even the *intersection* of the two index sets must carry significant weight:

$$\sum_{\substack{I \in \mathfrak{I} \\ \omega(I) > \varepsilon \|I\|}} \|I\| > \frac{1}{4} \sum_I \|I\| \approx \frac{1}{4} n \log n^{1-e}.$$

Therefore,

$$T \geqslant \sum_{\substack{I \in \mathfrak{I} \\ \omega(I) > \varepsilon \|I\|}} \omega(I) > \sum_{\substack{I \in \mathfrak{I} \\ \omega(I) > \varepsilon \|I\|}} \varepsilon \|I\| = \Omega(\varepsilon n \log n). \quad ∎$$

COROLLARY (Paturi and Simon, 1983). *Let $\mathscr{S}$ be a set of sequences, each parsed into n subsequences, and let $p > 0$ be some fixed small positive fraction. If, for each subinterval $I \subseteq [1, n]$ with $\|I\| \geqslant n^{\varepsilon}$, $\omega(I) > \varepsilon\|I\|$ holds for at least the fraction $1 - p$ of the sequences in $\mathscr{S}$, then at least the fraction $1 - 4p$ of the sequences in $\mathscr{S}$ have length $\Omega(\varepsilon n \log n)$.*

Proof. Again restrict attention to intervals I satisfying both $I \in \mathfrak{I}(I_0)$ and $\|I\| \geqslant n^{\varepsilon}$. For each such I, by hypothesis,

$$\sum_{\substack{S \in \mathscr{S} \\ \omega(I) \leqslant \varepsilon\|I\| \text{ in } S}} 1 \leqslant p \cdot \sum_{S \in \mathscr{S}} 1.$$

Multiplying by $\|I\|$, summing over all such I, and changing the order of summation, we get

$$\sum_{S \in \mathscr{S}} \sum_{\omega(I) \leqslant \varepsilon\|I\| \text{ in } S} \|I\| \leqslant p \cdot \sum_{S \in \mathscr{S}} \sum_{I} \|I\|.$$

If the fraction of the sequences $S \in \mathscr{S}$ satisfying the negation

$$\sum_{\omega(I) \leqslant \varepsilon\|I\| \text{ in } S} \|I\| > \frac{1}{4} \sum_{I} \|I\|$$

of the hypothesis of Overlap Lemma 2 were to exceed $4p$, this would yield a contradiction. Therefore, the fraction satisfying the hypothesis and conclusion of that lemma must be at least $1 - 4p$. ∎

INFORMATION-THEORETIC TOOLS

To show that a simulator is not fast, it suffices to find a family of particular command sequences on which it runs for a long time. The information-theoretic approach is to do this by incorporating into the command sequences bit strings that have no short descriptions, and hence that are unlikely to be susceptible to expedited handling as "special cases." For further discussions of this general approach and its applications, see Paul *et al.* (1981) and Li and Vitányi (1988).

The key tool in the information-theoretic approach is Kolmogorov's robust notion of descriptional complexity. Any computable partial function $F\colon \{0, 1\}^* \to \{0, 1\}^*$ can be viewed as a description scheme, in terms of which we can define a descriptional complexity $K_F\colon \{0, 1\}^* \to \{0, 1, 2, ..., \infty\}$ by

$$K_F(x) = \min\{\, |d| \mid F(d) = x \,\}.$$

Because there is a "universal" computable partial function, there is some F_0 for which

$$\forall F\, \exists c_F\, \forall x\; K_{F_0}(x) \leqslant K_F(x) + c_F.$$

Except for an additive constant, therefore, F_0 is as succinct a description scheme as any; so we define *the* descriptional complexity $K(x)$ of x to be $K_{F_0}(x)$. A word x is *(algorithmically) incompressible* if $K(x) \geqslant |x|$. Since there are 2^n binary words of length n, but only $2^n - 1$ possible shorter descriptions d, there is sure to be at least one incompressible word of each length. In fact, if we relax our standards of incompressibility by even one $(K(x) \geqslant |x| - 1)$, which is still quite sufficient for our purposes, then *most* words of each length must qualify as incompressible.

Our arguments usually proceed by (a tree of) contradictions: If we assume that the simulator is fast on the command sequence we concoct, then every case leads to a too-short description (partly based on the *operation* of the allegedly efficient simulator) of the incompressible string on which the command sequence is based. Therefore, the art of succinct description is an important part of our approach.

The natural technique of modular design leads to descriptions comprised of several separate components. To combine the components into one unambiguous composite description, however, we cannot simply concatenate them; and our fixed binary description alphabet has no reserved delimiter symbol. A solution is to replace each component by a "self-delimiting" variant. A usually adequate self-delimiting variant $\bar{x}$ of a bit string x can be obtained by inserting one additional bit after each of the original bits, to indicate whether that bit is an end of the string, at the modest cost of doubling the string's length $(|\bar{x}| = 2|x|)$. Sometimes there is one particular component whose length we cannot afford to increase; we can accomodate *one such exception*, provided the number of components is clear from the rest of the description: The trick is to place that one component, unchanged, at the end of the concatenated list of self-delimiting variants of all the other components.

We can similarly define *relative* descriptional complexities, based on description schemes F that are functions of *two* arguments:

$$K_F(x \mid y) = \min\{|d| \mid F(d, y) = x\}.$$

Informally, the second argument y is granted as "free help." Again this leads to a robust notion, which we denote $K(x \mid y)$. Relative to *any* fixed y, incompressible strings exist by exactly the same simple counting argument as before.

In terms of this notion of relative complexity, we can formulate a remarkable proposition, usually referred to as "the symmetry of information" (Zvonkin and Levin, 1970; Gács, 1974): For any pair of strings x and y, $K(x)-K(x\mid y)$ is approximately equal to $K(y)-K(y\mid x)$. Informally, this says that y is always approximately as helpful in describing x as vice versa! The approximations here are to within an additive term proportional to $\log K(x, y)$, where "x, y" can be taken to be the binary string $\bar{x}y$. (To avoid any distraction when logarithms might be 0 or undefined, assume our fixed description scheme admits no description shorter than 2, the base for all our logarithms.)

Since symmetry is the only nontrivial property of descriptional complexity that we need, and since the proof is short, we include a proof of a convenient reformulation, for the sake of completeness. This proof is based on the one in (Zvonkin and Levin, 1970; Gács, 1974), but Paturi (1985) includes an independently devised proof of an equivalent result.

THEOREM. *To within an additive term proportional to* $\log K(x, y)$,

$$K(x, y)=K(x)+K(y\mid x).$$

Proof. ($\leqslant$) We get a description for x and y if we concatenate a description for x, a description for y given x, and prefatory indication of where to separate the two, using the self-delimiting version of binary notation. Therefore,

$$\begin{aligned}K(x, y) &\leqslant K(x)+K(y\mid x)+\mathcal{O}(\log K(x))\\ &\leqslant K(x)+K(y\mid x)+\mathcal{O}(\log K(x, y)).\end{aligned}$$

($\geqslant$) We show

$$K(y\mid x)\leqslant K(x, y)-K(x)+\mathcal{O}(\log K(x, y)).$$

To do this, we can describe y in terms of its serial number in the recursively enumerable set $S=\{y'\mid K(x, y')\leqslant K(x, y)\}$. Given x, the size of such a description can be held to $\log(\# S)+\mathcal{O}(\log K(x, y))$, where X denotes cardinality.

It remains only to show that $\log(\# S)\leqslant K(x, y)-K(x)+\mathcal{O}(\log K(x, y))$. For the sake of argument, suppose this is *not* true. For every constant c, then, there are x and y for which $\# S>2^{d}$ for $d=K(x, y)-K(x)+c\log K(x, y)$. Because each distinct pair (x', y') with $K(x', y')\leqslant K(x, y)$ requires a distinct description of length at most $K(x, y)$, of which there are no more than $2^{K(x,y)+1}$ in all, this leads to a short description of x, as a member of a recursively enumerable set that is small:

$$\begin{aligned}\#\{x'\mid \#\{y'\mid K(x', y')\leqslant K(x, y)\}>2^{d}\} &< 2^{K(x,y)+1}/2^{d}\\ &= 2^{K(x)+1-c\log K(x,y)}.\end{aligned}$$

We can describe x using $K(x, y)$ ($\mathcal{O}(\log K(x, y))$ bits), d (also $\mathcal{O}(\log K(x, y))$ bits), and the serial number of x (at most $K(x)+1-c\log K(x, y)$ bits). For c large, this adds up to *less than* $K(x)$ bits, a contradiction. ∎

Proof of Theorem 1

First, we give an overview of the proof. Consider simulation of h pushdown stores using only $h-1$ single-head one-dimensional tapes. We must find command sequences that are time-consuming for the simulator. As in Aanderaa's original proof, the idea for such a sequence is to push incompressible data at h very different virtual rates, so that the heads of the simulator will always be neglecting (in some sense) at least one virtual rate. At times when we can get a handle on this neglect, we seize the opportunity and confront the simulator with a virtually transparent sequence of pop and push commands to the neglected virtual store that the simulator cannot possibly respond to quickly. (Implicit in any fast response would be a too-short description of the incompressible virtual data being stored.) After interjection of enough such interrogation sequences into the command sequence, the sequence will have become sufficiently time-consuming to demonstrate the promised lower bound.

Like Aanderaa's argument, this one requires the choice of several parameters subject to, but not completely determined by, constraints that arise only later in the proof. (The fraction $\frac{3}{4}$ in Overlap Lemma 2 is clearly an example of this; any fraction greater than $\frac{1}{2}$ could have served just as well. The fraction $\frac{1}{10}$ used below is another such example that does not tend to be too distracting.) To give a better idea of just what the essential constraints are and to avoid making too many decisions before they are motivated, we will just watch for the constraints and try to make it clear that they are consistent.

Now we elaborate. Let the "overlap fraction" ε and the "relative-push-density factor" d be appropriate functions of n, which will be the "computation weight." Suppose M performs the simulation on-line with fewer than h tapes; and assume, without loss of generality, that each tape's alphabet contains just two letters. For each sufficiently large n, we will find an input sequence of length $\mathcal{O}(n)$ requiring time $\Omega(\varepsilon n \log n)$. Assuming a choice $\varepsilon = \Theta((\log n)^{1/h}/\log n)$ works out, we will have the promised result.

We start with an input string that "deals" the bits of an incompressible string $x \in \{0, 1\}^n$ onto the h virtual stores at the respective relative rates $d, d^2, ..., d^h$. In the computation by M on this input string, we will focus on the sequence of accessed storage locations, parsed by the n operations that read the commands to push the successive bits of x onto the various virtual pushdown stores. As we interject additional input commands (the "interrogation sequences" referred to above), and as this changes the resulting computation, we will continue to parse only at these same n "read" operations; so the weight of the sequence corresponding to the evolving computation will remain n.

If $c > 0$ is the multiplicative constant implicit in the conclusion of the first overlap lemma (when $e = \frac{2}{3}$, say), then either the length of the computation is already at least $c\varepsilon n \log n$ (our goal), or there is some computation subinterval I with $\|I\| \geqslant n^{2/3}$ and $\omega(I) \leqslant \varepsilon \|I\|$. Focus on some such *$I$ with the earliest completion.*

Lemma. *Right after I, we can interject a (virtually) "transparent" interrogation sequence of length $m = \Theta(\|I\|/d^i)$ for some i $(0 \leqslant i \leqslant h-1)$ that requires $\Omega(dm)$ steps by M.*

Moreover, if the time is *still* less than $c\varepsilon n \log n$, we will be able to apply the lemma again to interject *another* such interrogation sequence *later* in the evolving input string, without changing the computation up to or disturbing the effect of any earlier one. If we continue this until either the time reaches $c\varepsilon n \log n$ or the input length reaches $2n$ (after at most $\mathcal{O}(n^{1/3} d^{h-1})$ interrogation sequences, since each one has length $\Omega(n^{2/3}/d^{h-1})$), then M will require time that is either $\Omega(\varepsilon n \log n)$ or $\Omega(dn)$. If we impose the constraint

$$d = \Omega(\varepsilon \log n), \tag{1}$$

then M will require time $\Omega(\varepsilon n \log n)$ in *either* case.

Proof of Lemma. Intuitively, the proof is quite simple. Because of the low internal overlap in I, the path of each simulator head in that subcomputation must be mostly monotonic on its one-dimensional tape, and the major variable must be its speed. And, there being too few simulator heads for the h very different rates of storage onto the h virtual stores, the simulator must "neglect" at least one of the rates. We should seize the opportunity to interrogate the corresponding virtual store.

To get a rigorous handle on the notion of the speed of a simulator head in I, let $I = I_1 \ldots I_h$, where $\|I_i\| = (1/h)\, \|I\|$. For each i, let n_i be the number of the original ("parsing") push commands in I_i addressed to the ith virtual store; i.e., $n_i = (d^i / \sum_{j=1}^{h} d^j)\, \|I_i\|$. Also for each i, let $I_{>i} = I_{i+1} \ldots I_h$.

Say that a simulator head *services* the ith virtual store in I if, in I_i, it ranges farther than, say, $\frac{1}{10} n_i/(h-1)$ ($\frac{1}{10}$ of its "fair share" of the bits to be stored on the ith virtual store), and if, in $I_{>i}$, it ranges *no* farther than $d \cdot \frac{1}{10} n_i/(h-1) = \frac{1}{10} n_{i+1}/(h-1)$ (assuming $i+1 \leqslant h$). In other words, the servicing head ranges far enough to take significant notes on the ith store's bits in I_i, and it subsequently stays relatively close to its notes from that interval.

In a rigorous sense now, at least one virtual store is *neglected*—i.e., it is serviced by no simulator head. For, otherwise, some one simulator head would have to service two distinct virtual stores, say i and i', where $i<i'$. In this case, that head would have to range both less than and more than $\frac{1}{10}n_{i+1}/(h-1)$ during $I_{i'}$.

Say it is the ith virtual store that is neglected in I. For the desired interrogation sequence, then, use a sequence of hn_i pops addressed to that store, followed by the appropriate sequence of hn_i pushes to undo the virtual changes, for the sake of transparency. To see that M requires time $\Omega(dn_i)$ to handle this interrogation sequence, observe that, otherwise, there can be no reference to any significant notes on the n_i bits of the incompressible string x that were pushed onto the ith virtual store in I_i. The intuitively implied small descriptional complexity of these "special" bits does lead, in turn, to an unreasonably short description of the entire incompressible string x.

To elaborate, note the situation at the end of I on any "fair share" tape: The range during I_i was at least $\frac{1}{10}n_i/(h-1)$, the subsequent range was at least d times that far, and the internal overlap was at most $\varepsilon\|I\|$. Since the tape is linear, the *final* head position must be at least

$$d\cdot\tfrac{1}{10}n_i/(h-1)-2\varepsilon\|I\|=\Omega(dn_i)$$

tape squares away from *any* storage location scanned during I_i, assuming, say,

$$2\varepsilon\|I\|<\tfrac{1}{2}\cdot\tfrac{1}{10}n_1/(h-1). \tag{2}$$

To return to a storage location scanned during I_i on a "fair share" tape would certainly require $\Omega(dn_i)$ steps by M, so suppose M can handle the interrogation sequence *without* any such return. Then the special bits can be described by information sufficient to determine those parts of M's instantaneous description at the end of I *not involving* the storage locations scanned during I_i on "fair share" tapes. Therefore, the *entire* string x can be described by the following information:

1. All bits except the special ones, last and literally ($n-n_i$ bits).
2. Enough information to determine where the omissions are:
   (a) a description of this whole scheme ($\mathcal{O}(1)$ bits),
   (b) the relevant lengths and locations—n, d, where I starts, $\|I\|$, and i ($\mathcal{O}(\log n)$ bits).

3. Enough additional information to run the simulator M up to the beginning of I_i:

(a) a description of M ($\mathcal{O}(1)$ bits),

(b) the location, length, and virtual store number for each interrogation sequence already inserted up to the beginning of I_i, ($\mathcal{O}(n/(\text{minimum interrogation length}))\cdot\mathcal{O}(\log n)$ $=\mathcal{O}(n/(n^{2/3}/d^{h-1}))\cdot\mathcal{O}(\log n)=\mathcal{O}(n^{1/3}d^{h-1}\log n)$ bits).

4. Enough *patches* to M's instantaneous description at the *beginning* of I_i to satisfy our needs *following* that subinterval:

(a) the new control state and head positions ($\mathcal{O}(\log n)$ bits),

(b) *full* patches after I_i to each "unfair share" tape (at most $2\cdot\frac{1}{10}n_i/(h-1)$ bits for each, using the simple double-length code),

(c) *enough* patches after I_i to each "fair share" tape to correctly determine the *further* changes in $I_{>i}$ (at most $2\cdot\varepsilon\|I\|$ bits for each, using the simple double-length code).

For all this to add up to less than n bits, the desired contradiction, the following would suffice:

$$n^{1/3}d^{h-1}\log n=o(n_i), \tag{3}$$

$$(h-1)\cdot 2\cdot(\tfrac{1}{10}n_i/(h-1)+\varepsilon\|I\|)<\tfrac{1}{2}n_i. \quad \blacksquare \tag{4}$$

It remains only choose ε and d to satisfy the accumulated constaints, (1)–(4). For (4), it is enough to have

$$2(h-1)\varepsilon\|I\|<\frac{3}{10}n_i=\frac{3}{10}\left(d^i\Big/\sum_j d^j\right)\frac{1}{h}\|I\|,$$

which holds if ε is smaller than some small fraction (depending on h) of $1/d^{h-1}$. For the strongest result, we want ε large and hence d small. Taking $\varepsilon=\Theta(1/d^{h-1})$ (the limit for our derivative of constraint (4)) and $d=\Theta(\varepsilon\log n)$ (the limit for constaint (1)) yields $d=\Theta((\log n)^{1/h})$ and $\varepsilon=\Theta((\log n)^{1/h}/\log n)$, which satisfy all four of constraints (1)–(4). This makes $\varepsilon n\log n=\Omega(n(\log n)^{1/h})$, as desired.

Proof of Theorem 2

We slightly modify the preceding proof, to deal with simulation of one single-head one-dimensional tape and $h-2$ pushdown stores by only $h-1$ pushdown stores. The hard input string of length $\mathcal{O}(n)$ will again evolve from one based on an incompressible string $x\in\{0,1\}^n$. This time, however, only the last $\lceil n/3\rceil$ bits will be dealt as above onto the virtual tapes, of which there are now only $h-1$. (For this purpose, the one virtual tape that is more powerful will be treated as a pushdown store that grows to the right.) *Before* dealing out these bits, the initial input string will write the *first* $\lfloor 2n/3 \rfloor$ bits from *right to left* on the more powerful virtual tape, to provide one additional potential challenge to the simulator M as these bits are later reached during left-to-right storage on that same virtual tape. In the evolving computation by the simulator, there will be parse "seams" only for the steps that read the commands that deal out the last $\lceil n/3 \rceil$ bits of x.

As before, we will insert transparent interrogation sequences addressed to neglected virtual stores for as long as necessary. The pigeon-hole argument that there is always neglect still goes through because there are now *two* ways to neglect the one virtual tape that is more powerful: The usual way, and also by abandoning all significant notes on the first $\lfloor 2n/3 \rfloor$ bits of x.

More precisely, we redefine "neglect" as follows: For $1 \leqslant i \leqslant h-1$, the ith virtual store is neglected in I if each simulator pushdown store that grows by more than $\frac{1}{10}n_i/(h-1)$ in I_i goes on to vary in size by more than $d\cdot\frac{1}{10}n_i/(h-1)=\frac{1}{10}n_{i+1}/(h-1)$ in $I_{>i}$. (By the low-internal-overlap assumption, the variation has to be mostly to larger sizes.) In addition, the "0th" virtual store is neglected if every simulator pushdown store varies in size by more than $\frac{1}{10}n_1/(h-1)$ in I.

Unless the neglected virtual store is the 0th, the next interrogation sequence is obtained exactly as before. If *only* the 0th virtual store is neglected, then each simulator pushdown store must be servicing one of the other virtual stores. Since this entails growth by at least $\frac{1}{10}n_1/(h-1)$ on every simulation pushdown store, all notes from before I must still be at least $\frac{1}{10}n_1/(h-1)-2\varepsilon\|I\|=\Omega(n_1)$ deep at the end of I, because of the low internal overlap in I. Therefore, we can interrogate with a sequence of $\lfloor n_1/d \rfloor$ *right* shifts on the one, more powerful virtual tape, followed by the same number of left shifts for the sake of transparency.

Proof of Theorem 3

The arbitrary initialization here is handled easily: Just measure all descriptional complexities relative to (a big enough piece of) the initial (possibly infinite) instantaneous description of the simulator. The remaining, more difficult new problem is that the old notion of "neglect" no longer seems to guarantee vulnerability. Even after an interval of low overlap, each simulator head's abandoned notes might be readily available to *other* simulator heads.

Before, we broke the low-overlap interval I into just h subintervals.

During the ith subinterval, we focused only on storage to the ith virtual store. The solution now will be to break I into a larger, though still constant (i.e., not dependent on n), number, $\Theta(h^4)$, of subintervals, enough for $\Theta(h^3)$ such "rounds" of focus. Although the simulator might successfully cope with the neglect in particular, individual rounds, there will be only so much potential for it to do so. Eventually, the simulator will be genuinely vulnerable to time-consuming interrogation of some virtual store.

During I, because of the low-overlap constraint, we can view the simulator as storing its notes on a set of $2(h-1)$ initially empty pushdown stores, in the worst case—one growing in each direction from each head's position at the beginning of I. With head-to-head jumps, the simulator actually can store significant notes on as many as $2(h-1)-1$ of these stores: After a first head stores significantly in one direction from its initial position, it is prevented by low overlap from backing up to store in the other direction; but it *can* jump to any other head's still-initial position, from which the *two* heads can store significantly in *both* directions.

As before, our notion of "storing significant notes" should depend on the current focus. If n_i is the number of bits to be stored on the ith virtual store during each of the subintervals of focus on storage to the ith virtual store, then we might use "ranging farther than $\frac{1}{10}n_i/(2(h-1))$" as our notion of "storing significant notes" during such a subinterval. Even the smallest of these ranges will be large compared to the allowable overlap.

Although the simulator might store significant notes on as many as $2(h-1)-1$ of the $2(h-1)$ conceptual pushdown stores introduced above, the shortage of heads does rule out arbitrary *alternation* among the stores. In particular, no head can ever again write on any one of the stores that is not still essentially empty and that does not already have a head currently "assigned" to it. For this purpose, each head is initially *un*assigned, and it becomes *assigned* to a store whenever it starts writing significantly (on an essentially empty store) or jumps to a store that already has a head currently assigned to it. (This "assignment" notion is similar to the "block membership" notion devised by Cook and Rackoff (1980) to keep track of the effect of head-to-head jumps in maze-recognizing automata.)

In the ith subinterval of each round, we can model each head taking significant notes on the commands to the ith virtual store as pushing an atomic "metasymbol" a_i corresponding to that virtual store onto each of the stores where it takes significant notes. By incompressibility, there must be at least one such metasymbol pushed during each round's ith subinterval, and the notes associated with each metasymbol a_{i+1} must be $\Omega(d)$ times as long as the virtual information associated with a_i. The simulator will be vulnerable to a sufficiently time-consuming interrogation of the ith virtual store if every metasymbol a_i pushed during some one such subinterval is sufficiently "hidden" from the nearest possible head in each direction. In each direction, any metasymbol a_j with $j > i$ is a sufficient obstacle to guarantee that, from that direction, just reaching the notes associated with the hidden metasymbol a_i would require $\Omega(d)$ times as long as would the virtual interrogation of the associated stored bits. Therefore, for example, we can be sure that a metasymbol a_i is sufficiently hidden if it lies in a "valley"—a metasymbol string of the form $a_j a_i^l a_k$ with $j > i$, $k > i$, and $l > 0$—that does not contain a simulator head.

To see that such vulnerability is inevitable, suppose it is not. Of the metasymbols pushed in each subinterval of each round, then, we can select one that subsequently remains unhidden from the $h-1$ simulator heads. Of course these selected metasymbols remain unhidden even if we *ignore* the *un*selected ones. But this contradicts the following simple combinatorial lemma.

LEMMA. *After $\mathcal{O}(h^3)$ rounds, or $\mathcal{O}(h^4)$ pushes, some one store has more than $h-1$ valleys.*

Proof. A store with only $h-1$ valleys can have at most $\mathcal{O}(h^2)$ adjacent pairs $a_i a_j$ with $i \neq j$. But each round that does not start storing onto a previously empty store, of which there are only $2(h-1)$ initially, pushes such a pair onto at least one store. ▮

PROOF OF THEOREM 4

Again we modify the proof of Theorem 1; it is a routine subsequent exercise to modify the proofs of Theorems 2 and 3 analogously. The novelty this time is to simultaneously consider *all* the computations by M on the evolving input string. Actually, we include enough *copies* of more probable computations so that we can equate probability with frequency. Since the probability of each computation is a power of $\frac{1}{2}$, and since there are only finitely many computations on any particular finite command sequence, this is always possible. In terms of the corresponding coin-toss sequences, we can simply pad out the short ones in all possible ways, so that the results all have the same length.

To deal with multiple computations in our modified proof, we replace the use of Overlap Lemma 1 with use of the corollary to Overlap Lemma 2. So let $c > 0$ now be the multiplicative constant implicit in the conclusion of the version of that corollary with $p = \frac{1}{5}$, say. Until either the length of the computation on the current input reaches $c\varepsilon n \log n$ with probability at least $1 - 4p = \frac{1}{5}$ or the length of that input reaches $2n$, we hope to insert, later and later in the input string, transparent interrogation sequences, of lengths

m, that require $\Omega(dm)$ steps by M with probability $\Omega(p/h)$, and hence on the average.

At each continuation stage, by the corollary, there must be some subinterval I with $\|I\| \geqslant n^{2/3}$ and with $\omega(I) \leqslant \varepsilon\|I\|$ with probability at least p. In each one of the computations with $\omega(I) \leqslant \varepsilon\|I\|$, some simulated virtual store must be neglected, exactly as in the proof of Theorem 1. Therefore, some one virtual store i must be neglected in I with probability at least p/h. We interrogate with hn_i pops to store i, followed, as usual, by the appropriate sequence of pushes to undo the virtual changes.

It remains only to show that the proposed interrogation sequence of length $m = 2hn_i$ does require $\Omega(dm)$ steps by M with probability $\Omega(p/h)$. For this, it will suffice to show now that the interrogation sequence requires at least that many steps by M in (every resulting adaptation of) at least some fixed fraction of the computations in which virtual store i is neglected in I.

To get this required fixed fraction, first discard the half of the designated computations that are *longest*. Since the average of *all* the computations' lengths is less than $\frac{1}{5}c\varepsilon n \log n$, these cannot all be longer than

$$T = \frac{(1/5)\, c\varepsilon n \log n}{(1/2)\, p/h} \leqslant n^{\mathcal{O}(1)}.$$

Therefore, *none* of the *remaining* designated computations can be longer than T, and we may as well truncate their coin-toss sequences to that length and discard duplicates that share the same truncated coin-toss sequence.

For each remaining designated computation that is an *exception* (i.e., that does *not* require $\Omega(dm)$ steps by M to handle the proposed interrogation sequence), the construction in Theorem 1 now gives a short description ($n - n^{\Omega(1)}$ bits) of the incompressible string x *in terms of the length-T coin-toss sequence s that determines the computation.* By the symmetry of information, this gives a short description, of length

$$|s| - n^{\Omega(1)} + \mathcal{O}(\log K(x, s)) \leqslant T - n^{\Omega(1)} + \mathcal{O}(\log n) \leqslant T - n^{\Omega(1)},$$

of s in terms of x. But the number of descriptions-in-terms-of-x of length $T - n^{\Omega(1)}$ cannot possibly exceed $2^{T - n^{\Omega(1)}}$, which is hopelessly less than any fixed fraction of the number (2^T) of length-T coint-toss sequences, if n is large enough. Therefore, exceptions are rare, and *any* fixed fraction of the remaining designated computations can qualify as *non*exceptions, as desired.

RECEIVED November 15, 1988; FINAL MANUSCRIPT RECEIVED June 8, 1989

References

AANDERAA, S. O. (1974), On k-tape versus $(k-1)$-tape real time computation, *in* "Complexity of Computation, SIAM-AMS Proceedings 7" (R. M. Karp, Ed.), Amer. Math. Soc., Providence, RI, pp. 75–96.

COOK, S. A., AND RACKOFF, C. W. (1980), Space lower bounds for maze threadability on restricted machines, *SIAM J. Comput.* **9**, No. 3, 636–652.

ĎURIŠ, P., GALIL, Z., PAUL, W., AND REISCHUK, R. (1984), Two nonlinear lower bounds for on-line computations, *Inform. and Control* **60**, Nos. 1–3, 1–11.

GÁCS, P. (1974), On the symmetry of algorithmic information, *Sov. Math. Dokl.* **15**, No. 5, 1477–1480.

HENNIE, F. C., AND STEARNS, R. E. (1966), Two-tape simulation of multitape Turing machines, *J. Assoc. Comput. Mach.* **13**, No. 4, 533–546.

KOLMOGOROV, A. N. (1965), Three approaches to the quantitative definition of information, *Problems Inform. Transmission* **1**, No. 1, 1–7.

LI, M., AND VITÁNYI, P. M. B. (1988), Two decades of applied Kolmogorov complexity: In memoriam Andrei Nikolaevich Kolmogorov 1903–1987, *in* "Proceedings, Structure in Complexity Theory Third Annual Conference," pp. 80–101, IEEE Comput. Soc. Press, Washington, DC.

PATURI, R. (1985), "Study of Certain Probabilistic Models of Information Transfer and On-line Models of Computation," Ph.D. thesis, Pennsylvania State University.

PATURI, R., AND SIMON, J. (1983), Lower bounds on the time of probabilistic on-line simulations (preliminary version), *in* "24th Annual Symposium on Foundations of Computer Science," pp. 343–350, IEEE Comput. Society Press, Washington, DC.

PAUL, W. J., SEIFERAS, J. I., AND SIMON, J. (1981), An information-theoretic approach to time bounds for on-line computation, *J. Comput. System Sci.* **23**, No. 2, 108–126.

PAUL, W. (1982), On-line simulation of $k+1$ tapes by k tapes requires nonlinear time, *Inform. and Control* **53**, Nos. 1–2, 1–8.

PIPPENGER, N. (1982), Probabilistic simulations (preliminary version), *in* "Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing," pp. 17–26, Assoc. Comput. Mach., pp. 17–26.

RABIN, M. O. (1963), Real time computation, *Israel J. Math.* **1**, No. 4, 203–211.

ZVONKIN, A. K., AND LEVIN, L. A. (1970), The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms, *Russian Math. Surveys* **25**, No. 6, 83–124.

Printed in Belgium