Separating Nondeterministic Time Complexity Classes

JOEL I. SEIFERAS

The Pennsylvania State University, University Park, Pennsylvania

MICHAEL J. FISCHER

University of Washington, Seattle, Washington

AND

ALBERT R. MEYER

Massachusetts Institute of Technology, Cambridge, Massachusetts

ABSTRACT. A recursive padding technique is used to obtain conditions sufficient for separation of nondeterministic multitape Turing machine time complexity classes If T_2 is a running time and $T_1(n + 1)$ grows more slowly than $T_2(n)$, then there is a language which can be accepted nondeterministically within time bound T_2 but which cannot be accepted nondeterministically within time bound T_1. If even $T_1(n + f(n))$ grows more slowly than $T_2(n)$, where f is the very slowly growing "rounded inverse" of some real-time countable function, then there is such a language over a single-letter alphabet. The strongest known diagonalization results for both deterministic and nondeterministic time complexity classes are reviewed and organized for comparison with the results of the new padding technique

KEY WORDS AND PHRASES: Turing machine, complexity class, complexity hierarchy, time complexity, nondeterminism, padding, recursion theorem, diagonalization, single-letter alphabet

CR CATEGORIES 5 23, 5.25, 5.26, 5 27

1. *Introduction*

Techniques such as those of Meyer and Stockmeyer [21], Meyer [19], Stockmeyer and Meyer [29], Hunt [14], M.J. Fischer and Rabin [8], and Stockmeyer [28] sometimes show that a particular computational task of interest is at least as difficult as any task in some nondeterministic Turing machine time complexity class. For this reason and also because nondeterministic complexity is even less well understood than deterministic complexity, it is of interest to find provably hard computational tasks in each nondeterministic time complexity class. For each rational $r > 1$, for example, Cook [7] (Theorem 3 below) has shown that there is a language (a set of finite strings of symbols from some finite alphabet) that is accepted by some nondeterministic Turing machine within time n^r but by no such machine within time $n^{r-\epsilon}$ for any $\epsilon > 0$ in terms of string length n. Our main results generalize Cook's theorem, even separating the class

This paper represents a portion of the first author's Ph D dissertation [25] written at M.I.T. Project MAC under the supervision of the third author

This work was partially supported by the National Science Foundation under research grant GJ-34671

Authors' addresses. J.I Seiferas, Computer Science Department, 314 Whitmore Laboratory, The Pennsylvania State University, University Park, PA 16802; M J Fischer, Department of Computer Science, FR-35, University of Washington, Seattle, WA 98195, A.R Meyer, Laboratory for Computer Science, NE 43-806, Massachusetts Institute of Technology, Cambridge, MA 02139.

NTIME(n^r) of languages accepted by nondeterministic Turing machines within time n^r from classes as large as NTIME(n^r/log log log n), for example, if $r > 1$ is rational.

We refer to what is usually called a nondeterministic multitape Turing machine [13] simply as a *TM*, and we refer to its deterministic version as a *deterministic TM*. If such an automaton has k tapes (each with a single read-write head), then we call it a *k-tape TM* or a *deterministic k-tape TM*, respectively. We often let a TM receive an *input*, a finite string of symbols from some finite input alphabet Σ, initially written to the right of the head on tape 1, the worktape which we sometimes call the input tape. A TM can act as an *acceptor* by halting in some designated accepting state at the end of some computations. We assume the reader is familiar with how concepts such as these can be formalized. A good single reference for formal definitions relating to Turing machines is [13].

Definition. Let M be any TM acceptor. *M accepts* the string $x \in \Sigma^*$, where Σ^* is the set of all finite strings of symbols from Σ, if there is some computation by M on input x that halts in a designated accepting state. *M accepts* the language $L(M) = \{x | M$ accepts string $x\}$. For $x \in L(M)$, $\text{Time}_M(x)$ is the number of steps in the shortest accepting computation by M on x; for $x \notin L(M)$, $\text{Time}_M(x) = \infty$ by convention.

Definition. A *time bound* is a function $T: N \to R$ with $T(n) \geq n$ for every n, where N is the set of all nonnegative integers and R is the set of reals. In this paper, T, T_1, T_2, etc., always refer to time bounds. The *T-cutoff* of the TM M is the language $L_T(M) = \{x | \text{Time}_M(x) \leq T(|x|)\}$, where $|x|$ denotes the length of the string x. (Note that $L_T(M) \subset L(M)$ because of the convention that $\text{Time}_M(x) = \infty$ for $x \notin L(M)$.) A language L is in *NTIME(T)* if $L = L(M) = L_T(M)$ for some TM acceptor M. Similarly, if M is deterministic and $L = L(M) = L_T(M)$, then L is in *DTIME(T)*. If $L(M) = L_T(M)$, then we say that *M accepts within time T*.

Other slightly different definitions of the NTIME and DTIME complexity classes have been proposed. Book, Greibach, and Wegbreit [4], for example, say that M accepts within time T only if *every* accepting computation on input $x \in L(M)$ reaches the accepting state within $T(|x|)$ steps. Such differences do not affect the complexity classes determined by time bounds of the following type, however; and time bounds of practical interest are of this type.

Definition. If M is a deterministic TM acceptor with $L(M) = \{1\}^*$ and $\text{Time}_M(x) = \lfloor T(|x|) \rfloor \geq |x|$, then T is a *running time*, and M is a *clock* for T.

Downward diagonalization is the best known technique for obtaining separation or "hierarchy" results among the NTIME and DTIME complexity classes (see the Appendix and [11, 12, 6]). For languages over {0, 1}, the following theorem summarizes the best separation results that have been proved by downward diagonalization.

THEOREM 1. *If T_2 is a running time, then each of the following set differences contains a language over* {0, 1}:

(*i*) $DTIME(T_2) - \cup\{DTIME(T_1) | T_2 \notin O(T_1 \log T_1)\}$,
(*ii*) $DTIME(T_2) - \cup\{NTIME(T_1) | \log T_2 \notin O(T_1)\}$,
(*iii*) $NTIME(T_2) - \cup\{DTIME(T_1) | T_2 \notin O(T_1)\}$.[1]

A technicality rules out such strong separation results for languages over a one-letter alphabet. Suppose, for example, that T_2 is a running time with $n \log n \in o(T_2(n))$ and that $L \in \text{DTIME}(T_2)$ is a language over just {1}. If the complement of L is finite, then L is regular (acceptable by a TM acceptor which does not write) and $L \in \text{DTIME}(n)$. If the complement of L is infinite, on the other hand, then our convention that only *acceptance*

[1] For g a nonnegative real-valued function on N, we use the notation $O(g)$ ($o(g)$, respectively) for the class of all nonnegative real-valued functions f on N that satisfy $\limsup(f(n)/g(n)) < \infty$ ($\lim(f(n)/g(n)) = 0$, respectively) as n tends to infinity.

When the precise specification of a time bound is not relevant, we allow an imprecise specification. Thus, in the context of the O and o notations, the base and rounding for the logarithms in Theorems 1, 2, and 2′ need not be specified (See also Lemma 2 below.)

time matters guarantees that $L \in \text{DTIME}(T_1)$ for

$$T_1(n) = \begin{cases} T_2(n) & \text{if } 1^n \in L, \\ n & \text{if } 1^n \notin L. \end{cases}$$

In either case, $L \in \cup\{\text{DTIME}(T_1) | T_2 \notin O(T_1 \log T_1)\}$. Theorems 2 and 2′ show two ways of further restricting T_1 to get separation results for languages over a one-letter alphabet. These theorems and Theorem 1 are proved in the Appendix.

THEOREM 2. *If T_2 is a running time, then each of the following set differences contains a language over* {1}:

(i) $DTIME(T_2) - \cup\{DTIME(T_1) | T_1 log\ T_1 \in o(T_2)\}$,
(ii) $DTIME(T_2) - \cup\{NTIME(T_1) | T_1 \in o(log\ T_2)\}$,
(iii) $NTIME(T_2) - \cup\{DTIME(T_1) | T_1 \in o(T_2)\}$.

THEOREM 2′. *If T_2 is a running time, then each of the following set differences contains a language over* {1}:

(i) $DTIME(T_2) - \cup\{DTIME(T_1) | T_1$ *is a running time*, $T_2 \notin O(T_1 log\ T_1)\}$,
(ii) $DTIME(T_2) - \cup\{NTIME(T_1) | T_1$ *is a running time*, $log\ T_2 \notin O(T_1)\}$,
(iii) $NTIME(T_2) - \cup\{DTIME(T_1) | T_1$ *is a running time*, $T_2 \notin O(T_1)\}$.

COROLLARY 2.1. *For no recursive time bound T does $NTIME(T)$ contain all the recursive languages over* {1}.

2. *Separation of Nondeterministic Time Complexity Classes*

The results obtained by diagonalizing over NTIME classes (part (ii) of each theorem above) are relatively poor. Not even the gross separation result $\text{NTIME}(n^2) \subsetneq \text{NTIME}(2^{n^2})$, for example, follows directly from Theorem 1; yet $\text{DTIME}(n^2) \subsetneq \text{DTIME}(n^2(\log n)^2)$ does follow. Recently, however, Cook [7] proved the following result by a new technique.

THEOREM 3 (Cook). $NTIME(n^r) \subsetneq NTIME(n^s)$ *whenever* $1 \leq r < s$.

We pursue Cook's technical breakthrough, simplifying his proof and generalizing the result. Our proof of the main generalization, Theorem 4 below, makes use of the very gross separation result Corollary 2.1 above and Lemmas 1, 2, 3, and 6 below.

Turing machine design is greatly simplified if we allow more than one head per tape. P. Fischer, Meyer, and Rosenberg [9] have shown that every TM with many heads per tape can be simulated without time loss by a TM with only one head on each of some greater number of tapes. (Furthermore, the simulation preserves determinism.) Using a multihead TM to carry out two computations at the same time leads to results of the following type.

LEMMA 1. *Let M, M' be TM acceptors, and let T be a running time. There are TM acceptors $M \cup M'$, $M \cap M'$, M_T with*

$$\begin{aligned} L(M \cup M') &= L(M) \cup L(M'), & Time_{M \cup M'}(x) &= min\{Time_M(x), Time_{M'}(x)\}; \\ L(M \cap M') &= L(M) \cap L(M'), & Time_{M \cap M'}(x) &= max\{Time_M(x), Time_{M'}(x)\}; \\ L(M_T) &= L_T(M), & Time_{M_T}(x) &= Time_M(x) \text{ for } x \in L_T(M). \end{aligned}$$

PROOF. To design $M \cup M'$ or $M \cap M'$, combine M and M' by providing a second head on the first tape of each and a new input tape with a single head. Use the extra heads to copy the input string at full speed from the new input tape onto the old input tapes. Meanwhile the remaining heads can be used to carry out computations by M and M' on the respective transcribed copies of the input string, even while they are still being transcribed from the real input tape (see Figure 1). To accept $L(M) \cup L(M')$ within the desired time, $M \cup M'$ enters its accepting state when the computation by either M or M' does. To accept $L(M) \cap L(M')$ within the desired time, $M \cap M'$ enters its accepting state

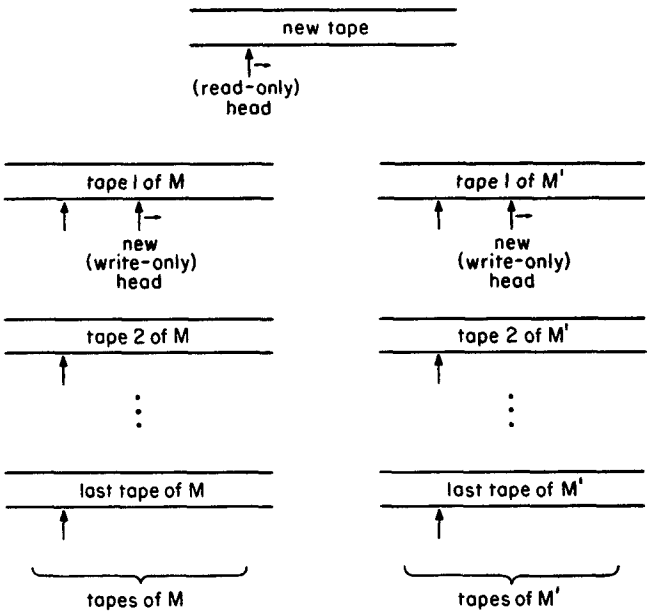

FIG 1. $M \cup M'$ or $M \cap M'$

when the computations by both M and M' have done so. By the result of [9], $M \cup M'$ and $M \cap M'$ can be redesigned without time loss to use only one head per tape.

Because T is a running time, we can modify a clock for T to get a deterministic TM acceptor M'' with $L(M'') = \Sigma^*$ and $\text{Time}_{M''}(x) = \lfloor T(|x|) \rfloor$, where Σ is the input alphabet of M. To design M_T, combine M and M'' in the same way that M and M' were combined above. To accept $L_T(M)$ within the desired time, M_T enters its accepting state when the computation by M enters its accepting state and the computation by M'' has not earlier done so. □

The next lemma indicates that the NTIME complexity classes depend only on time bound growth rates. It also shows that we need at least the condition $T_2 \notin O(T_1)$ to be able to prove $\text{NTIME}(T_2) - \text{NTIME}(T_1) \neq \emptyset$. For T_2 a running time, it follows by Theorem 1 (iii) that, if (contrary to the intuition of most researchers) $\text{DTIME}(T) = \text{NTIME}(T)$ for all T, then $\text{NTIME}(T_2) - \text{NTIME}(T_1) = \text{DTIME}(T_2) - \text{DTIME}(T_1)$ is nonempty *precisely when* $T_2 \notin O(T_1)$.

LEMMA 2. *If* $T_2 \in O(T_1)$, *then* $NTIME(T_2) \subset NTIME(T_1)$.

PROOF. For $T_1(n) \geq (1 + \epsilon)n$ for some $\epsilon > 0$, this is just the constant-factor speedup theorem of Hartmanis and Stearns [11]. The idea is to increase the size of each TM's worktape alphabet so that several steps can be performed in one big step. The limitation is that the reading of the input string cannot be sped up.

That the lemma holds for arbitrary $T_1(n) \geq n$ has been observed by Book and Greibach [3]. The idea is to use nondeterminism to guess the entire input string at the rate of several symbols per step so that the reading of the *guessed* input string *can* be sped up by the method of [11]. Meanwhile additional heads can be used to check the actual input string against the guessed one at the rate of one symbol per step ("full speed"). By the result of [9], the use of more than one head per tape can be eliminated without time loss. □

The following lemma, due to Book, Greibach, and Wegbreit [4], indicates that for nondeterministic time complexity we can get by with TMs having a fixed number of tapes. No similar result is known for deterministic time complexity.

LEMMA 3. *For each TM* M *there is a 2-tape TM* M' *and a constant* c *such that* $L(M')$

$= L(M)$ *and* $Time_{M'}(x) \leq c \cdot Time_M(x)$ *for every* $x \in L(M)$.[2]

PROOF. If M has k tapes, then the "display" of a configuration of M will be a $(k+1)$-tuple consisting of the control state and the k tape symbols scanned in that configuration. The display of a configuration determines which actions are legal as the next move and whether the configuration is an accepting one. The first task for M' is to nondeterministically guess on its second tape an alternating sequence of displays and legal actions by M. The question of whether the sequence describes a legal computation by M on the supplied input is just the question of whether the symbols actually scanned on each tape when the actions are taken agree with the guessed displays. This can be checked independently for each tape in turn by letting the first tape of M' play the role of the tape while running through the guessed sequence of displays and actions. Clearly M' runs for time proportional to the length of the sequence it guesses. For further details the reader is referred to [4]. □

Like Cook's proof of Theorem 3, our proof of the generalization (Theorem 4 below) makes crucial use of a trick called "padding." Acceptance time is measured as a function of *input length*; so if we can increase the lengths of the strings in a language L without significantly changing the time needed to accept the strings, then we get a padded language L' that is less complex than L as we measure complexity relative to input length. One way to pad the language L to L' is to take

$$L' = p(L) = \{x10^k | x \in L, |x10^k| = p(|x|)\}$$

for some $p: N \to N$ with $p(n) > n$. The next lemma shows how such padding reduces complexity.

LEMMA 4. *If* $p(n) > n$ *is a running time, then*

$$p(L) \in NTIME(T) \Leftrightarrow L \in NTIME(T \circ p),$$

where $T \circ p(n) = T(p(n))$.

PROOF. ($\Rightarrow$) Suppose M_1 accepts $p(L)$ within time T. Design M_2 to pad its input string x (which is found at the read-write head on the first worktape) out to $x10^k$, where $|x10^k| = p(|x|)$, and then to compute on input $x10^k$ according to the transition rules of M_1. Because p is a running time, the padding can be done in time proportional to $p(|x|)$. Therefore M_2 accepts L within time proportional to $p(n) + T(p(n)) \leq 2 \cdot T(p(n))$. By Lemma 2 we conclude that $L \in \text{NTIME}(T(p(n)))$.

($\Leftarrow$) Suppose M_2 accepts L within time $T(p(n))$. Design M_1 to check that its input is of the form $x10^k$, where $|x10^k| = p(|x|)$, and then to compute on input x according to the transition rules of M_2. Then certainly $\text{L}(M_1) = p(L)$. Because p is a running time, the padding can be checked in time proportional to the length of the input. Therefore, if $n = |x10^k| = p(|x|)$ and $x \in L$, then $\text{Time}_{M_1}(x10^k)$ is proportional to

$$n + \text{Time}_{M_2}(x) \leq n + T(p(|x|)) = n + T(n) \leq 2 \cdot T(n).$$

By Lemma 2 we conclude that $p(L) = \text{L}(M_1) \in \text{NTIME}(T(n))$. □

The following lemma, used below to prove Corollary 4.2 from Theorem 4, shows how padding of the above type may be used to refine separation results. Ruby and P. Fischer [24] first used this technique in connection with the deterministic time complexity of sequence generation, and Ibarra [15] used it more explicitly in connection with the nondeterministic space complexity of language acceptance (see [25] or [26, 27] for more on space complexity). Ibarra has used similar techniques in other contexts as well [16, 17].

LEMMA 5. *Let sets* $\mathcal{T}_1, \mathcal{T}_2$ *of time bounds be given. Say* $p_1(n) > n, \ldots, p_l(n) > n$ *are*

[2] An idea of [3] allows us to take $c = 1$ if we settle for a 3-tape TM M' (see Lemma 2 above) Aanderaa [1] has shown that we cannot get by with $c = 1$ in the deterministic case no matter what fixed number of tapes we allow M' to have (His counterexample is provided by deterministic TMs which accept in "real time" ($\text{Time}_M(x) = |x|$).)

running times with $T_1 \circ p_{i+1} \in O(T_2 \circ p_i)$ *whenever* $1 \leq i < l$, $T_1 \in \mathcal{T}_1$, $T_2 \in \mathcal{T}_2$. *If* $L \in \cap\{NTIME(T_2 \circ p_l) | T_2 \in \mathcal{T}_2\} - \cup\{NTIME(T_1 \circ p_1) | T_1 \in \mathcal{T}_1\}$, *then* $p_i(L) \in \cap\{NTIME(T_2) | T_2 \in \mathcal{T}_2\} - \cup\{NTIME(T_1) | T_1 \in \mathcal{T}_1\}$ *for some* i.

PROOF. For $1 \leq i \leq l$, let

$$C(l, 1) = \cup\{\text{NTIME}(T_1 \circ p_i) | T_1 \in \mathcal{T}_1\}, \quad C(l, 2) = \cap\{\text{NTIME}(T_2 \circ p_i) | T_2 \in \mathcal{T}_2\}.$$

Suppose $L \in C(l, 2) - C(1, 1)$. By Lemma 2, $\text{NTIME}(T_1 \circ p_{i+1}) \subset \text{NTIME}(T_2 \circ p_i)$ whenever $1 \leq i < l$, $T_1 \in \mathcal{T}_1$, $T_2 \in \mathcal{T}_2$; so, for $1 \leq i < l$,

$$L \in C(i + 1, 1) \Rightarrow L \in C(i, 2).$$

If we were to have also

$$L \in C(i, 2) \Rightarrow L \in C(i, 1)$$

for every i, then we would conclude from $L \in C(l, 2)$ that $L \in C(1, 1)$, a contradiction. For some i, therefore, we must have

$$L \in C(i, 2) - C(i, 1) = \cap\{\text{NTIME}(T_2 \circ p_i) | T_2 \in \mathcal{T}_2\} - \cup\{\text{NTIME}(T_1 \circ p_i) | T_1 \in \mathcal{T}_1\}.$$

By Lemma 4,

$$p_i(L) \in \cap\{\text{NTIME}(T_2) | T_2 \in \mathcal{T}_2\} - \cup\{\text{NTIME}(T_1) | T_1 \in \mathcal{T}_1\} \text{ for that same } i. \quad \square$$

Remark. We do not know how to determine the particular value of i for which $p_i(L) \in \cap\{\text{NTIME}(T_2) | T_2 \in \mathcal{T}_2\} - \cup\{\text{NTIME}(T_1) | T_1 \in \mathcal{T}_1\}$ above. In fact, we do not know how to exhibit any particular language that must be in $\cap\{\text{NTIME}(T_2) | T_2 \in \mathcal{T}_2\} - \cup\{\text{NTIME}(T_1) | T_1 \in \mathcal{T}_1\}$.

It is interesting that the same technique can be applied to DTIME, with a minor restriction, to strengthen the results of diagonalization (Theorem 1 (i)) in some cases. The restriction that each time bound should exceed $(1 + \epsilon) n$ for some positive ϵ allows the deterministic version

$$T_2 \in O(T_1) \Rightarrow \text{DTIME}(T_2) \subset \text{DTIME}(T_1)$$

of Lemma 2 to follow from just [11]. The deterministic versions of Lemmas 4 and 5 then follow as above. We state only the latter.

LEMMA 5D. *Let sets* $\mathcal{T}_1$, $\mathcal{T}_2$ *of time bounds be given, with* $\lim_{n\to\infty} inf(T(n)/n) > 1$ *for each* $T \in \mathcal{T}_1 \cup \mathcal{T}_2$. *Say* $p_1(n) > n, \ldots, p_l(n) > n$ *are running times with* $T_1 \circ p_{i+1} \in O(T_2 \circ p \Leftrightarrow)$ *whenever* $1 \leq i < l$, $T_1 \in \mathcal{T}_1$, $T_2 \in \mathcal{T}_2$. *If* $L \in \cap\{DTIME(T_2 \circ p_1) | T_2 \in \mathcal{T}_2\} - \cup\{DTIME(T_1 \circ p_1) | T_1 \in \mathcal{T}_1\}$, *then* $p_i(L) \in \cap\{DTIME(T_2) | T_2 \in \mathcal{T}_2\} - \cup\{DTIME(T_1) | T_1 \in \mathcal{T}_1\}$ *for some* i.

Example. By Theorem 1 (i),

$$\text{DTIME}(n^2) \subsetneq \text{DTIME}(n^2(\log n)^2).$$

In Lemma 5D, take

$$\mathcal{T}_1 = \{n^2\}, \quad \mathcal{T}_2 = \{n^2(\log n)^{1/200}\}, \quad l = 400, \quad p_i(n) = \lfloor n\ (\log n)^{(i-1)/400} \rfloor.$$

Then conclude that

$$\text{DTIME}(n^2) \subsetneq \text{DTIME}(n^2(\log n)^{1/200}).$$

Another key idea in Cook's proof and our extensions of it involves universal simulation of TMs. So that we may speak with precision about universal simulation, we associate a distinct *program code* from $\{0, 1\}^*$ with each 2-tape TM acceptor having input alphabet $\{0, 1\}$ and worktape alphabet contained in some fixed countably infinite set; we do this in agreement with the easily satisfied conditions listed below. We use the notation L_{pc} for the set of all program codes, and we denote by M_e the 2-tape TM acceptor with program code e.

Condition 1. No program code is a prefix of another, and $L_{p.c} \in \mathrm{DTIME}(n)$.

Condition 2. There is a TM acceptor U (a "universal simulator") with

$$L(U) = \{ex \mid e \in L_{p.c.}, x \in L(M_e)\},$$
$$\mathrm{Time}_U(ex) \leq c_e \cdot \mathrm{Time}_{M_e}(x) \text{ for } e \in L_{p.c.}, x \in L(M_e),$$

where $c_e \geq 1$ depends only on e.

Condition 3. There is a recursive function $f: L_{p.c.} \to L_{p.c.}$ such that $M_{f(e)}$ first writes e at its head on tape 2 and thereafter acts according to the transition rules of M_e. (This condition is a variant of the s_1^1-theorem of recursive function theory [23].)

Most common instruction-by-instruction or state-by-state codings of TM programs can be tailored to satisfy these conditions.

We shall want to pad strings and use the simulator that we design in a recursive control structure. To this end we use Condition 3 to prove one more lemma, a version of the fixed point theorem (Recursion Theorem) of recursive function theory [23].

LEMMA 6. *For each 2-tape TM acceptor M with $L(M) \subset \{0, 1\}^*$, there is a 2-tape TM acceptor M_{e_0} and a constant c with*

$$L(M_{e_0}) = \{x \mid e_0 x \in L(M)\},$$
$$\mathit{Time}_{M_{e_0}}(x) \leq c + \mathit{Time}_M(e_0 x) \text{ for every } x \in L(M_{e_0}).$$

PROOF. Let f be as in Condition 3. Take M_{e_1} to be a 2-tape TM that operates as follows, given x at its head on tape 1 and e at its head on tape 2:

1. Convert e to $f(e)$.
2. Convert x to $f(e)x$, and erase everything else
3. Operate according to the transition rules of M on input $f(e)x$.

Let $e_0 = f(e_1)$. Then by definition M_{e_0} operates as follows on input x:

1. Write e_1 at the head on tape 2
2. Convert e_1 to $f(e_1) = e_0$.
3. Convert x to $e_0 x$
4. Behave like M on $e_0 x$.

Thus

$$x \in L(M_{e_0}) \Leftrightarrow e_0 x \in L(M), \quad \mathrm{Time}_{M_{e_0}}(x) \leq c + \mathrm{Time}_M(e_0 x),$$

where c is the number of steps used in writing e_1, converting e_1 to e_0, and writing e_0 in front of x. □

THEOREM 4. *If T_2 is a running time, then the following set difference contains a language over $\{0, 1\}$:*

$$\mathit{NTIME}(T_2) - \cup\{\mathit{NTIME}(T_1) \mid \textit{there is some recursively bounded but strictly increasing function } f: N \to N \textit{ for which } T_1(f(n+1)) \in o(T_2(f(n)))\}.^3$$

PROOF. Let T_2 be a running time, and let U be a TM acceptor with

$$L(U) = \{ex \mid e \in L_{p.c.}, x \in L(M_e)\},$$
$$\mathrm{Time}_U(ex) \leq c_e \cdot \mathrm{Time}_{M_e}(x) \text{ for } e \in L_{p.c.}, x \in L(M_e),$$

where $c_e \geq 1$ depends only on e, as in Condition 2. By Lemma 1, $L_{T_2}(U) \in \mathrm{NTIME}(T_2)$. Let $f: N \to N$ be any strictly increasing function that is bounded above by some

[3] The operator gap theorem [5, 31] shows that even results such as this are impossible without some "honesty" condition on T_2, such as its being a running time. For example, the operator gap theorem can be used to show that there are arbitrarily large, arbitrarily complex time bounds T for which $\mathrm{NTIME}(T(n))$ *equals* $\mathrm{NTIME}(n \cdot T(n+1))$, even though $T(n+1)$ is certainly a member of $o(n \cdot T(n+1))$.

recursive function. We prove that $L_{T_2}(U) \notin \text{NTIME}(T_1)$ for any time bound T_1 satisfying $T_1(f(n+1)) \in o(T_2(f(n)))$.

Suppose that U_1 does accept $L_{T_2}(U)$ within some time bound T_1 satisfying $T_1(f(n+1)) \in o(T_2(f(n)))$. By Lemma 1, $U' = U_1 \cup U$ accepts $L(U_1 \cup U) = L(U_1) \cup L(U) = L_{T_2}(U) \cup L(U) = L(U)$, and

$$\text{Time}_{U'}(ex) \leq \begin{cases} T_1(|ex|) & \text{if } c_e \cdot \text{Time}_{M_e}(x) \leq T_2(|ex|), \\ c_e \cdot \text{Time}_{M_e}(x) & \text{in any event} \end{cases} \tag{1}$$

for $e \in L_{pc}$, $x \in L(M_e)$. The second inequality holds since $\text{Time}_U(ex) \leq c_e \cdot \text{Time}_{M_e}(x)$ by choice of U, and the first inequality holds because

$$\begin{aligned} \text{Time}_U(ex) \leq c_e \cdot \text{Time}_{M_e}(x) \leq T_2(|ex|) &\Rightarrow ex \in L_{T_2}(U) = L(U_1) \\ &\Rightarrow \text{Time}_{U_1}(ex) \leq T_1(|ex|). \end{aligned}$$

Note that when $T_1(|ex|) < \text{Time}_{M_e}(x) \leq T_2(|ex|)/c_e$, the universal simulator U' will simulate the computation of M_e on x faster than the computation runs directly; i.e. there will be simulation time *gain*. Padding will enable us to exploit this extreme efficiency even for longer computations. Using this idea recursively will lead below to a contradiction of Corollary 2.1.

Let $L \subset \{1\}^*$ be any recursive language over $\{1\}$. Because L is recursive, we can take a running time $T: N \to N$ so large that $L \in \text{NTIME}(T)$. Let M accept L within time T. Design a TM acceptor M' that operates as follows:

1 Check that the input string is of the form $ex0^k$ for some $e \in L_{pc}$, $x \in \{1\}^*$ Condition 1 guarantees that this can be done in time proportional to the length of the input string

2 Use a clock for the running time T to determine whether $k \geq T(|x|)$ This requires at most k steps, so it certainly can be done in time proportional to the length of the input string

3 If $k \geq T(|x|)$, then erase everything but x and compute on input x according to the transition rules of M For $x \in L(M)$, since $\text{Time}_M(x) \leq T(|x|) \leq k$, this step, too, can be performed in time proportional to the length of the input string

4 If $k < T(|x|)$, then pad the input string to $ex0^{k'}$ for some nondeterministically chosen $k' > k$, erase everything else, and compute on input $ex0^{k'}$ according to the transition rules of the universal simulator U' This step can be performed in time proportional to the length of the padded input string $ex0^{k'}$ plus $\text{Time}_{U'}(ex0^{k'})$.

For some constant d_1, we may summarize the behavior and timing of M' as follows:

(i) M' accepts only strings of the form $ex0^k$ for $e \in L_{pc}$, $x \in \{1\}^*$.

(ii) If $k \geq T(|x|)$, then
  (a) $ex0^k \in L(M') \Leftrightarrow x \in L(M)$,
  (b) $\text{Time}_{M'}(ex0^k) \leq d_1 \cdot |ex0^k|$ for $ex0^k \in L(M')$.

(iii) If $k < T(|x|)$, then
  (a) $ex0^k \in L(M') \Leftrightarrow ex0^{k'} \in L(U')$ for some $k' > k$,
  (b) $\text{Time}_{M'}(ex0^k) \leq d_1 \cdot |ex0^{k'}| + \text{Time}_{U'}(ex0^{k'})$ for every $k' > k$.

Applying Lemma 3 to obtain a 2-tape TM that accepts $L(M')$ with only a constant factor time loss, and then applying the Recursion Theorem (Lemma 6) to this machine, we get a program code e_0 for a 2-tape TM that accepts $L(M_{e_0}) = \{x0^k | e_0 x0^k \in L(M')\}$ within time $\text{Time}_{M_{e_0}}(x0^k) \leq d_2 \cdot \text{Time}_{M'}(e_0 x0^k)$ for some constant d_2.

CLAIM 1. *For each string $x \in \{1\}^*$, the following holds for every k:*

$$x0^k \in L(M_{e_0}) \Leftrightarrow x \in L(M).$$

PROOF. For each x, we establish the claim by induction on k running *down* from $k \geq T(|x|)$ to $k = 0$.

$k \geq T(|x|)$:

$$\begin{aligned} x0^k \in L(M_{e_0}) &\Leftrightarrow e_0 x0^k \in L(M') \quad \text{(by choice of } e_0\text{)} \\ &\Leftrightarrow x \in L(M) \quad \text{(by (ii-a))}. \end{aligned}$$

$k < T(|x|)$: Assume $x0^{k'} \in \mathrm{L}(\mathrm{M}_{e_0}) \Leftrightarrow x \in \mathrm{L}(M)$ holds for every $k' > k$. Then

$$\begin{aligned} x0^k \in \mathrm{L}(\mathrm{M}_{e_0}) &\Leftrightarrow e_0x0^k \in \mathrm{L}(M') && \text{(by choice of } e_0\text{)} \\ &\Leftrightarrow e_0x0^{k'} \in \mathrm{L}(U') \text{ for some } k' > k && \text{(by (iii-a))} \\ &\Leftrightarrow x0^{k'} \in \mathrm{L}(\mathrm{M}_{e_0}) \text{ for some } k' > k && \text{(because } e_0 \in \mathrm{L}_{\mathrm{p.c}}\text{)} \\ &\Leftrightarrow x \in \mathrm{L}(M) && \text{(by induction hypothesis).} \ \square \end{aligned}$$

CLAIM 2. *For each sufficiently long string* $x \in L(M)$, $\mathit{Time}_{M_{e_0}}(x) \leq T_2(f(|e_0x|))$.

PROOF. Let $d_3 = c_{e_0} \cdot d_2 \cdot (d_1 + 1)$. By the "translational" hypothesis $T_1(f(n + 1)) \in o(T_2(f(n)))$, $d_3 \cdot T_1(f(n + 1)) \leq T_2(f(n))$ for every sufficiently large n. Let $x \in \mathrm{L}(M)$ be so long that $d_3 \cdot T_1(f(n + 1)) \leq T_2(f(n))$ for every $n \geq |e_0x|$.

Assuming we could show

$$\mathrm{Time}_{U'}(e_0x0^{f(|e_0x|+1)-|e_0x|}) \leq T_1(f(|e_0x| + 1)),$$

we could reason as follows:

$$\begin{aligned} \mathrm{Time}_{\mathrm{M}_{e_0}}(x) &\leq d_2 \cdot \mathrm{Time}_{M'}(e_0x) && \text{(by choice of } e_0\text{)} \\ &\leq d_2 \cdot (d_1 \cdot f(|e_0x| + 1) + \mathrm{Time}_{U'}(e_0x0^{f(|e_0x|+1)-|e_0x|}) \\ &\qquad \text{(by (iii-b) since } f(|e_0x| + 1) > |e_0x|\text{)} \\ &\leq d_2 \cdot (d_1 \cdot f(|e_0x| + 1) + T_1(f(|e_0x| + 1))) \\ &\leq d_3 \cdot T_1(f(|e_0x| + 1)) \\ &\qquad \text{(since } T_1 \text{ being a time bound implies } f(|e_0x| + 1) \leq T_1(f(|e_0x| + 1))\text{)} \\ &\leq T_2(f(|e_0x|)) \quad \text{(because } x \text{ is so long).} \end{aligned}$$

To prove

$$\mathrm{Time}_{U'}(e_0x0^{f(|e_0x|+1)-|e_0x|}) \leq T_1(f(|e_0x| + 1)),$$

we prove more generally that

$$\mathrm{Time}_{U'}(e_0x0^{f(n)-|e_0x|}) \leq T_1(f(n))$$

for every $n \geq |e_0x|$. We do this by induction on n running down from $n \geq |e_0x|$ so large that $f(n) \geq |e_0x| + T(|x|)$ to $n = |e_0x|$.

$n \geq |e_0x|$ and $f(n) \geq |e_0x| + T(|x|)$:

$$\begin{aligned} c_{e_0} \cdot \mathrm{Time}_{\mathrm{M}_{e_0}}(x0^{f(n)-|e_0x|}) &\leq c_{e_0} \cdot d_2 \cdot \mathrm{Time}_{M'}(e_0x0^{f(n)-|e_0x|}) && \text{(by choice of } e_0\text{)} \\ &\leq c_{e_0} \cdot d_2 \cdot d_1 \cdot f(n) && \text{(by (ii-b))} \\ &\leq d_3 \cdot T_1(f(n + 1)) && \text{(since } f(n) < f(n + 1) \\ &&& \quad \leq T_1(f(n + 1))\text{)} \\ &\leq T_2(f(n)) && \text{(because } n \geq |e_0x|\text{).} \end{aligned}$$

Therefore $\mathrm{Time}_{U'}(e_0x0^{f(n)-|e_0x|}) \leq T_1(f(n))$, by (1).

$|e_0x| \leq n \leq f(n) < |e_0x| + T(|x|)$: Assume $\mathrm{Time}_{U'}(e_0x0^{f(n+1)-|e_0x|}) \leq T_1(f(n + 1))$. Then

$$\begin{aligned} c_{e_0} \cdot \mathrm{Time}_{\mathrm{M}_{e_0}}(x0^{f(n)-|e_0x|}) &\leq c_{e_0} \cdot d_2 \cdot \mathrm{Time}_{M'}(e_0x0^{f(n)-|e_0x|}) && \text{(by choice of } e_0\text{)} \\ &\leq c_{e_0} \cdot d_2 \cdot (d_1 \cdot f(n + 1) + \mathrm{Time}_{U'}(e_0x0^{f(n+1)-|e_0x|})) \\ &\qquad \text{(by (iii-b) since } f(n + 1) > f(n)\text{)} \\ &\leq c_{e_0} \cdot d_2 \cdot (d_1 \cdot f(n + 1) + T_1(f(n + 1))) \\ &\leq d_3 \cdot T_1(f(n + 1)) \quad \text{(since } f(n + 1) \leq T_1(f(n + 1))\text{)} \\ &\leq T_2(f(n)) \quad \text{(because } n \geq |e_0x|\text{).} \end{aligned}$$

Therefore $\mathrm{Time}_{U'}(e_0x0^{f(n)-|e_0x|}) \leq T_1(f(n))$, by (1). $\square$

Finally, by Lemma 1, M_{e_0} can be modified without time loss to reject padded inputs (those not members of $\{1\}^*$) and to quickly agree with M on short ones (those not

sufficiently long for Claim 2). This gives a TM that accepts $L = L(M)$ within time

$$T_2(f(|e_0| + n)) \in O\left(\sum_{n' \le 2n} T_2(f(n'))\right);$$

so $L \in \text{NTIME}(\sum_{n' \le 2n} T_2(f(n')))$ by Lemma 2. Since the latter time bound is recursively bounded (because both f and T_2 are) and independent of the particular recursive language $L \subset \{1\}^*$, this contradicts Corollary 2.1. □

Example. For an arbitrary set A of nonnegative integers, let

$$\delta(2n) = \begin{cases} 1 & \text{if } n \in A, \\ n & \text{if } n \notin A; \end{cases} \qquad \delta(2n+1) = \begin{cases} n & \text{if } n \in A, \\ 1 & \text{if } n \notin A. \end{cases}$$

To see that $\text{NTIME}(n^2 \cdot \delta(n)) \subsetneq \text{NTIME}(n^3)$, just apply Theorem 4 with

$$f(n) = \begin{cases} 2n & \text{if } n \in A, \\ 2n+1 & \text{if } n \notin A, \end{cases}$$

so that $\delta(f(n+1)) = 1$ for every n.

In many applications it suffices to have Theorem 4 for the single function $f(n) = n$, especially if we are concerned only with nondecreasing time bounds.

COROLLARY 4.1. *If T_2 is a running time, then*

$$NTIME(T_2) - \cup\{NTIME(T_1) | T_1(n+1) \in o(T_2(n))\}$$

contains a language over $\{0, 1\}$.

The informal diagram in Figure 2 illustrates how the proof of Theorem 4 uses padding to take advantage of deeply nested simulations by U' to bring the time for an arbitrary computation down to the vicinity of T_1 and T_2 in the case $f(n) = n$ of Corollary 4.1. The direct computation on x, up around the level of $T(|x|)$, is brought down to below T_2 *in terms of the input length* by padding x out to $x0^{T(|x|)}$. By the hypothesized nature of U', simulating that computation brings its time down to below T_1. If we unpad by a single 0, then the hypothesis that $T_1(n+1)$ is small compared to $T_2(n)$ keeps the computation still below T_2 in terms of the input length. A simulation by U' of this computation on $x0^{T(|x|)-1}$ brings *its* time down to below T_1. Continuing to nest alternating unpaddings and simulations finally yields a computation on the original input string x down at the level of T_1 and T_2.

The "translational" condition $T_1(n+1) \in o(T_2(n))$ of Corollary 4.1 is a severe one for a rapidly growing running time T_2. When $T_2(n+1)$ is worse than exponential in $T_2(n)$, in fact, deterministic downward diagonalization within time bound T_2 (Theorem 1) yields stronger results than does Corollary 4.1. Because Corollary 4.1 applies for $T_1(n+1) \in o(T_2(n))$ and Theorem 1 applies for $\log T_2(n) \notin O(T_1(n))$, Corollary 4.1 contributes new results precisely when $\log T_2(n+1) \in o(T_2(n))$.

To see the strength of Corollary 4.1, let

$$\log^* n = \min\{k | \underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{k} \ge n\}.$$

For constants $c > 1$ and $r \ge 1$ whose digits in radix notation can be generated rapidly, and in particular for rational c and r, note that n^r, $n^r \cdot \log^* n$, $n^r \cdot (\log^* n)^2$, c^n, $c^n \cdot \log^* n$, etc., are running times. Thus we conclude that

$$\begin{aligned} &\text{NTIME}(n^r) \subsetneq \text{NTIME}(n^r \cdot \log^* n) \subsetneq \text{NTIME}(n^r \cdot (\log^* n)^2) \subsetneq \cdots, \\ &\text{NTIME}(c^n) \subsetneq \text{NTIME}(c^n \cdot \log^* n) \subsetneq \text{NTIME}(c^n \cdot (\log^* n)^2) \subsetneq \cdots. \end{aligned}$$

These results do not follow immediately from Cook's result (Theorem 3) or by diagonalization (Theorem 1).

It is interesting to note that the containments corresponding to the examples above are not known to be proper for deterministic time (DTIME). The fundamental reason

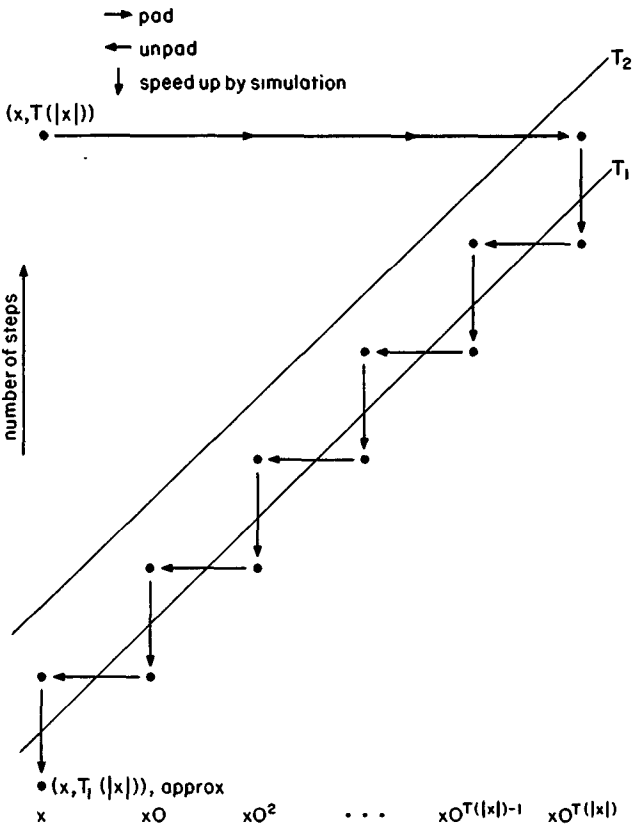

FIG 2. Intuitive proof of Corollary 4 1

is that Lemma 3 is not known for DTIME. If that lemma were known for DTIME, then downward diagonalization would give generally stronger results for DTIME than would the proof of Theorem 4 anyway.[4]

Corollary 4 1 obviously implies that

$$\mathrm{NTIME}(2^{n^2}) \subsetneqq \mathrm{NTIME}(2^{(n+1)^2}\cdot \log^* n), \quad \mathrm{NTIME}(2^{2^n}) \subsetneqq \mathrm{NTIME}(2^{2^{n+1}}\cdot \log^* n).$$

In fact we can strengthen these results to

$$\mathrm{NTIME}(2^{n^2}) \subsetneqq \mathrm{NTIME}(2^{(n+1)^2}), \quad \mathrm{NTIME}(2^{2^n}) \subsetneqq \mathrm{NTIME}(2^{2^{n+1}}),$$

by appeal to the following corollary.

COROLLARY 4.2. *If T_2 is a running time, then*

$$\cup\{NTIME(T_1) | T_1(n+1) \in O(T_2(n)),\ T_1(n) \in o(T_2(n))\} \subsetneqq NTIME(T_2),$$

and there is a language over $\{0, 1\}$ that bears witness to this fact.

PROOF. Because $T_1(n) \in o(T_2(n))$ implies $T_1((n+1)+1) \in o(T_2(n+2))$, Corollary 4.1 gives a language $L \subset \{0, 1\}^*$ in

$$\mathrm{NTIME}(T_2(n+2)) - \cup\{\mathrm{NTIME}(T_1(n+1)) |\ T_1(n+1) \in O(T_2(n)),\ T_1(n) \in o(T_2(n))\}.$$

Applying Lemma 5 with

[4] In the light of Lemma 3, another point of view is that our results separate the nondeterministic time complexity classes determined by *k-tape* TMs, for any fixed $k \geq 2$ W J Paul has shown recently [22] that separation results as strong as our examples do happen to hold for the analogous deterministic classes

$$\mathcal{T}_1 = \{T_1 \mid T_1(n+1) \in O(T_2(n)),\ T_1(n) \in o(T_2(n))\},\quad \mathcal{T}_2 = \{T_2\},$$
$$p_1(n) = n+1,\quad p_2(n) = n+2,$$

we conclude that either $p_1(L)$ or $p_2(L)$ is a member of

$$\mathrm{NTIME}(T_2) - \cup\{\mathrm{NTIME}(T_1) \mid T_1(n+1) \in O(T_2(n)),\ T_1(n) \in o(T_2(n))\}.$$

Containment holds by Lemma 2 □

Remarks. (i) Lemma 5 goes through equally well if we pad to the left rather than to the right. For this remark, therefore, we may assume that $p_i(L) = \{0^k 1x \mid x \in L,\ |0^k 1x| = p_i(|x|)\}$ for $i = 1, 2$ above.

For U the universal simulator of Condition 2, $L_{T_2(n)}(U)$ serves as a witness language for Theorem 4 and Corollary 4.1. One might naturally suspect, therefore, that $L_{T_2(n)}(U)$ would be a witness language for Corollary 4.2 as well. In the proof of Corollary 4.2, $L = L_{T_2(n+2)}(U)$ satisfies the condition for choosing L. If we slightly modify our program coding by concatenating a single 1 in front of each old program code and if we let V be the naturally derived new universal simulator, then we do get $L_{T_2(n+1)}(V) = 1 \cdot L_{T_2(n+2)}(U) = \{1x \mid x \in L_{T_2(n+2)}(U)\} = p_1(L)$. Similarly, if we further concatenate a 0 in front of each program code and let W be derived from V by taking this into account, then we get $L_{T_2(n)}(W) = 01 \cdot L_{T_2(n+2)}(U) = p_2(L)$. Yet we can show only that *either* $L_{T_2(n)}(W)$ or $L_{T_2(n+1)}(V)$ is a witness to Corollary 4.2. We do not know whether there is necessarily a witness language of the form $L_{T_2(n)}(U)$ and whether the particular choice of program coding and universal simulator U affects whether $L_{T_2(n)}(U)$ is such a language.

(ii) Corollary 4.2 contributes new results (over Theorem 1) precisely when $\log T_2(n+1) \in O(T_2(n))$.

3. *Separation by Unary Languages*

Padding strings over a one-letter alphabet by one symbol at a time does not leave them decodable; so we cannot hope to use our method to get a result as strong as Corollary 4.2 for languages over a one-letter alphabet. Our final theorem, Theorem 5 below, demonstrates that we can come very close, however.

Definition. The *rounded inverse* of a strictly increasing function $f: N \to N$ is the function $\lceil f^{-1} \rceil: N \to N$ defined by

$$\lceil f^{-1} \rceil(n) = \min\{k \mid f(k) \geq n\}.$$

Examples.

function	*rounded inverse*
n^2	$\lceil n^{1/2} \rceil$
2^n	$\lceil \log_2 n \rceil$
$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n}$	$\log^* n$

LEMMA 7. *Let $g: \{0,1\}^* \to N - \{0\}$ be the bijection which maps each binary word x to the integer whose binary representation (high-order bit first) is $1x$. For $f: N \to N$ real× time countable,*[5] *define $h: \{0,1\}^* \to N$ inductively by*

$$h(x) = \begin{cases} f(g(x)) + g(x) + 1 & \text{if } x \text{ is not of the form } y0, \\ h(y) + \lceil f^{-1} \rceil (h(y)) & \text{if } x = y0. \end{cases}$$

Then h is an injection, and a deterministic TM can compute $1^{h(x)}$ from x or x from $1^{h(x)}$ within time $2 \cdot h(x)$.

[5] A strictly increasing function $f\ N \to N$ is *real-time countable* [30] if some deterministic Turing machine generates the characteristic sequence of the range of f in real time (i.e one character per step). (The characteristic sequence has a 1 in position n if n is in the range of f and a 0 otherwise.)

PROOF. In the case that neither x nor x' is of the form $y0$, we have

$$h(x) = h(x') \text{ only if } x = x'$$

because g is an injection and $f(n) + n + 1$ is strictly increasing. In the case that $x = y0$ and $x' = y'0$, we have

$$h(x) = h(x') \text{ only if } h(y) = h(y')$$

because $n + \lceil f^{-1} \rceil(n)$ is strictly increasing. Unless there are strings $x = y0$ and x' not of that form with $h(x) = h(x')$, therefore, h must be an injection. For such strings to exist, the ranges of the strictly increasing functions $f(n) + n + 1$ and $n + \lceil f^{-1} \rceil(n)$ must intersect. For every n, however,

$$\begin{aligned} f(n) + \lceil f^{-1} \rceil(f(n)) &= f(n) + n < f(n) + n + 1 \\ &< (f(n) + 1) + (n + 1) = (f(n) + 1) + \lceil f^{-1} \rceil(f(n) + 1); \end{aligned}$$

so the ranges are disjoint and h is an injection.

By the constant-factor speedup technique of [11] and Lemma 2 above, it remains only to prove that a deterministic TM can perform the indicated conversions within time *proportional to* the indicated time.

Let us first consider the conversion of an arbitrary string $x0^k$, where x is not of the form $y0$, to $1^{h(x0^k)}$. A deterministic TM can first compute $1^{g(x)}$ by converting the binary integer $1x$ to unary. It can do this in time proportional to $g(x) \leq h(x) \leq h(x0^k)$. Because f is real-time countable, the deterministic TM can then compute $1^{f(g(x))}$ within time proportional to $f(g(x)) \leq h(x) \leq h(x0^k)$. The machine can then combine these intermediate results to get $1^{h(x)} = 1^{f(g(x))+g(x)+1}$, still within time proportional to $h(x) \leq h(x0^k)$.

The final conversion to $1^{h(x0^k)}$ is slightly more difficult. One way to compute $1^{h(x0^k)}$ from $1^{h(x)}$ and 0^k is to generate and use a table of the values of $\lceil f^{-1} \rceil$ at arguments up to $h(x0^{k-1})$. (Find the value $\lceil f^{-1} \rceil(h(x))$, compute $1^{h(x0)} = 1^{h(x)+1+\lceil f^{-1} \rceil(h(x))}$; find the value $\lceil f^{-1} \rceil(h(x0))$, compute $1^{h(x0^2)} = 1^{h(x0)+1+\lceil f^{-1} \rceil(h(x0))}$; etc.) Since $h(x0^{l+1}) - h(x0^l) = \lceil f^{-1} \rceil(h(x0^l))$, sequential storage of the values in the table would make it easy to go from the $h(x0^l)$-th value i ($i = \lceil f^{-1} \rceil(h(x0^l))$) to the $h(x0^{l+1})$-th value: Just skip to the ith following value. Successive values of $\lceil f^{-1} \rceil$ differ by at most 1; so a table of one-bit values actually suffices, the nth bit ($n = 0, 1, 2, \ldots$) telling whether $\lceil f^{-1} \rceil$ increases at argument $n + 1$, and the number of positive bits preceding bit n therefore being equal to $\lceil f^{-1} \rceil(n)$. This table is just the characteristic sequence of the range of f; so it can be generated in real time. The skipping can be done in linear time (in the number of skips) by maintaining, on a separate tape, a unary count of the number of positive bits preceding the currently scanned bit of the table. Thus $1^{h(x0^k)}$ can be computed from $1^{h(x)}$ within time proportional to $h(x0^k)$.

Before we describe the reverse conversion, observe that $n + \lceil f^{-1} \rceil(n)$ is a strictly increasing function of n and that $n + \lceil f^{-1} \rceil(n) > n$ if and only if $n > f(0)$. For each $n > f(0)$, it follows that there is at most one n' for which $n' + \lceil f^{-1} \rceil(n') = n$ and that any such n' must satisfy $n > n' > f(0)$. For each $n_0 > f(0)$, therefore, there is a unique sequence

$$n_0 > n_1 > \cdots > n_k > m \geq 0$$

such that

$$n_i + \lceil f^{-1} \rceil(n_i) = n_{i-1} \text{ for } 1 \leq i \leq k, \quad m + \lceil f^{-1} \rceil(m) < n_k < (m + 1) + \lceil f^{-1} \rceil(m + 1).$$

Since $\lceil f^{-1} \rceil(m + 1) - \lceil f^{-1} \rceil(m) \in \{0, 1\}$, the latter pair of inequalities implies

$$\lceil f^{-1} \rceil(m + 1) - \lceil f^{-1} \rceil(m) = 1$$

and hence

$$f(\lceil f^{-1} \rceil(m)) = m.$$

By the pair of inequalities, therefore,

$$n_k = m + \lceil f^{-1}\rceil(m) + 1 = f(\lceil f^{-1}\rceil(m)) + \lceil f^{-1}\rceil(m) + 1.$$

By definition, therefore, $n_0 = h(x0^k)$ if $\lceil f^{-1}\rceil(m) = g(x)$ for x not of the form $y0$. Conversely, if

$$n_0 \leq f(0) \quad \text{or}$$
$$(n_0 > f(0) \;\&\; \lceil f^{-1}\rceil(m) = 0) \quad \text{or}$$
$$(n_0 > f(0) \;\&\; \lceil f^{-1}\rceil(m) > 0 \;\&\; \lceil f^{-1}\rceil(m) = g(y0)),$$

then n_0 must not be in the range of h. (If n_0 were equal to $h(x0^k)$ for x not of the form $y0$, then we would have

$$n_0 \geq h(x) = f(g(x)) + g(x) + 1 > f(0),$$

and the unique sequence for n_0 would have to be

$$h(x0^k), h(x0^{k-1}), \ldots, h(x), f(g(x)).)$$

To determine whether n_0 is in the range of h and to calculate $h^{-1}(n_0)$ if it is, it therefore suffices to execute the following program:

1 Check whether $n_0 > f(0)$ If not, then halt, $h^{-1}(n_0)$ does not exist

2. Calculate the length k and the end m of the sequence starting at n_0

3 Check whether $\lceil f^{-1}\rceil(m) > 0$ If not, then halt; $h^{-1}(n_0)$ does not exist

4 Calculate $x = g^{-1}(\lceil f^{-1}\rceil(m))$

5 Check whether x is of the form $y0$. If so, then halt, $h^{-1}(n_0)$ does not exist. If not, then $h^{-1}(n_0) = x0^k$.

Trivially, a deterministic TM can execute steps 1, 3, and 5 in time proportional to n_0. It is straightforward to execute step 4 in time proportional to $m < n_0$. It remains only to describe how a deterministic TM can execute step 2 in time proportional to n_0.

The TM for step 2 starts by counting up to position n_0 of the table of bits used above in the encoding process. It records in unary the number of positive bits passed in the process; this number is $\lceil f^{-1}\rceil(n_0)$. Given position n_i in the table and $\lceil f^{-1}\rceil(n_i)$ in unary, the TM finds the next position in the sequence by skipping to preceding positions p in the table until either $p + \lceil f^{-1}\rceil(p) = n_i$ (in which case $p = n_{i+1}$) or $p + \lceil f^{-1}\rceil(p) < n_i$ (in which case $p = m$). (One or the other certainly must occur for some $p \geq 0$.) Substituting $\lceil f^{-1}\rceil(p) = \lceil f^{-1}\rceil(n_i) -$ (the number of positive bits reached while skipping) gives the termination condition $(n_i - p) +$ (the number of positive bits reached while skipping) $\geq \lceil f^{-1}\rceil(n_i)$; i.e. the skipping should continue until the number of skips, plus the number of positive bits reached while skipping, equals or exceeds $\lceil f^{-1}\rceil(n_i)$. It follows that the next table position in the sequence and the corresponding value of $\lceil f^{-1}\rceil$ can be found in time proportional to the number of skips necessary. Therefore the length k and the end m of the entire sequence can be determined in time proportional to n_0, as required. □

THEOREM 5. *If T_2 is a running time and f is real-time countable, then there is a language over $\{1\}$ in*

$$NTIME(T_2) - \cup\{NTIME(T_1) \mid T_1(n + \lceil f^{-1}\rceil(n)) \in o(T_2(n))\}.$$

PROOF. Let T_2 be a running time, and let f be real-time countable. We start with U as in the proof of Theorem 4; i.e.

$$L(U) = \{ex \mid e \in L_{p.c.}, x \in L(M_e)\},$$
$$\text{Time}_U(ex) \leq c_e \cdot \text{Time}_{M_e}(x) \quad \text{for} \quad e \in L_{p.c.}, x \in L(M_e),$$

where $c_e \geq 1$ depends only on e. To adapt the earlier proof, however, we must construct a witness language as the T_2 cutoff of some other "universal simulator" V having input alphabet just $\{1\}$. It is to this end that we define an injection $h: \{0, 1\}^* \to N$ from f as in Lemma 7.

From U we construct V to operate as follows on input $y \in \{1\}^*$:

1 Find x with $1^{h(x)} = y$ if it exists

2. Compute on x according to the transition rules of U

By Lemma 1, $L_{T_2}(V) \in \text{NTIME}(T_2)$. We prove that $L_{T_2}(V) \notin \text{NTIME}(T_1)$ for any time bound T_1 satisfying $T_1(n + \lceil f^{-1}\rceil(n)) \in o(T_2(n))$.

Suppose that V_1 does accept $L_{T_2}(V)$ within some time bound T_1 satisfying $T_1(n + \lceil f^{-1}\rceil(n)) \in o(T_2(n))$. By Lemma 1, $V' = V_1 \cup V$ accepts $L(V_1 \cup V) = L(V_1) \cup L(V) = L_{T_2}(V) \cup L(V) = L(V)$, and

$$\text{Time}_{V'}(1^{h(ex)}) \leq \begin{cases} T_1(h(ex)) & \text{if } 2\cdot h(ex) + c_e \cdot \text{Time}_{M_e}(x) \leq T_2(h(ex)), \\ 2\cdot h(ex) + c_e \cdot \text{Time}_{M_e}(x) & \text{in any event} \end{cases}$$

for $e \in L_{\text{p.c.}}$, $x \in L(M_e)$. From V' we construct U' to operate as follows on input $x \in \{0, 1\}^*$:

1. Compute $1^{h(x)}$.

2 Compute on $1^{h(x)}$ according to the transition rules of V'

Then $L(U') = \{x \mid 1^{h(x)} \in L(V')\} = \{x \mid 1^{h(x)} \in L(V)\} = L(U)$, and

$$\begin{aligned} \text{Time}_{U'}(ex) &\leq 2\cdot h(ex) + \text{Time}_{V'}(1^{h(ex)}) \\ &\leq \begin{cases} 2\cdot h(ex) + T_1(h(ex)) & \text{if } 2\cdot h(ex) + c_e \cdot \text{Time}_{M_e}(x) \leq T_2(h(ex)), \\ 4\cdot h(ex) + c_e \cdot \text{Time}_{M_e}(x) & \text{in any event} \end{cases} \end{aligned}$$

for $e \in L_{\text{p.c.}}$, $x \in L(M_e)$.

For any recursive $L \subset \{1\}^*$, we can use U' as in the proof of Theorem 4 to get a 2-tape TM acceptor M_{e_0} for $\{x0^k \mid x \in L, k \in N\}$, with

$$\text{Time}_{M_{e_0}}(x0^k) \leq \begin{cases} d\cdot|e_0x0^k| & \text{if } k \geq T(|x|), \\ d\cdot|e_0x0^{k+1}| + d\cdot\text{Time}_{U'}(e_0x0^{k+1}) & \text{if } k < T(|x|) \end{cases}$$

for some sufficiently large constant d and some appropriate time bound T.

CLAIM. *For each sufficiently long string $x \in L$, $\text{Time}_{M_{e_0}}(x) \leq T_2(h(e_0x))$.*

PROOF. Let $x \in L$ be so long that

$$(2 + c_{e_0}\cdot 4d)\cdot T_1(n + \lceil f^{-1}\rceil(n)) \leq T_2(n)$$

for every $n \geq h(e_0x)$. Then certainly

$$4d\cdot T_1(h(e_0x0)) = 4d\cdot T_1(h(e_0x) + \lceil f^{-1}\rceil(h(e_0x))) \leq T_2(h(e_0x));$$

so it suffices to prove $\text{Time}_{M_{e_0}}(x) \leq 4d\cdot T_1(h(e_0x0))$. In fact we prove by induction on k running down from $k \geq T(|x|)$ to $k = 0$ that

$$\text{Time}_{M_{e_0}}(x0^k) \leq 4d\cdot T_1(h(e_0x0^{k+1})).$$

$k \geq T(|x|)$: Using the facts $|y| \leq h(y) \leq T_1(h(y))$ for $y = e_0x0^{k+1}$, we have

$$\text{Time}_{M_{e_0}}(x0^k) \leq d\cdot|e_0x0^k| \leq 4d\cdot T_1(h(e_0x0^{k+1})).$$

$k < T(|x|)$: Assume $\text{Time}_{M_{e_0}}(x0^{k+1}) \leq 4d\cdot T_1(h(e_0x0^{k+2}))$. Then

$$\begin{aligned} &2\cdot h(e_0x0^{k+1}) + c_{e_0}\cdot\text{Time}_{M_{e_0}}(x0^{k+1}) \\ &\qquad \leq 2\cdot h(e_0x0^{k+1}) + c_{e_0}\cdot 4d\cdot T_1(h(e_0x0^{k+2})) \\ &\qquad = 2\cdot h(e_0x0^{k+1}) + c_{e_0}\cdot 4d\cdot T_1(h(e_0x0^{k+1}) + \lceil f^{-1}\rceil(h(e_0x0^{k+1}))) \\ &\qquad \leq (2 + c_{e_0}\cdot 4d)\cdot T_1(h(e_0x0^{k+1}) + \lceil f^{-1}\rceil(h(e_0x0^{k+1}))) \\ &\qquad \leq T_2(h(e_0x0^{k+1})). \end{aligned}$$

Therefore

$$\text{Time}_{U'}(e_0x0^{k+1}) \leq 2\cdot h(e_0x0^{k+1}) + T_1(h(e_0x0^{k+1})) \leq 3\cdot T_1(h(e_0x0^{k+1})).$$

Therefore

$$\begin{aligned}\text{Time}_{M_{e_0}}(x0^k) &\leq d \cdot |e_0 x 0^{k+1}| + d \cdot \text{Time}_{U'}(e_0 x 0^{k+1}) \\ &\leq d \cdot |e_0 x 0^{k+1}| + 3d \cdot T_1(h(e_0 x 0^{k+1})) \leq 4d \cdot T_1(h(e_0 x 0^{k+1})).\end{aligned}$$ □

By Lemma 1, M_{e_0} can be modified without time loss to reject padded inputs. This gives a TM that accepts L within a time bound of $O(\Sigma_{|x|\leq 2n} T_2(h(x)))$; so $L \in \text{NTIME}(\Sigma_{|x|\leq 2n} T_2(h(x)))$ by Lemma 2. Since the latter time bound is recursively bounded and independent of the particular recursive language $L \subset \{1\}^*$, this contradicts Corollary 2.1. □

Example. Taking $f(n) = \underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{2^{2^n}}$, we get a language over $\{1\}$ in $\text{NTIME}(2^n \cdot \log^* n) - \text{NTIME}(2^n)$.

4. *Open Questions*

1. For T_2 a running time, is the condition $T_2 \notin O(T_1)$ enough in general for separation between $\text{NTIME}(T_1)$ and $\text{NTIME}(T_2)$ or between $\text{DTIME}(T_1)$ and $\text{DTIME}(T_2)$?

2. Is there an actual difference between the separation results that hold for NTIME and those that hold for DTIME? Is $\text{DTIME}(n^2) \subsetneq \text{DTIME}(n^2 \cdot \log \log n)$? Is $\text{NTIME}(2^{2^n}) \subsetneq \text{NTIME}(2^{2^{n+1}}/\log^* n)$? Is there a language over a *one-letter alphabet* in $\text{NTIME}(2^{2^{n+1}}) - \text{NTIME}(2^{2^n})$?

3. What is the relationship between NTIME and DTIME? Does $\text{NTIME}(T) = \text{DTIME}(T)$?

4. That a language L is not a member of $\text{NTIME}(T_1)$ means only that every acceptor M for L has $\text{Time}_M(x) > T_1(|x|)$ for strings $x \in L$ of *infinitely many lengths*. Stronger senses of lower bounds, requiring that $\text{Time}_M(x) > T_1(|x|)$ for strings $x \in L$ of all but finitely many lengths or for all but finitely many strings $x \in L$, have been studied extensively (see [2, 18, 10], for example). It is known, for instance, that there is a language L that requires more than $2^{|x|}$ many steps deterministically on almost every string $x \in L$ but that can be accepted within time $(2 + \epsilon)^n$ for any $\epsilon > 0$. Our methods do not give such results for nondeterministic acceptance time complexity; so we leave it open whether there is a language $L \in \text{NTIME}((2 + \epsilon)^n)$ that requires, even on nondeterministic machines, more than $2^{|x|}$ steps on inputs $x \in L$ of all but finitely many lengths or on all but finitely many $x \in L$.

5. A purely technical question arising from Theorem 4 is whether we can allow f to range over all one-one functions rather than just over strictly increasing recursively bounded ones. A plausible proof strategy is to design M' in the proof of Theorem 4 so that, in the case $k < T(|x|)$, it pads *or unpads* $ex0^k$ to $ex0^{k'}$ for some nondeterministically chosen $k' \neq k$. Under this strategy, however, even Claim 1 seems to elude proof.

6. What is the relationship between deterministic time complexity and number of worktapes?

7. What is the relationship between time complexity and worktape alphabet size? (Compare [25] or [27] on the relationship between space complexity and worktape alphabet size.)

8. Is there any language in $\text{NTIME}(T_2)$ that requires more time than the language $L_{T_2}(U)$ in the proof of Theorem 4?

9. In the conclusion of Lemma 5, can we exhibit a single language that must definitely belong to $\cap\{\text{NTIME}(T_2) | T_2 \in \mathcal{T}_2\} - \cup\{\text{NTIME}(T_1) | T_1 \in \mathcal{T}_1\}$? (Compare Remark (i) following the proof of Corollary 4.2.)

Appendix. Downward Diagonalization

In this Appendix we prove Theorems 1, 2, and 2′. We proceed less formally than above,

not explicitly stating the conditions our program codes must satisfy.

THEOREM 1. *If T_2 is a running time, then each of the following set differences contains a language over* $\{0, 1\}$:

(i) $DTIME(T_2) - \cup\{DTIME(T_1) | T_2 \notin O(T_1 \log T_1)\}$,

(ii) $DTIME(T_2) - \cup\{NTIME(T_1) | \log T_2 \notin O(T_1)\}$,

(iii) $NTIME(T_2) - \cup\{DTIME(T_1) | T_2 \notin O(T_1)\}$.

PROOF. Let T_2 be a running time. For (i) and (ii), we use the construction of [11, 12]. Let $L^D_{p.c.} \subset \{0, 1\}^*$ be the set of program codes for deterministic 2-tape TM acceptors having input alphabet $\{0, 1\}$. First we design a deterministic TM acceptor U with

$$L(U) = \{ex | e \in L^D_{p.c.}, ex \in L(M_e)\},$$
$$\text{Time}_U(ex) \leq c_e \cdot \text{Time}_{M_e}(ex) \quad \text{for} \quad e \in L^D_{p.c.}, \ ex \in L(M_e),$$

where c_e depends only on e. Then we design another deterministic TM acceptor M to accept the complement of $L_{T_2}(U)$ in time T_2. (This uses the closure of $DTIME(T_2)$ under complements. Because it is not known whether $NTIME(T_2)$ is closed under complements, we cannot reason analogously with nondeterministic TM acceptors.)

(i) Let T_1 satisfy $T_2 \not\in O(T_1 \log T_1)$, and suppose $L(M) \in DTIME(T_1)$. According to [12], there is some $e \in L^D_{p.c.}$ such that M_e accepts $L(M)$ within time $c \cdot T_1 \log_2 T_1$ for some constant c. Since $T_2 \not\in O(T_1 \log T_1)$, we can take $x \in \{0, 1\}^*$ so that $c_e \cdot c \cdot T_1(|ex|) \log_2 T_1(|ex|) \leq T_2(|ex|)$. Then

$$\begin{aligned} ex \in L(M) &\Rightarrow ex \in L(M_e) \\ &\Rightarrow c_e \cdot \text{Time}_{M_e}(ex) \leq c_e \cdot c \cdot T_1(|ex|) \log_2 T_1(|ex|) \leq T_2(|ex|) \\ &\Rightarrow ex \in L_{T_2}(U) \\ &\Rightarrow ex \notin L(M) \end{aligned}$$

and also

$$ex \notin L(M) \Rightarrow ex \in L_{T_2}(U) \subset L(U) \Rightarrow ex \in L(M_e) = L(M).$$

This contradiction establishes $L(M) \notin DTIME(T_1)$. Therefore $L(M) \in DTIME(T_2) - \cup\{DTIME(T_1) | T_2 \notin O(T_1 \log T_1)\}$.

(ii) Let T_1 satisfy $\log T_2 \notin O(T_1)$, and suppose $L(M) \in NTIME(T_1)$. By straightforward simulation, there is some $e \in L^D_{p.c.}$ such that M_e accepts $L(M)$ within time c^{T_1} for some constant c. Since $\log T_2 \notin O(T_1)$, we can take $x \in \{0, 1\}^*$ so that $c_e \cdot c^{T_1(|ex|)} \leq T_2(|ex|)$. Then

$$\begin{aligned} ex \in L(M) &\Rightarrow ex \in L(M_e) \\ &\Rightarrow c_e \cdot \text{Time}_{M_e}(ex) \leq c_e \cdot c^{T_1(|ex|)} \leq T_2(|ex|) \\ &\Rightarrow ex \in L_{T_2}(U) \\ &\Rightarrow ex \notin L(M) \end{aligned}$$

and also

$$ex \notin L(M) \Rightarrow ex \in L_{T_2}(U) \subset L(U) \Rightarrow ex \in L(M_e) = L(M).$$

This contradiction establishes $L(M) \notin NTIME(T_1)$. Therefore $L(M) \in DTIME(T_2) - \cup\{NTIME(T_1) | \log T_2 \notin O(T_1)\}$.

For (iii), we make use of the simulation technique of [4] (Lemma 3 above). We assume familiarity with the proof sketch above of Lemma 3. Recall that the simulation involves guessing a sequence of displays and actions and then checking it (deterministically) for one of three outcomes: not a legal computation, legal computation without acceptance, legal computation with acceptance. The TM acceptor M which we design expects an input of the form ex, with e now a program code for an *arbitrary* (multitape) deterministic TM acceptor M_e having input alphabet $\{0, 1\}$ that halts only to accept. On

such an input, M performs the (nondeterministic) Lemma 3 simulation of M_e on ex. If the guessed sequence involves exactly $\lfloor T_2(|ex|)/|e| \rfloor$ actions by M_e and the outcome is "legal computation without acceptance," then M accepts ex, and this is the only way M accepts its input. Since M_e is deterministic and halts only to accept, it follows that

$$ex \in L(M) \Leftrightarrow \text{Time}_{M_e}(ex) > \lfloor T_2(|ex|)/|e| \rfloor.$$

(Recall that, by convention, $\text{Time}_{M_e}(ex) = \infty$ if $ex \notin L(M_e)$.)

Assuming we do not choose unusually succinct program codes, M can guess and check in $T_2(|ex|)$ steps any display and action sequence involving only $\lfloor T_2(|ex|)/|e| \rfloor$ actions by M_e. Since T_2 is a running time, checking whether the number of actions is exactly $\lfloor T_2(|ex|)/|e| \rfloor$ also requires only $T_2(|ex|)$ steps. Therefore $L(M) \in \text{NTIME}(T_2)$.

Let T_1 satisfy $T_2 \notin O(T_1)$, and suppose some deterministic TM acceptor M_e accepts $L(M)$ within time T_1. Since $T_2 \notin O(T_1)$, we can take $x \in \{0, 1\}^*$ so that $\lfloor T_2(|ex|)/|e| \rfloor \geq T_1(|ex|)$. Then

$$ex \in L(M) \Rightarrow \text{Time}_{M_e}(ex) > \lfloor T_2(|ex|)/|e| \rfloor \geq T_1(|ex|) \Rightarrow ex \notin L(M_e) = L(M),$$

and also

$$ex \notin L(M) \Rightarrow \text{Time}_{M_e}(ex) \leq \lfloor T_2(|ex|)/|e| \rfloor < \infty \Rightarrow ex \in L(M_e) = L(M).$$

This contradiction establishes $L(M) \notin \text{DTIME}(T_1)$. Therefore $L(M) \in \text{NTIME}(T_2) - \cup\{\text{DTIME}(T_1) | T_2) \notin O(T_1)\}$. □

THEOREM 2. *If T_2 is a running time, then each of the following set differences contains a language over* $\{1\}$:

(*i*) $DTIME(T_2) - \cup\{DTIME(T_1) | T_1 log\ T_1 \in o(T_2)\}$,
(*ii*) $DTIME(T_2) - \cup\{NTIME(T_1) | T_1 \in o(log\ T_2)\}$,
(*iii*) $NTIME(T_2) - \cup\{DTIME(T_1) | T_1 \in o(T_2)\}$.

PROOF. Let T_2 be a running time. To adapt the proof of Theorem 1, we make use of the function $g : \{0, 1\}^* \to N$ defined in Lemma 7 so that the binary representation of the integer $g(x)$ is $1x$. We design a deterministic TM acceptor U with

$$L(U) = \{1^{g(ex)} | e \in L^{\text{D}}_{\text{p c}},\ 1^{g(ex)} \in L(M_e)\},$$
$$\text{Time}_U(1^{g(ex)}) \leq c_e \cdot \text{Time}_{M_e}(1^{g(ex)}) \quad \text{for} \quad e \in L^{\text{D}}_{\text{p c}},\ 1^{g(ex)} \in L(D_e),$$

where c_e depends only on e. Then we design another deterministic TM acceptor M to accept $\{1\}^* - L_{T_2}(U)$ in time T_2.

(i) Let T_1 satisfy $T_1 \log T_1 \in o(T_2)$, and suppose $L(M) \in \text{DTIME}(T_1)$. According to [12], there is some $e \in L^{\text{D}}_{\text{p c}}$ such that M_e accepts $L(M)$ within time $c \cdot T_1 \log_2 T_1$ for some constant c. Since $T_1 \log\ T_1 \in o(T_2)$, we can take $x \in \{0, 1\}^*$ so long that $c_e \cdot c \cdot T_1(g(ex)) \log_2 T_1(g(ex)) \leq T_2(g(ex))$. Then

$$\begin{aligned} 1^{g(ex)} \in L(M) &\Rightarrow 1^{g(ex)} \in L(M_e) \\ &\Rightarrow c_e \cdot \text{Time}_{M_e}(1^{g(ex)}) \leq c_e \cdot c \cdot T_1(g(ex)) \log_2 T_1(g(ex)) \leq T_2(g(ex)) \\ &\Rightarrow 1^{g(ex)} \in L_{T_2}(U) \\ &\Rightarrow 1^{g(ex)} \notin L(M) \end{aligned}$$

and also

$$1^{g(ex)} \notin L(M) \Rightarrow 1^{g(ex)} \in L_{T_2}(U) \subset L(U) \Rightarrow 1^{g(ex)} \in L(M_e) = L(M).$$

This contradiction establishes $L(M) \notin \text{DTIME}(T_1)$. Therefore $L(M) \in \text{DTIME}(T_2) - \cup\{\text{DTIME}(T_1) | T_1 \log\ T_1 \in o(T_2)\}$. For part (ii), similarly, $L(M) \in \text{DTIME}(T_2) - \cup\{\text{NTIME}(T_1) | T_1 \in o(\log\ T_2)\}$.

For (iii) as for part (iii) of Theorem 1, we make use of the technique of [4] to design the nondeterministic TM acceptor M. The acceptor M expects an input of the form $1^{g(ex)}$, with e now a program code for an arbitrary (multitape) deterministic TM

acceptor M_e having input alphabet {1} that halts only to accept. On such an input, M calculates e and then performs the (nondeterministic) Lemma 3 simulation of M_e on $1^{g(ex)}$. If the guessed sequence involves exactly $\lfloor T_2(g(ex))/|e|\rfloor$ actions by M_e and the outcome is "legal computation without acceptance," then M accepts $1^{g(ex)}$, and this is the only way M accepts its input. It follows that

$$1^{g(ex)} \in L(M) \Leftrightarrow \text{Time}_{M_e}(1^{g(ex)}) > \lfloor T_2(g(ex))/|e|\rfloor.$$

Since M can calculate e from $1^{g(ex)}$ in time proportional to $g(ex) \leq T_2(g(ex))$, it follows as in the proof of Theorem 1 (iii) that $L(M) \in \text{NTIME}(T_2)$.

Let T_1 satisfy $T_1 \in o(T_2)$, and suppose some deterministic TM acceptor M_e accepts $L(M)$ within time T_1. Since $T_1 \in o(T_2)$, we can take $x \in \{0, 1\}^*$ so long that $\lfloor T_2(g(ex))/|e|\rfloor \geq T_1(g(ex))$. Then

$$\begin{aligned} 1^{g(ex)} \in L(M) &\Rightarrow \text{Time}_{M_e}(1^{g(ex)}) > \lfloor T_2(g(ex))/|e|\rfloor \geq T_1(g(ex)) \\ &\Rightarrow 1^{g(ex)} \notin L(M_e) = L(M) \end{aligned}$$

and also

$$\begin{aligned} 1^{g(ex)} \notin L(M) &\Rightarrow \text{Time}_{M_e}(1^{g(ex)}) \leq \lfloor T_2(g(ex))/|e|\rfloor < \infty \\ &\Rightarrow 1^{g(ex)} \in L(M_e) = L(M). \end{aligned}$$

This contradiction establishes $L(M) \notin \text{DTIME}(T_1)$. Therefore $L(M) \in \text{NTIME}(T_2) - \cup\{\text{DTIME}(T_1) | T_1 \in o(T_2)\}$. □

THEOREM 2′. *If T_2 is a running time, then each of the following set differences contains a language over* {1}:

(i) $DTIME(T_2) - \cup\{DTIME(T_1) | T_1$ *is a running time,* $T_2 \notin O(T_1 \log T_1)\}$,
(ii) $DTIME(T_2) - \cup\{NTIME(T_1) | T_1$ *is a running time,* $\log T_2 \notin O(T_1)\}$,
(iii) $NTIME(T_2) - \cup\{DTIME(T_1) | T_1$ *is a running time,* $T_2 \notin O(T_1)\}$.

PROOF. We make use of the technique of [20] to further adapt the proof of Theorem 2. Let T_2 be a running time. Then we can design a deterministic TM acceptor M which halts on every input and which operates as follows on input 1^n:

1 Check in $\lfloor T_2(n)\rfloor$ steps whether $T_2(n) \geq 2n$. If not, then halt without accepting the input and without canceling any integer (see step 2).

2 For $i = 1, 2, 3, \ldots, \lfloor \log_2 T_2(n)\rfloor$, successively, do the following:
a. Spend $[T_2(n)/2^i]$ steps reviewing the computations by M on as many of the shorter inputs $1, 1^2, 1^3, \ldots, 1^{n-1}$ as time permits, trying to discover whether M cancels i (in step 2b) on some shorter input (The standard Recursion Theorem [23] makes it possible for M to review its own computations. Efficiency will not matter here.)

b. If i was not discovered to have been canceled on some shorter input, then spend $\lfloor T_2(n)/2^i\rfloor$ steps trying to discover whether the ith deterministic 2-tape TM acceptor (in some fixed effective enumeration of these machines) accepts 1^n Do this by performing a simulation of the ith machine in the usual manner, so that no more than $c_i\ t$ steps are required to simulate t steps, where c_i depends only on i. If a halt is discovered in the simulated computation, then differ from the outcome, cancel i, and halt

3. If this step is reached, then halt without canceling any integer and (arbitrarily) without accepting the input

By design, M accepts within time

$$T_2(n) + \sum_{i=1}^{\infty} 2 \cdot T_2(n)/2^i = 3 \cdot T_2(n);$$

so $L(M) \in \text{DTIME}(3T_2)$. Because of step 1, $x \in L(M)$ implies $T_2(|x|) \geq 2|x|$. Therefore the constant-factor speedup technique of [11] and Lemma 2 above yields $L(M) \in \text{DTIME}(T_2)$.

(i) Let T_1 be a running time that satisfies $T_2 \notin O(T_1 \log T_1)$, and suppose $L(M) \in \text{DTIME}(T_1)$. According to [12], there is some deterministic 2-tape TM acceptor M' that accepts $L(M)$ within time $c \cdot T_1 \log_2 T_1$ for some constant c. Because T_1 is a running

time, we can assume that M' halts even on inputs $x \notin L(M')$ within $c \cdot T_1(|x|)\log_2 T_1(|x|)$ steps. Suppose that M' is the kth machine in our fixed effective enumeration of deterministic 2-tape TM acceptors. For each $j < k$, let

$$f(j) = \begin{cases} 0, \text{ if } j \text{ never gets canceled;} \\ \text{the number of steps it takes to discover (by the review procedure used} \\ \quad \text{in step 2a) that } j \text{ gets canceled, if } j \text{ does get canceled.} \end{cases}$$

Since $T_2 \notin O(T_1 \log T_1)$, we can take n so that $T_2(n) \geq 2n$; every integer $j < k$ that gets canceled gets canceled on some input shorter than 1^n; $[T_2(n)/2^k] \geq f(j)$ for every $j < k$; $[T_2(n)/2^k] \geq c_k \cdot c \cdot T_1(n)\log_2 T_1(n)$. Consider the computation by M on input 1^n for this n. Since $T_2(n) \geq 2n$, M gets by step 1 Since every integer $j < k$ that gets canceled gets canceled on some input shorter than 1^n and $[T_2(n)/2^j] \geq [T_2(n)/2^k] \geq f(j)$ for each such j, M does discover the cancellation of each such j in the execution of step 2a for $i = j$. Therefore M does eventually go on to stage $i = k$. If M does not discover in step 2a for $i = k$ that k itself is canceled on some shorter input, then M finally tries to cancel k. Since M' halts on input 1^n within $c \cdot T_1(n)\log_2 T_1(n)$ steps and $[T_2(n)/2^k] \geq c_k \cdot c \cdot T_1(n)\log_2 T_1(n)$, M does cancel k in this case. In either case, therefore, k gets canceled on some input $1^{n'}$. But then $1^{n'} \in L(M) \Leftrightarrow 1^{n'} \notin L(M')$. This contradiction establishes $L(M) \notin \text{DTIME}(T_1)$. Therefore $L(M) \in \text{DTIME}(T_2) - \cup\{\text{DTIME}(T_1) | T_1$ is a running time, $T_2 \notin O(T_1 \log T_1)\}$. For part (ii), similarly, $L(M) \in \text{DTIME}(T_2) - \cup\{\text{NTIME}(T_1) | T_1$ is a running time, $\log T_2 \notin O(T_1)\}$.

For (iii), we make use of the technique of [4] to design a nondeterministic TM acceptor M which is quite similar to the deterministic TM acceptor designed above. M will halt in every computation on every input In at most one of its computations on each input, M will cancel an integer before halting; in all other computations, M will halt without canceling any integer and without accepting the input. On input 1^n, M operates as follows:

1 For $i = 1, 2, 3, \ldots, \lfloor \log_2 T_2(n) \rfloor$, successively, do the following

a Spend $\lfloor T_2(n)/2^i \rfloor$ steps deterministically reviewing all the computations by M on as many of the shorter inputs $1, 1^2, 1^3, \ldots, 1^{n-1}$ as time permits, trying to discover whether M cancels i in some computation on some shorter input

b If i was not discovered to have been canceled on some shorter input, then spend $\lfloor T_2(n)/2^i \rfloor$ steps trying to discover whether the ith deterministic (multitape) TM acceptor (in some fixed effective enumeration of these machines) accepts 1^n Do this by performing the (nondeterministic) Lemma 3 simulation of the ith deterministic multitape machine, so that no more than $c_i \cdot t$ steps are required to simulate (i.e guess and check) each computation of length t, where c_i depends only on i If an entire halting computation is simulated, then differ from the outcome, cancel i, and halt If an incomplete but legal computation is simulated and there would have been insufficient time to guess and check a longer computation, then just continue in the loop of step 1 (Because the simulated machine is deterministic, the discovery of such a maximal computation indicates that i is canceled in *none* of the computations on 1^n) Otherwise, halt without canceling any integer and without accepting the input (Because an illegal or nonmaximal computation has been simulated, the integer i might be canceled in some other computation on 1^n)

2 If this step is reached, then halt without canceling any integer and (arbitrarily) without accepting the input

Note that, since each simulated machine above is deterministic, there is a unique computation by M on output 1^n which never reaches the "otherwise" in step 1b. Call this the *main computation* by M on input 1^n. Since only a main computation can lead to cancellation or acceptance, cancellation of the integer i on the input 1^n will guarantee that M does disagree with the ith deterministic multitape machine on input 1^n.

By design, $L(M) \in \text{NTIME}(3T_2)$, and $\text{NTIME}(3T_2) \subset \text{NTIME}(T_2)$ by Lemma 2. On the other hand, let T_1 be a running time that satisfies $T_2 \notin O(T_1)$, and suppose some deterministic TM acceptor M' accepts $L(M)$ within time T_1. Because T_1 is a running time, we can assume that M' halts even on inputs $x \notin L(M')$ within $T_1(|x|)$ steps. Suppose that M' is the kth machine in our fixed effective enumeration of deterministic (multitape) TM acceptors. For each $j < k$, let

$$f(j) = \begin{cases} 0, \text{ if } j \text{ never gets canceled;} \\ \text{the number of steps it takes to discover (by the review procedure used} \\ \quad \text{in step 1a) that } j \text{ gets canceled, if } j \text{ does get canceled.} \end{cases}$$

Since $T_2 \notin O(T_1)$, we can take n so that every integer $j < k$ that gets canceled gets canceled on some input shorter than 1^n; $\lfloor T_2(n)/2^k \rfloor \geq f(j)$ for every $j < k$; $\lfloor T_2(n)/2^k \rfloor \geq c_k \cdot T_1(n)$. Consider the main computation by M on input 1^n for this n, implicitly ignoring all other computations. Since every integer $j < k$ that gets canceled gets canceled on some input shorter than 1^n and $\lfloor T_2(n)/2^j \rfloor \geq \lfloor T_2(n)/2^k \rfloor \geq f(j)$ for each such j, M does discover the cancellation of each such j in the execution of step 1a for $i = j$. Therefore M does eventually go on to stage $i = k$. If M does not discover in step 1a for $i = k$ that k itself is canceled on some shorter input, then M finally tries to cancel k. Since M' halts on input 1^n within $T_1(n)$ steps and $\lfloor T_2(n)/2^k \rfloor \geq c_k \cdot T_1(n)$, M does cancel k in this case, in its main computation. In either case, therefore, k gets canceled on some input 1^n). But then $1n' \in L(M) \Leftrightarrow 1^{n'} \notin L(M')$. This contradiction establishes $L(M) \notin \text{DTIME}(T_1)$. Therefore $L(M) \in \text{NTIME}(T_2) - \cup\{\text{DTIME}(T_1) \mid T_1 \text{ is a running time}, T_2 \notin O(T_1)\}$. □

REFERENCES

1. AANDERAA, S.O On k-tape versus $(k-1)$-tape real time computation. In *Complexity of Computation* (SIAM-AMS Proceedings, Vol 7), R M Karp, Ed , Amer Math. Soc , Providence, R I , 1974, pp 75-96.
2. BLUM, M. A machine-independent theory of the complexity of recursive functions *J. ACM 14*, 2 (April 1967), 322-336
3. BOOK, R V , AND GREIBACH, S.A. Quasi-realtime languages *Math. Syst Theory 4*, 2 (June 1970), 97-111.
4. BOOK, R V., GREIBACH, S A , AND WEGBREIT, B. Time- and tape-bounded Turing acceptors and AFLs. *J. Comptr Syst Sci 4*, 6 (Dec 1970), 606-621.
5. CONSTABLE, R.L. The operator gap *J. ACM 19*, 1 (Jan. 1972), 175-183.
6. CONSTABLE, R.L. Two types of hierarchy theorem for axiomatic complexity classes In *Computational Complexity*, R Rustin, Ed., Algorithmics Press, New York, 1973, pp 37-63
7. COOK, S A. A hierarchy for nondeterministic time complexity. *J Comptr Syst Sci. 7*, 4 (Aug. 1973), 343-353
8. FISCHER, M J , AND RABIN, M.O Super-exponential complexity of Presburger arithmetic In *Complexity of Computation* (SIAM-AMS Proceedings, Vol. 7), R M Karp, Ed., Amer Math Soc , Providence, R I., 1974, pp 27-41
9. FISCHER, P C., MEYER, A.R , AND ROSENBERG, A L Real-time simulation of multihead tape units. *J ACM 19*, 4 (Oct 1972), 590-607
10. GILL, J., AND BLUM, M On almost everywhere complex recursive functions. *J ACM 21*, 3 (July 1974), 425-435
11. HARTMANIS, J., AND STEARNS, R.E. On the computational complexity of algorithms. *Trans Amer. Math. Soc 117* (May 1965), 285-306.
12. HENNIE, F.C , AND STEARNS, R.E. Two-tape simulation of multitape Turing machines. *J ACM 13*, 4 (Oct. 1966), 533-546
13. HOPCROFT, J E., AND ULLMAN, J D. *Formal Languages and Their Relation to Automata* Addison-Wesley, Reading, Mass , 1969.
14. HUNT, H.B III The equivalence problem for regular expressions with intersection is not polynomial in tape. Tech Rep. 73-161, Dept. of Comptr Sci., Cornell U , Ithaca, N.Y , March 1973
15. IBARRA, O H A note concerning nondeterministic tape complexities. *J. ACM 19*, 4 (Oct 1972), 608-612
16. IBARRA, O.H On two-way multihead automata. *J Comptr. Syst. Sci 7*, 1 (Feb 1973), 28-36.
17. IBARRA, O H A hierarchy theorem for polynomial-space recognition *SIAM J. Compting 3*, 3 (Sept. 1974), 184-187.
18. LYNCH, N.A., MEYER, A.R., AND FISCHER, M J Relativization of the theory of computational complexity. *Trans. Amer Math. Soc 220* (June 1976), 243-287.
19. MEYER, A.R Weak monadic second order theory of successor is not elementary-recursive In *Logic Colloquium, Lecture Notes in Mathematics, No 453*, R Parikh, Ed., Springer-Verlag, Berlin, 1975, pp 132-154.
20. MEYER, A.R , AND MCCREIGHT, E.M Computationally complex and pseudo-random zero-one valued functions. In *Theory of Machines and Computations*, Z. Kohavi and A. Paz, Ed , Academic Press, New York, 1971, pp 19-42.

21. MEYER, A R., AND STOCKMEYER, L J The equivalence problem for regular expressions with squaring requires exponential space Proc 13th Annual Symp on Switching and Automata Theory, College Park, Md , 1972, pp 125–129.
22. PAUL, W J On time hierarchies Proc Ninth Annual ACM Symp on Theory of Comptng., Boulder, Colo , 1977, pp. 218–222
23 ROGERS, H JR *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, New York, 1967.
24. RUBY, S , AND FISCHER, P C Translational methods and computational complexity Conf. Rec. IEEE Symp. on Switching Circuit Theory and Logical Design. Ann Arbor, Mich , 1965, pp. 173–178.
25 SEIFERAS, J I Nondeterministic time and space complexity classes. Tech. Rep. 137, Proj. MAC, M.I T., Cambridge, Mass , Sept 1974
26. SEIFERAS, J.I. Techniques for separating space complexity classes *J Comptr. Syst. Sci. 14*, 1 (Feb. 1977), 73–99
27 SEIFERAS, J I. Relating refined space complexity classes. *J Comptr Syst. Sci. 14*, 1 (Feb. 1977), 100–129.
28 STOCKMEYER, L.J The complexity of decision problems in automata theory and logic. Tech. Rep. 133, Proj MAC, M I T., Cambridge, Mass., June 1974.
29 STOCKMEYER, L J , AND MEYER, A R Word problems requiring exponential time: Preliminary report. Proc Fifth Annual ACM Symp. on Theory of Comptng., Austin, Tex., 1973, pp. 1–9.
30. YAMADA, H. Real-time computation and recursive functions not real-time computable. *IRE Trans. Electron. Comptrs. EC-11*, 6 (Dec 1962), 753–760, Corrections *IEEE Trans. Electron. Comptrs. EC-12*, 4 (Aug 1963), 400.
31. YOUNG, P Easy constructions in complexity theory· Gap and speed-up theorems. *Proc. Amer. Math. Soc 37*, 2 (Feb. 1973), 555–563.

RECEIVED SEPTEMBER 1976, REVISED MARCH 1977