MIT/LCS/TR-137

NONDETERMINISTIC TIME AND SPACE COMPLEXITY CLASSES

Joel Irvin Seiferas

September 1974

NONDETERMINISTIC TIME AND SPACE COMPLEXITY CLASSES

by

Joel Irvin Seiferas

September 1974

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

This empty page was substituted for a blank page in the original document.

NONDETERMINISTIC TIME AND SPACE COMPLEXITY CLASSES

by

Joel Irvin Seiferas

Submitted to the Department of Electrical Engineering on August 30, 1974, in partial fulfillment of the requirements for the Degree of Doctor of Philosophy.

ABSTRACT

The marginal utility of the Turing machine computational resources running time and storage space are studied. A technique is developed which, unlike diagonalization, applies equally well to nondeterministic and deterministic automata. For f, g time or space bounding functions with f(n+1) small compared to g(n), it is shown that, in terms of word length n, there are languages which are accepted by Turing machines operating within time or space g(n) but which are accepted by no Turing machine operating within time or space f(n). The proof involves use of the recursion theorem together with "padding" or "translational" techniques of formal language theory.

Relations between worktape alphabet size, number of worktape heads, number of input heads, and Turing machine storage space are established. Within every common subexponential space bound, it is shown that enlarging the worktape alphabet always increases computing power. A hierarchy of two-way multihead finite automata is obtained even in the nondeterministic case.

Results that are only slightly weaker are obtained for Turing machines that accept only languages over a one-letter alphabet.

THESIS SUPERVISOR: Albert R. Meyer
TITLE: Associate Professor of Electrical Engineering

ACKNOWLEDGEMENTS

I would like first to thank Albert Meyer for his confidence, inspiration, and guidance during the last two and a half years and especially in this research. Many of his ideas, only a few of which he remembers having, have found their way into the thesis.

I am grateful to Michael Fischer, who also collaborated along with Prof. Meyer in some of this work, for his insightful criticism and probing technical questions.

Several helpful suggestions were made by Ronald Book and Celia Wrathall, who carefully read an earlier version of this work.

I thank the National Science Foundation (research grant GJ-34671), Project MAC, and M. I. T., which I now leave after nine happy years as a student.

On the home front, I am eternally indebted to my wife Diane, who has always placed my research environment first, even waiting until the day after the defense of this thesis to deliver our son. I also thank our parents for helping me to reach this occasion.

This thesis is dedicated to my wife and to the memory of her late father.

TABLE OF CONTENTS

CHAPTER ONE

INTRODUCTION

The ultimate purpose of studies in computational complexity is to establish the complexity, in terms of computational resources such as time and storage space, that is inherent in particular computational tasks. The existence of computational tasks with various inherent complexities is the subject of this thesis. The mere existence of such computational tasks does not, *a priori*, have a bearing on the complexity of computational tasks of practical interest; but in fact techniques such as those of Meyer and Stockmeyer [MS72], Meyer [Mey73], Stockmeyer and Meyer [SM73], Hunt [Hun73], M. Fischer and Rabin [FR74], and Stockmeyer [St74] sometimes show that a particular task of interest lies at some level if *anything* does, and results like ours can then serve.

The computational tasks that we consider are "language acceptance" tasks. A language is a set of strings of symbols from some finite alphabet. A language is accepted by a computer M if M enters an accepting state when and only when applied to a member of the language. We denote by L(M) the language accepted by M.

Turing machines are the computer model we use. Customarily, we measure time and space usage by a Turing acceptor in terms of input *length* only. (We denote by $|x|$ the length of the string x.) The Turing machine M accepts L(M) within time T(n) or space S(n) if each string $x \in L(M)$ is accepted in some computation involving no more than $T(|x|)$ steps or $S(|x|)$ worktape squares, respectively. (More precise definitions appear in Chapters Two and Three.) We denote by NTIME(T) and

DTIME(T) the classes of languages accepted within time T by nondeterministic and deterministic Turing machines, respectively; and we denote by NSPACE(S) and DSPACE(S) the classes of languages accepted within space S by nondeterministic and deterministic Turing machines, respectively.

We would like, for example, to define the inherent deterministic time complexity of accepting a language L to be the "least" time bound T, in some sense, such that $L \in \mathrm{DTIME}(T)$. The existence of languages which have no best acceptor (languages with "speed-up" [Blm67]), however, makes such an approach impossible. Instead, we content ourselves to specify <u>pairs</u> of time bounds T_1, T_2 for which $L \in \mathrm{DTIME}(T_2) - \mathrm{DTIME}(T_1)$. In effect, then, we are led to a study of the containment lattice of the complexity classes. For example we address ourselves to the problems of finding what we call "containment" and "separation" conditions on time bounds T_1, T_2 which imply that $\mathrm{DTIME}(T_1) \subset \mathrm{DTIME}(T_2)$ and that $\mathrm{DTIME}(T_2) - \mathrm{DTIME}(T_1) \neq \emptyset$ (i. e., $\mathrm{DTIME}(T_2) \not\subset \mathrm{DTIME}(T_1)$), respectively.

Rather strong separation results for the DTIME and DSPACE complexity classes are well known ([HaS65], [HeS66], [SHL65], [HU69a], [HU69b], [Con73], Appendix I of this thesis), but the diagonalization technique that most of them rely on does not give very strong separation results for the NTIME and NSPACE classes. A result of Cook [Ck73] first separated $\mathrm{NTIME}(n^r)$ from $\mathrm{NTIME}(n^s)$ for $r \neq s$. Our contribution is a simplified and greatly generalized version of Cook's technique that applies to nondeterministic and deterministic time and space complexity.

The major value of our technique is for nondeterministic computation, and the results are most dramatic at exponential and subexponential complexity levels. Although no real computer actually operates nondeterministically, the concept does arise naturally in connection with formal language theory ([HU69b], [AU72], [AU73]), proof theory, and the description and complexity of other processes involving arbitrary searches [F167]. (A deterministic description would force one to specify the essentially irrelevant details of some arbitrary search algorithm.) The Cook-Karp question of whether P = NP, where

$$P = \bigcup \{DTIME(T) \mid T \text{ is a polynomial time bound}\},$$

$$NP = \bigcup \{NTIME(T) \mid T \text{ is a polynomial time bound}\},$$

represents a mathematical formulation of the problem of characterizing the complexity of a large class of combinatorial optimization problems involving such unstructured searches ([Ck71], [Krp72]).

We first generalize Cook's technique for multitape Turing machine time complexity in Chapter Two. For well-behaved T_2, we show that $NTIME(T_2) - NTIME(T_1) \neq \emptyset$ whenever $T_1(n+1) \in o(T_2(n))$,[†] for example. Surprisingly, this yields some specific separation results for NTIME which are stronger than the corresponding known separation results for DTIME. In contrast, the earlier results based on diagonalization were always stronger for DTIME than for NTIME. Separation results with respect to languages over a one-letter alphabet are obtained that are only slightly weaker than the general ones.

In Chapter Three we refine the NSPACE and DSPACE complexity classes

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

by carefully bounding worktape alphabet size and number of worktape heads. We reformulate the known separation results for these classes and apply our technique to get new ones of the kind given above for NTIME. By relating the various resources (space, worktape alphabet size, number of worktape heads), we obtain separation results that focus on the marginal utility of each resource. As a corollary we get a hierarchy theorem for two-way multihead finite automata. As in Chapter Two, only slightly weaker separation results are obtained with respect to languages over a one-letter alphabet.

A preliminary version of this thesis has been reported jointly with M. Fischer and A. Meyer [SFM73]. In particular, Corollaries 14, 16 of Chapter Two were obtained and reported jointly. Chapter Three contains much new material, but Corollaries 19(iii), 20(iii), 22 were also presented in the preliminary version. Problems 2, 3, 4 of [SFM73] are settled affirmatively by the current results of Chapters Two and Three.

For convenient reference we now list the results which are the main contributions of this thesis. All of the relevant definitions and notation are provided in Chapters Two and Three.

Chapter Two

Assume T_2 is a running time.

The most general result we prove is Theorem 13.

Theorem 13.

$$\mathrm{NTIME}(T_2) - \bigcup \{\mathrm{NTIME}(T_1) \mid \text{there is some recursively bounded but strictly increasing function } f: N \to N \text{ for which } T_1(f(n+1)) \in o(T_2(f(n)))\}$$

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

In practice it is the simpler but only slightly less general Corollary 14 that we emphasize. (For nondecreasing time bounds the result is no weaker than Theorem 13.)

Corollary 14. $\mathrm{NTIME}(T_2) - \bigcup \{\mathrm{NTIME}(T_1) \mid T_1(n+1) \in o(T_2(n))\}$ contains a language over $\{0,1\}$.

Corollary 14 gives results that diagonalization does not give precisely when $\log T_2(n+1) \in o(T_2(n))$. Corollary 15 is a refinement that gives new results even when $\log T_2(n+1) \in O(T_2(n))$.

Corollary 15.
$\mathrm{NTIME}(T_2) - \bigcup \{\mathrm{NTIME}(T_1) \mid T_1(n+1) \in O(T_2(n)),\ T_1(n) \in o(T_2(n))\}$
contains a language over $\{0,1\}$.

For deterministic time complexity, we prove a version of Corollary 14 that has an additional hypothesis.

Theorem 16. Suppose that there is some fixed k such that for each deterministic TM acceptor M there is a deterministic k-tape TM acceptor M' and a constant c such that $L(M') = L(M)$ and $\mathrm{Time}_{M'}(x) \leq c \cdot f(\mathrm{Time}_M(x))$. Then $\mathrm{DTIME}(T_2) - \bigcup \{\mathrm{DTIME}(T_1) \mid f(T_1(n+1)) \in o(T_2(n))\} \neq \emptyset$.

When we restrict our attention to languages over a one-letter alphabet, we still get Theorem 17.

Theorem 17. If f is real-time countable, then

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

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

Chapter Three

Containment:

First we relate worktape alphabet size to storage space.

Proposition 3. $NSPACE(S,m,\ell) \subset NSPACE(S',m',\ell)$ if $S(n)/S'(n) \leq \delta \leq \log_m m'$ for some rational δ. Similarly for DSPACE.

The next two results relate the number of worktape heads to (the logarithm of) storage space.

Proposition 4. $NSPACE(S, m, \ell+k) \subset NSPACE(S+(k+1+\epsilon)\log_m S, m, \ell)$ for every $\epsilon > 0$. Similarly for DSPACE.

Proposition 5. $NSPACE(S + k\cdot\log_m S, m, \ell) \subset NSPACE(S, m, \ell+k+3)$. Similarly for DSPACE.

Later on we relate multihead finite automata to logarithmic storage space.

Lemma 21. $NHEADS(k) \subset NSPACE(\log_2 n, 2^k, 1) \subset NHEADS(k+4)$. Similarly for DHEADS, DSPACE.

Separation:

Theorems 18 and 26 are somewhat analogous to Corollary 14 and Theorem 17 of Chapter Two, especially in proof.

Theorem 18. $NSPACE(S_2,m,\ell+3) - \bigcup \{NSPACE(S_1,m,\ell+2) \mid 1 \in o(S_2(n)-S_1(n+1))\}$ contains a language over $\{0,1\}$ if S_2 is fully constructable by an (m,ℓ)-machine. Similarly for DSPACE.

Theorem 26.

$NSPACE(S_2,m,\ell+6) - \bigcup \{NSPACE(S_1,m,\ell) \mid S_2(n)-S_1(n+f(n)) \geq 4\cdot\log_m n\}$ contains a language over just $\{1\}$

if S_2 is fully constructable by an $(m,\ell+2)$-machine,

$\log n \in o(S_2(n))$,

$f(n) \in O(n) - O(1)$ is nondecreasing and linear space honest.

Similarly for DSPACE.

Unlike Theorem 26, Theorem 27 applies even for logarithmic space bounds.

Theorem 27. $\mathrm{NSPACE}(S_2,2,\ell+1) - \bigcup \{\mathrm{NSPACE}(S_1,2,\ell) \mid 1 \in o(S_2(n)-S_1(2n))\}$ contains a language over just $\{1\}$

if S_2 is fully constructable by a $(2,\ell)$-machine,

$1 \in o(S_2(n) - \log_2 n)$,

$\ell \geq 3$.

CHAPTER TWO

TIME SEPARATION THEOREMS FOR NONDETERMINISTIC MULTITAPE TURING MACHINES

In this chapter we refer to what is usually called a nondeterministic multitape Turing machine [HU69b] 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 call the input tape. A TM can act as an acceptor by halting in some specified 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 [HU69b].

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 accepting computation by M on input x. M accepts the language $L(M) = \{x \mid M \text{ accepts string } x\}$. For $x \in L(M)$, $\mathrm{Time}_M(x)$ is the number of steps in the shortest accepting computation by M on x; for $x \notin L(M)$,

†We use the Kleene star * more generally as well, along with other regular expression notation for regular sets. For $A, B \subset \Sigma^*$,

$A + B = A \cup B = \{x \mid x \in A \text{ or } x \in B\}$,

$A \cdot B = AB = \{xy \mid x \in A, y \in B\}$,

$A^* = \{\lambda\} + A + A \cdot A + A \cdot A \cdot A + \cdots = \{\lambda\} + A + A^2 + A^3 + \cdots$,

where λ is the null or empty string. When it causes no ambiguity, we sometimes omit set brackets in regular expressions.

$Time_M(x) = \infty$.

Definition. A time bound is a function $T:N \to N$ with $T(n) \geq n$ for every n. For T a time bound, the T-cutoff of the TM M is the language $L_T(M) = \{x \mid Time_M(x) \leq T(|x|)\}$, which is always a subset of $L(M)$. A language L is in NTIME(T) iff $L = L(M) = L_T(M)$ 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 [BGW70], 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 $Time_M(x) = T(|x|) \geq |x|$, then T is a running time, and M is a clock for T.

Diagonalization is the best known technique for obtaining separation or "hierarchy" results among the NTIME and DTIME complexity classes. A summary of the best separation results that have been proved by diagonalization alone is given by the following pair of theorems. (See Appendix I, [HaS65], [HeS66], [Con73].)

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

$$\mathrm{DTIME}(T_2) - \bigcup \{\mathrm{DTIME}(T_1) \mid T_2 \notin O(T_1 \log T_1)\},$$

$$\mathrm{NTIME}(T_2) - \bigcup \{\mathrm{DTIME}(T_1) \mid T_2 \notin O(T_1)\},$$

$$\mathrm{DTIME}(T_2) - \bigcup \{\mathrm{NTIME}(T_1) \mid \log T_2 \notin O(T_1)\}.^{\dagger}$$

<u>Theorem 2</u>. If T_2 is a running time, then each of the following set differences contains a language over $\{1\}$:

$$\mathrm{DTIME}(T_2) - \bigcup \{\mathrm{DTIME}(T_1) \mid T_1 \log T_1 \in o(T_2)\},$$

$$\mathrm{NTIME}(T_2) - \bigcup \{\mathrm{DTIME}(T_1) \mid T_1 \in o(T_2)\},$$

$$\mathrm{DTIME}(T_2) - \bigcup \{\mathrm{NTIME}(T_1) \mid T_1 \in o(\log T_2)\}.^{\dagger}$$

<u>Remark</u>. By restricting the unions of Theorem 2 to range only over running times T_1, we can use the diagonalization technique of [MM71] to show that each of the following set differences contains a language over $\{1\}$ if T_2 is a running time:

$$\mathrm{DTIME}(T_2) - \bigcup \{\mathrm{DTIME}(T_1) \mid T_1 \text{ is a running time, } T_2 \notin O(T_1 \log T_1)\},$$

$$\mathrm{NTIME}(T_2) - \bigcup \{\mathrm{DTIME}(T_1) \mid T_1 \text{ is a running time, } T_2 \notin O(T_1)\},$$

$$\mathrm{DTIME}(T_2) - \bigcup \{\mathrm{NTIME}(T_1) \mid T_1 \text{ is a running time, } \log T_2 \notin O(T_1)\}.$$

Note the relatively poor results obtained in diagonalizing over NTIME. Not even the gross separation result $\mathrm{NTIME}(n^2) \subsetneq \mathrm{NTIME}(2^{n^2})$, for example, follows directly from Theorem 1; yet, $\mathrm{DTIME}(n^2) \subsetneq \mathrm{DTIME}(n^2(\log n)^2)$ does follow. Recently, however, Cook [Ck73] proved the following result by a new technique.

<u>Theorem 3</u>. $\mathrm{NTIME}(n^r) \subsetneq \mathrm{NTIME}(n^s)$ whenever $1 \le r < s$.

In this chapter we pursue Cook's technical breakthrough, simplifying his

† When the precise specification of a time bound is not relevant in some context, 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 make no difference. (See also Lemma 7 below.)

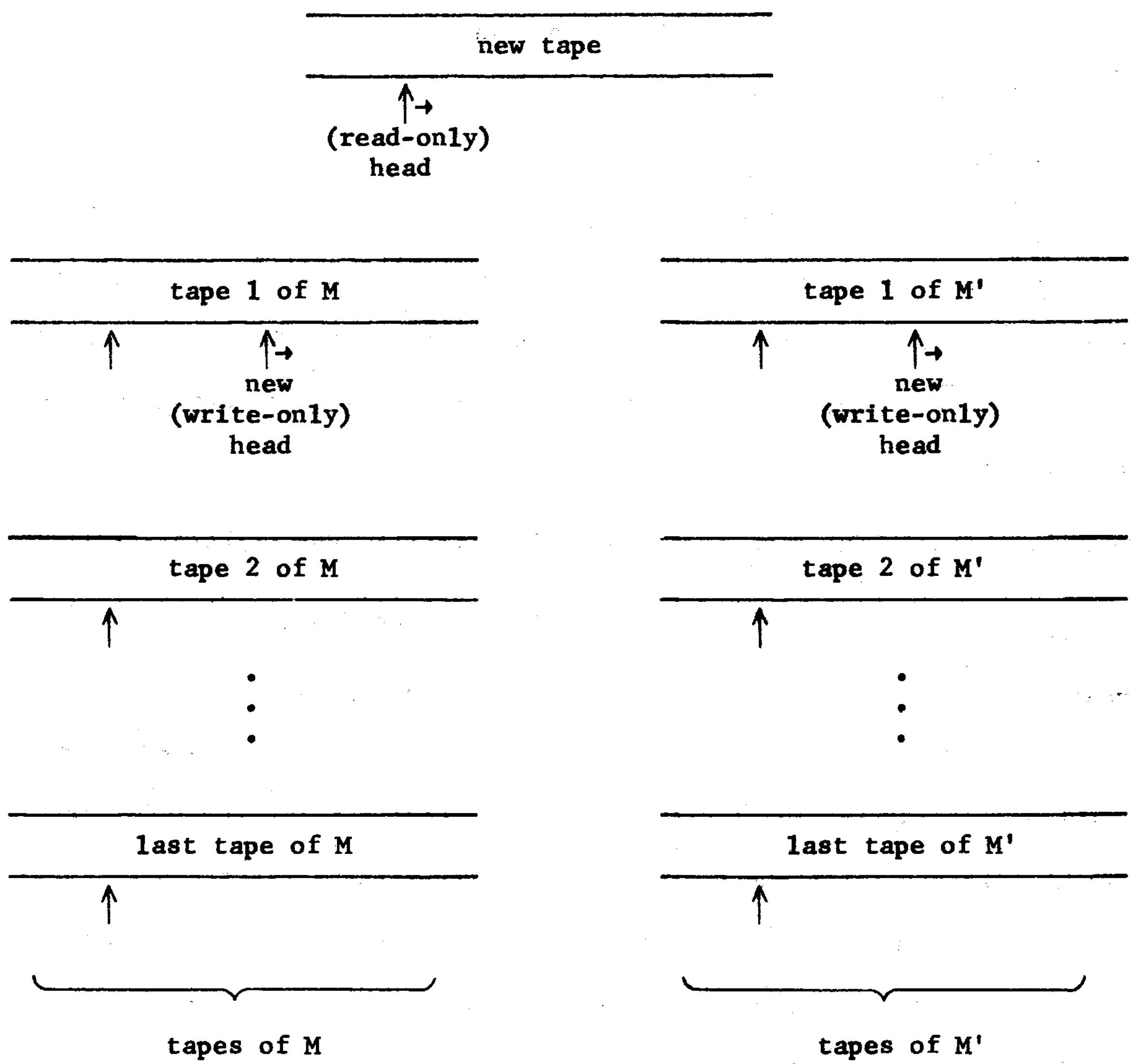
proof and generalizing the result. The main generalization is Theorem 13 below. We turn now to some lemmas that will be useful in the proof of that theorem and its corollaries.

P. Fischer, Meyer, and Rosenberg [FMR72] 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. This allows a TM to carry out two computations at the same time, leading to proofs of the following lemmas.

Lemma 4. $L(M) \cup L(M')$ can be accepted in time $\min\{\mathrm{Time}_M(x), \mathrm{Time}_{M'}(x)\}$.

Lemma 5. $L(M) \cap L(M')$ can be accepted in time $\max\{\mathrm{Time}_M(x), \mathrm{Time}_{M'}(x)\}$.

Proof sketches. 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 two 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.



To accept $L(M) \cup L(M')$, the composite machine enters its accepting state when the computation by either M or M' does. To accept $L(M) \cap L(M')$, the composite machine enters its accepting state when computations by both M and M' do. Thus we have described multihead multitape Turing machines that accept $L(M) \cup L(M')$ and $L(M) \cap L(M')$ in the desired times. By the result of [FMR72], these can be simulated without time loss by multitape Turing machines with only one head per tape. □

The same technique leads to a proof of the next lemma, in which a single TM carries out computations by M and a clock for T simultaneously, accepting if M accepts before time T runs out.

__Lemma 6__. If M is a TM acceptor and T is a running time, then $L_T(M) \in \mathrm{NTIME}(T)$.

The next lemma indicates that the NTIME complexity classes depend only on growth rates. It also shows that we need at least the condition $T_2 \notin O(T_1)$ to be able to prove $\mathrm{NTIME}(T_2) - \mathrm{NTIME}(T_1) \neq \emptyset$. It follows by Theorem 1 that, if (contrary to most people's intuition) DTIME(T) = NTIME(T) for all T, then $\mathrm{NTIME}(T_2) - \mathrm{NTIME}(T_1) = \mathrm{DTIME}(T_2) - \mathrm{DTIME}(T_1)$ is nonempty __precisely__ __when__ the running time T_2 is a member of the complement of $O(T_1)$.

__Lemma 7__. If $T_1 \in O(T_2)$, then $\mathrm{NTIME}(T_1) \subset \mathrm{NTIME}(T_2)$.

__Proof sketch__. For $T_2(n) \geq (1+\epsilon)n$ for some $\epsilon > 0$, this is just the linear time speedup theorem of Hartmanis and Stearns [HaS65]. The idea is to increase the size of each TM's worktape alphabet so that several steps can be performed in one big step.

That the lemma holds for arbitrary $T_2(n) \geq n$ has been observed by Book and Greibach [BG70]. The key idea is to use nondeterminism to guess the entire input string before it is read. □

The following lemma, due to Book, Greibach, and Wegbreit [BGW70], 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 TMs.

<u>Lemma 8</u>. For each TM M there is a 2-tape TM M' and a constant c such that $L(M') = L(M)$ and $\mathrm{Time}_{M'}(x) \leq c \cdot \mathrm{Time}_M(x)$ for every $x \in L(M)$.[†]

<u>Proof sketch</u>. 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 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 [BGW70]. □

<u>Lemma 9</u>. For no recursive time bound T does NTIME(T) contain all the recursive languages over $\{1\}$.

<u>Proof</u>. Each recursive time bound lies below some running time T_1, and Theorem 2 gives a recursive language over $\{1\}$ in $\mathrm{DTIME}(2^{2^{T_1}}) - \mathrm{NTIME}(T_1)$. □

[†]An idea of [BG70] allows us to take c = 1 if we settle for a 3-tape TM M'. (See Lemma 7.) Aanderaa [Aan74] 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" ($\mathrm{Time}_M(x) = |x|$).)

Like Cook's proof of Theorem 3, our proof of Theorem 13 makes crucial use of a trick called "padding." Acceptance time is measured as a function of <u>input length</u>; 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

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

for some $p:N \to N$ with $p(n) > n$.

<u>Lemma 10</u>. If $p(n) > n$ is a running time, then

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

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

<u>Proof</u>. ($\Rightarrow$) 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 $p(|x|)$ by using an extra head on tape 1. A third head can be used for the computation by M_1 on input $x10^k$. Clearly, then, M_2 accepts L within time $p(n) + T(p(n)) \leq 2 \cdot T(p(n))$. But $NTIME(2 \cdot T(p(n))) \subset NTIME(T(p(n)))$, by Lemma 7.

($\Leftarrow$) Suppose M_2 accepts L within time $T(p(n))$. Design M_1 to check that its input string is of the form $x10^k$, where $|x10^k| = p(|x|)$, and then to behave like M_2 on input x. Clearly M_1 accepts p(L) within time proportional to T(length of input to M_1), as required. $\square$

The following lemma shows how padding may be used to derive separation results. Ruby and P. Fischer [RF65] first used essentially this technique in connection with the deterministic time complexity of sequence generation, and Ibarra [Ib72] used it more explicitly in connection with the nondeterministic space complexity of language acceptance. (See Chapter Three of this thesis.) Ibarra has used similar techniques in other contexts as well ([Ib73a], [Ib73b], [IS73]).

Lemma 11. Let sets $\mathcal{S}_1$, $\mathcal{S}_2$ of time bounds be given. Say $p_1(n) > n, \ldots, p_\ell(n) > n$ are running times with $T_1 \circ p_{i+1} \in O(T_2 \circ p_i)$ whenever $1 \le i < \ell$, $T_1 \in \mathcal{S}_1$, $T_2 \in \mathcal{S}_2$.
If $L \in \bigcap \{\mathrm{NTIME}(T_2 \circ p_\ell) \mid T_2 \in \mathcal{S}_2\} - \bigcup \{\mathrm{NTIME}(T_1 \circ p_1) \mid T_1 \in \mathcal{S}_1\}$, then $p_i(L) \in \bigcap \{\mathrm{NTIME}(T_2) \mid T_2 \in \mathcal{S}_2\} - \bigcup \{\mathrm{NTIME}(T_1) \mid T_1 \in \mathcal{S}_1\}$ for some i.

Proof. For $1 \le i \le \ell$, let

$$C(i,1) = \bigcup \{\mathrm{NTIME}(T_1 \circ p_i) \mid T_1 \in \mathcal{S}_1\},$$

$$C(i,2) = \bigcap \{\mathrm{NTIME}(T_2 \circ p_i) \mid T_2 \in \mathcal{S}_2\}.$$

Suppose $L \in C(\ell,2) - C(1,1)$. By Lemma 7, $\mathrm{NTIME}(T_1 \circ p_{i+1}) \subset \mathrm{NTIME}(T_2 \circ p_i)$ whenever $1 \le i < \ell$, $T_1 \in \mathcal{S}_1$, $T_2 \in \mathcal{S}_2$; so, for $1 \le i < \ell$,

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

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 $L \in C(\ell,2)$ that $L \in C(1,1)$, a contradiction. For some i, therefore, we must have

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

$$= \bigcap \{\mathrm{NTIME}(T_2 \circ p_i) \mid T_2 \in \mathcal{S}_2\} - \bigcup \{\mathrm{NTIME}(T_1 \circ p_i) \mid T_1 \in \mathcal{S}_1\}.$$

By Lemma 10,

$$p_i(L) \in \bigcap \{\mathrm{NTIME}(T_2) \mid T_2 \in \mathcal{S}_2\} - \bigcup \{\mathrm{NTIME}(T_1) \mid T_1 \in \mathcal{S}_1\}$$

for that same i. □

Remarks. (i) We do not know how to exhibit the particular language that must be in $\bigcap \{NTIME(T_2) \mid T_2 \in \mathcal{S}_2\} - \bigcup \{NTIME(T_1) \mid T_1 \in \mathcal{S}_1\}$.

(ii) The same technique can be applied to DTIME, and it allows us to strengthen the results of diagonalization a bit. For example we can use it to show $DTIME(n^2) \subsetneq DTIME(n^2(\log n)^{1/200})$. (Take $p_i(n) = n(\log n)^{i/400}$ for $0 \le i \le 399$.)

Another key idea in Cook's proof and our extensions of it involves a universal TM simulator. So that we may speak with precision about universal simulation, let us now choose an appropriate program coding for TMs. With each TM having at least two tapes and input alphabet $\{0,1\}$, we associate a distinct program code from $\{0,1\}^*$; and we do this in agreement with the easily-satisfied conditions listed below. We use the notation $L^k_{p.c.}$ for the set of program codes for k-tape TMs and $L_{p.c.}$ for the set of all program codes. We denote by M_e the TM with program code e.

Condition 1. No program code is a prefix of another, and $L^k_{p.c.}$ is in DTIME(n) for each k.

Condition 2. For each fixed k, there is a TM acceptor U_0 (a "universal simulator") with

$$L(U_0) = \{ex \mid e \in L^k_{p.c.},\ x \in L(M_e)\},$$

$$Time_{U_0}(ex) \le c_e \cdot Time_{M_e}(x) \text{ if } e \in L^k_{p.c.},$$

where c_e depends only on e.

Condition 3. There is a recursive function $f: L_{p.c.} \to L_{p.c.}$ such that

$f: L^k_{p.c.} \to L^k_{p.c.}$ for each k and such that $M_{f(e)}$ spends its first $|e|$ steps putting e at its head on tape 2 (by writing backwards) 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 [Rog 67].) Most common instruction-by-instruction or state-by-state codings of TM programs can be tailored to satisfy these conditions. An example of a satisfactory program coding is described in Appendix II.

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.

Lemma 12. For each k-tape TM acceptor M with $L(M) \subset \{0,1\}^*$, there is a k-tape TM acceptor M_{e_0} with

$$L(M_{e_0}) = \{x \mid e_0 x \in L(M)\},$$

$$\exists c(\mathrm{Time}_{M_{e_0}}(x) \le c + \mathrm{Time}_M(e_0 x)).$$

Proof. Let f be as in Condition 3. Take M_{e_1} to be a k-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. Spend $|e_1|$ steps putting e_1 at the head on tape 2.

2. Convert e_1 to $f(e_1) = e_0$.

3. Convert x to e_0x.

4. Behave like M on e_0x.

Thus,

$$x \in L(M_{e_0}) \Leftrightarrow e_0x \in L(M),$$

$$\mathrm{Time}_{M_{e_0}}(x) \le c + \mathrm{Time}_M(e_0x),$$

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. $\square$

Theorem 13. If T_2 is a running time, then

$$\mathrm{NTIME}(T_2) - \bigcup \{\mathrm{NTIME}(T_1) \mid \text{there is some recursively bounded but strictly increasing function } f{:}N \to N \text{ for which } T_1(f(n+1)) \in o(T_2(f(n)))\}$$

contains a language over $\{0,1\}$.†

Proof. Let T_2 be a running time, and let U_0 be the universal simulator of Condition 2 for $k = 2$. By Lemma 6, $L_{T_2}(U_0) \in \mathrm{NTIME}(T_2)$. Let $f{:}N \to N$ be any recursively bounded but strictly increasing function. We prove that $L_{T_2}(U_0) \notin \mathrm{NTIME}(T_1)$ for any time bound T_1 with $T_1(f(n+1)) \in o(T_2(f(n)))$.

Suppose that U_1 accepts $L_{T_2}(U_0)$ within time T_1, where $T_1(f(n+1)) \in o(T_2(f(n)))$. By Lemma 4, there is an acceptor U for

---

†The operator gap theorem ([Con72], [Yng71]) shows that such results are impossible without some "honesty" condition on T_2 such as T_2 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 $o(n \cdot T(n+1))$.

$$L(U_1) \cup L(U_0) = L_{T_2}(U_0) \cup L(U_0) = L(U_0)$$

such that for every $e \in L^2_{p.c.}$,

$$\mathrm{Time}_U(ex) \le \begin{cases} T_1(|ex|), \text{ if } c_e \cdot \mathrm{Time}_{M_e}(x) \le T_2(|ex|); \\ c_e \cdot \mathrm{Time}_{M_e}(x), \text{ in any event.} \end{cases}$$

Note that when $T_1(|ex|) < \mathrm{Time}_{M_e}(x) \le T_2(|ex|)/c_e$, 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. This extreme efficiency will lead below to a contradiction of Lemma 9.

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

1. Check that the input string is a member of $L^2_{p.c.} \cdot 1^* \cdot 0^*$, and parse it into $e \in L^2_{p.c.}$, $x \in \{1\}^*$, and 0^k. Condition 1 guarantees that this can be done in time that is linear in the length of the input string.
2. Use a clock for the running time T to determine whether $k \ge T(|x|)$. This requires at most k steps, so it can be done in linear time, too.
3. If $k \ge T(|x|)$, then erase everything but x and compute on input x according to the transition rules of M. For $x \in L(M)$, since $\mathrm{Time}_M(x) \le T(|x|) \le k$, this step can be performed in linear time, too.
4. If $k < T(|x|)$, then pad the input string to $ex0^{k'}$ for some non-

deterministically 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 linear time plus $\mathrm{Time}_U(ex0^{k'})$.

To summarize the behavior of M' on $ex0^k$,

$k \geq T(|x|) \Rightarrow$ behave like M on x;

$k < T(|x|) \Rightarrow$ behave like U on $ex0^{k'}$ for some $k' > k$ (thus simulating M_e on $x0^{k'}$).

To summarize the timing for $ex0^k \in L(M')$,

$$\mathrm{Time}_{M'}(ex0^k) \leq \begin{cases} d_1 \cdot |ex0^k|, & \text{if } k \geq T(|x|); \\ d_1 \cdot |ex0^{k'}| + \mathrm{Time}_U(ex0^{k'}), & \text{if } k < T(|x|) \end{cases}$$

for some constant d_1 and every $k' > k$.

Applying Lemma 8 to obtain a 2-tape TM that accepts L(M') with only linear time loss, and then applying the recursion theorem (Lemma 12) to this machine, we get a program code e_0 for a 2-tape TM that accepts

$$L(M_{e_0}) = \{x0^k \mid e_0x0^k \in L(M')\} \subset 1^*0^*$$

within time

$$\mathrm{Time}_{M_{e_0}}(x0^k) \leq d_2 \cdot \mathrm{Time}_{M'}(e_0x0^k)$$

for some constant d_2.

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

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

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

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

$$x0^k \in L(M_{e_0}) \Leftrightarrow e_0x0^k \in L(M')$$

(by choice of e_0)

$\Leftrightarrow x \in L(M) = L$

(because by definition M' behaves like M in this case).

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

$$x0^k \in L(M_{e_0}) \Leftrightarrow e_0x0^k \in L(M')$$

(by choice of e_0)

$\Leftrightarrow e_0x0^{k'} \in L(U)$ for some $k' > k$

(because by definition M' behaves like U in this case)

$\Leftrightarrow x0^{k'} \in L(M_{e_0})$ for some $k' > k$

(because $e_0 \in L^2_{p.c.}$)

$\Leftrightarrow x \in L$

(by induction hypothesis). $\square$

Claim 2. For each sufficiently long string $x \in L$, the following holds for every $n \geq |e_0x|$:

$$\mathrm{Time}_{M_{e_0}}(x0^{f(n)-|e_0x|}) \leq d_3 \cdot T_1(f(n+1)),$$

where $d_3 = d_2d_1 + d_2$.

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

$$c_{e_0} \cdot d_3 \cdot T_1(f(n+1)) \leq T_2(f(n))$$

for every $n \geq |e_0x|$. (This uses the "translational" hypothesis $T_1(f(n+1)) \in o(T_2(f(n)))$.) We establish the claim for x by induction on n running down from n so large that $f(n) \geq |e_0x| + T(|x|)$ to $n = |e_0x|$.

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

$$\begin{aligned}\mathrm{Time}_{M_{e_0}}(x0^{f(n)-|e_0x|}) &\leq d_2 \cdot \mathrm{Time}_{M'}(e_0x0^{f(n)-|e_0x|}) \\ &\leq d_2 \cdot d_1 \cdot f(n) \\ &\quad \text{(because } f(n)-|e_0x| \geq T(|x|)) \\ &\leq d_3 \cdot T_1(f(n+1)).\end{aligned}$$

$|e_0x| \leq n \leq f(n) < |e_0x| + T(|x|)$:

Assume $\mathrm{Time}_{M_{e_0}}(x0^{f(n+1)-|e_0x|}) \leq d_3 \cdot T_1(f(n+2))$. Then

$$\begin{aligned}c_{e_0} \cdot \mathrm{Time}_{M_{e_0}}(x0^{f(n+1)-|e_0x|}) &\leq c_{e_0} \cdot d_3 \cdot T_1(f(n+2)) \\ &\leq T_2(f(n+1)) \\ &\quad \text{(because n is so large).}\end{aligned}$$

Therefore,

$$\mathrm{Time}_U(e_0x0^{f(n+1)-|e_0x|}) \leq T_1(f(n+1)).$$

Therefore,

$$\begin{aligned}\mathrm{Time}_{M_{e_0}}(x0^{f(n)-|e_0x|}) &\leq d_2 \cdot \mathrm{Time}_{M'}(e_0x0^{f(n)-|e_0x|}) \\ &\leq d_2 \cdot d_1 \cdot f(n+1) + d_2 \cdot \mathrm{Time}_U(e_0x0^{f(n+1)-|e_0x|}) \\ &\quad \text{(by padding out to length } f(n+1) > f(n)) \\ &\leq d_2 \cdot d_1 \cdot f(n+1) + d_2 \cdot T_1(f(n+1)) \\ &\leq d_3 \cdot T_1(f(n+1)). \quad \square\end{aligned}$$

Claim 3. For each sufficiently long string $x \in L$,

$$\mathrm{Time}_{M_{e_0}}(x) \leq d_3 \cdot T_1(f(|e_0x|+1)).$$

<u>Proof</u>. Let $x \in L$ be as long as in the proof of Claim 2. Then

$$c_{e_0}\cdot \mathrm{Time}_{M_{e_0}}\left(x0^{f(|e_0x|+1)-|e_0x|}\right) \le c_{e_0}\cdot d_3\cdot T_1(f(|e_0x|+1))$$
$$\le T_2(f(|e_0x|+1)).$$

Therefore,

$$\mathrm{Time}_U\left(e_0x0^{f(|e_0x|+1)-|e_0x|}\right) \le T_2(f(|e_0x|+1)).$$

Therefore,

$$\mathrm{Time}_{M_{e_0}}(x) \le d_2\cdot \mathrm{Time}_{M'}(e_0x)$$
$$\le d_2\cdot d_1\cdot f(|e_0x|+1) + d_2\cdot \mathrm{Time}_U\left(e_0x0^{f(|e_0x|+1)-|e_0x|}\right)$$
(by padding out to length $f(|e_0x|+1) > |e_0x|$)
$$\le d_2\cdot d_1\cdot f(|e_0x|+1) + d_2\cdot T_1(f(|e_0x|+1))$$
$$\le d_3\cdot T_1(f(|e_0x|+1)). \quad \square$$

Finally, by Lemmas 4 and 5, 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 $d_3\cdot T_1(f(|e_0|+n+1))$. If g is a nondecreasing primitive upper bound for f_{e_0} then

$$d_3\cdot T_1(f(|e_0|+n+1)) \in o(T_2(f(|e_0|+n)))$$
$$\subseteq o\Big(\sum_{n'\le g(2n)} T_2(n')\Big);$$

thus Lemma 7 gives $L \in \mathrm{DTIME}\Big(\sum_{n'\le g(2n)} T_2(n')\Big)$. But $L \subseteq \{1\}^*$ was chosen arbitrarily, and by Lemma 9 not every such language $L \subseteq \{1\}^*$ can be in the particular class $\mathrm{DTIME}\Big(\sum_{n'\le g(2n)} T_2(n')\Big)$: a contradiction. $\square$

<u>Example</u>. For an arbitrary set A of nonnegative integers, let

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

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

To see that $\mathrm{NTIME}(n^2 \cdot \delta(n)) \subsetneq \mathrm{NTIME}(n^3)$, just apply Theorem 13 with

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

In many applications it suffices to have Theorem 13 for the single function $f(n) = n$, especially if we are concerned only with nondecreasing time bounds.

Corollary 14. If T_2 is a running time, then

$$\mathrm{NTIME}(T_2) - \bigcup \{\mathrm{NTIME}(T_1) \mid T_1(n+1) \in o(T_2(n))\}$$

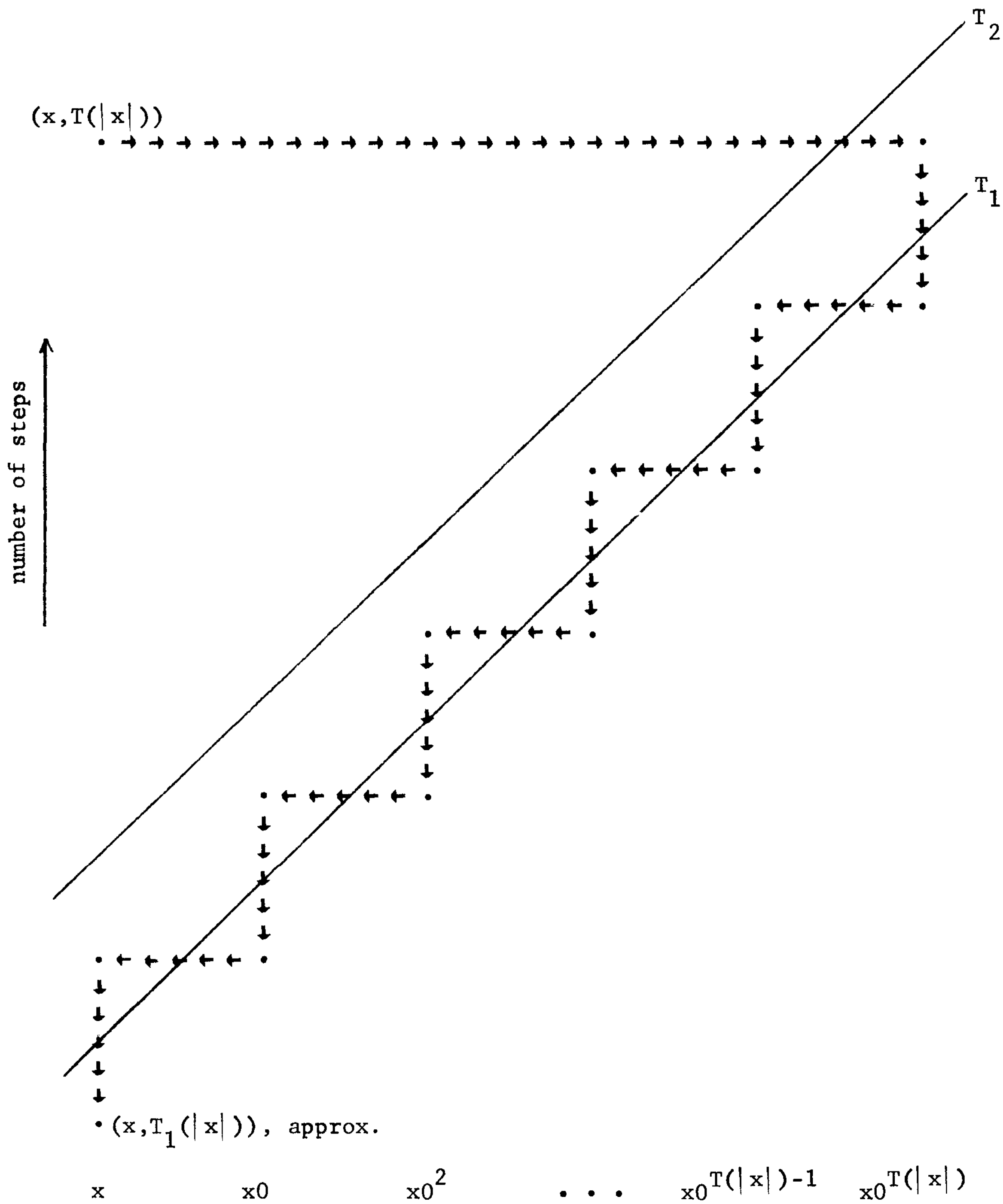
contains a language over $\{0,1\}$.

The informal diagram in Figure 1 illustrates how the proof of Theorem 13 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 $f(n) = n$. 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

Figure 1. → pad

← unpad

↓ speed up by simulation



level of T_1 and T_2.

The "translational" condition $T_1(n+1) \in o(T_2(n))$ 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 diagonalization within time bound T_2 (Theorem 1) yields stronger results than does Corollary 14. Because Corollary 14 applies for $T_1(n+1) \in o(T_2(n))$ and Theorem 1 applies for $\log T_2(n) \notin O(T_1(n))$, it is easy to see that Corollary 14 contributes new results precisely when $\log T_2(n+1) \in o(T_2(n))$.

To see the strength of Corollary 14, let $\log^* n = \min\{k \mid n \le \underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{k}\}$.

For constants $c > 1$, $r \ge 1$ whose digits in radix notation can be generated rapidly, and in particular for rational c, 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

$$\mathrm{NTIME}(n^r) \subsetneqq \mathrm{NTIME}(n^r \cdot \log^* n) \subsetneqq \mathrm{NTIME}(n^r \cdot (\log^* n)^2) \subsetneqq \cdots,$$
$$\mathrm{NTIME}(c^n) \subsetneqq \mathrm{NTIME}(c^n \cdot \log^* n) \subsetneqq \mathrm{NTIME}(c^n \cdot (\log^* n)^2) \subsetneqq \cdots.$$

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

Corollary 14 obviously implies that

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

In fact we can strengthen these results to

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

$$\mathrm{NTIME}(2^{2^n}) \subsetneq \mathrm{NTIME}(2^{2^{n+1}})$$

by appeal to the following corollary.

<u>Corollary 15</u>. If T_2 is a running time, then

$$\bigcup \{\mathrm{NTIME}(T_1) \mid T_1(n+1) \in O(T_2(n)),\ T_1(n) \in o(T_2(n))\} \subsetneq \mathrm{NTIME}(T_2);$$

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

<u>Proof</u>. Because $T_1(n) \in o(T_2(n))$ implies $T_1((n+1)+1) \in o(T_2(n+2))$, Corollary 14 gives a language $L \subset \{0,1\}^*$ in

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

Applying Lemma 11 with

$$\mathcal{S}_1 = \{T_1 \mid T_1(n+1) \in O(T_2(n)),\ T_1(n) \in o(T_2(n))\},$$

$$\mathcal{S}_2 = \{T_2\},$$

$$p_1(n) = n + 1,$$

$$p_2(n) = n + 2,$$

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

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

Containment holds by Lemma 7. $\square$

<u>Remarks</u>. (i) Lemma 11 goes through equally well if we pad to the left rather than to the right. For this remark, then, 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_0 the universal simulator of Theorem 13, $L = L_{T_2(n+2)}(U_0)$ satisfies the condition for choosing L in the proof of Corollary 15. One might suppose therefore that $L_{T_2(n)}(U_0)$ would be a witness language for Corollary 15. If we slightly modify our program coding by concatenating a single 1 in front of each old program code and if we let U_0' be the naturally derived new universal simulator, then we do indeed get $L_{T_2(n+1)}(U_0') = 1 \cdot L_{T_2(n+2)}(U_0) = p_1(L)$. Similarly, if we further concatenate a 0 in front of each program code and let U_0'' be derived from U_0' by taking this into account, then we get $L_{T_2(n)}(U_0'') = 01 \cdot L_{T_2(n+2)}(U_0) = p_2(L)$. Yet we can show only that <u>either</u> $L_{T_2(n)}(U_0'')$ or $L_{T_2(n+1)}(U_0')$ is a witness to Corollary 15. We leave it open whether there is necessarily a witness language of the form $L_{T_2(n)}(U_0)$ and whether the particular choice of U_0 affects whether $L_{T_2(n)}(U_0)$ is such a language.

(ii) Corollary 15 contributes new results (over Theorem 1) precisely when $\log T_2(n+1) \in O(T_2(n))$.

It is interesting to note that the containments corresponding to the examples following Corollary 14 are not known to be proper for deterministic time (DTIME). The fundamental reason is that Lemma 8 is not known for DTIME. The proof of Theorem 13 in the case of such an easy function f as $f(n) = n$ does carry over in every other detail, however, to give Theorem 16.

<u>Theorem 16</u>. Suppose that there is some fixed k such that for each deterministic TM acceptor M there is a deterministic k-tape TM acceptor M' and a constant c such that $L(M') = L(M)$ and $\mathrm{Time}_{M'}(x) \leq c \cdot f(\mathrm{Time}_M(x))$. Then

$$\mathrm{DTIME}(T_2) - \bigcup \{\mathrm{DTIME}(T_1) \mid f(T_1(n+1)) \in o(T_2(n))\} \neq \emptyset$$

for every running time T_2.

Remark. We do not require that M' be effectively constructable from M; if it were, then we could actually diagonalize to get

$$\mathrm{DTIME}(T_2) - \bigcup\{\mathrm{DTIME}(T_1) \mid T_2(n) \notin O(f(T_1(n)))\} \neq \emptyset,$$

a somewhat stronger result.

Example. If we should discover even a nonconstructive proof that for each deterministic TM acceptor M there is a deterministic 5-tape TM acceptor M' and a constant c such that L(M') = L(M) and $\mathrm{Time}_{M'}(x) \leq c \cdot \mathrm{Time}_M(x)$, then we could conclude that

$$\mathrm{DTIME}(T_2) - \bigcup\{\mathrm{DTIME}(T_1) \mid T_1(n+1) \in o(T_2(n))\} \neq \emptyset$$

for every running time T_2.

Padding strings over a one-letter alphabet by one character at a time does not leave them decodable, so we cannot hope to use our method to get a result as strong as Corollary 14 for languages over a one-letter alphabet. The final result of this chapter 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	$\lceil \log_2 n \rceil$
	$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n}$	$\log^* n.$

<u>Theorem 17</u>. If T_2 is a running time and f is real-time countable,† then there is a language over $\{1\}$ in

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

<u>Proof</u>. Let T_2 be a running time, and let f be real-time countable. To adapt the proof of Theorem 13, we must construct a witness language as the T_2-cutoff of some "universal simulator" having input alphabet $\{1\}$. We start with U_0 as in the earlier proof; i. e., U_0 is the universal simulator of Condition 2 for k = 2.

Define an injection $g:\{0,1\}^* \to N$ so that the binary representation of the integer g(x) is 1x; i. e., we concatenate a high-order digit 1 to x to get the binary representation of g(x). Define another function $h:\{0,1\}^* \to N$ by the conditions

$$h(x0) = h(x) + \lceil f^{-1} \rceil(h(x)),$$

$$h(x1) = f(g(x1)) + g(x1) - 1,$$

$$h(\lambda) = f(1).$$

Because g is an injection and f(n)+n-1 is strictly increasing, h(x1) = h(y1) only if x = y. Because $n+\lceil f^{-1} \rceil(n)$ is strictly increasing, h(x0) = h(y0) only if h(x) = h(y). Unless there are strings x, y with h(x0) = h(y1), therefore, h must be an injection. For such strings to exist, the ranges of the strictly increasing functions $n+\lceil f^{-1} \rceil(n)$ and f(n)+n-1 must intersect; but

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

†A strictly increasing function $f:N \to N$ is <u>real-time countable</u> [Yam62] 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.)

$$< f(n) + n - 1$$

$$< f(n) + n$$

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

so the ranges do not intersect and h is an injection.

Because f is real-time countable, we can compute $1^{h(x)}$ from x or x from $1^{h(x)}$ within time proportional to h(x), within time $2 \cdot h(x)$ if we wish. From U_0 we construct U_0' 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_0.

By Lemma 6, $L_{T_2}(U_0') \in NTIME(T_2)$. We prove that $L_{T_2}(U_0') \notin NTIME(T_1)$ for any time bound T_1 with $T_1(n + \lceil f^{-1} \rceil (n)) \in o(T_2(n))$.

Suppose that U_1' accepts $L_{T_2}(U_0')$ within time T_1, where $T_1(n + \lceil f^{-1} \rceil (n)) \in o(T_2(n))$. By Lemma 4, there is an acceptor U' for

$$L(U_1') \cup L(U_0') = L_{T_2}(U_0') \cup L(U_0') = L(U_0')$$

such that for every $e \in L^2_{p.c.}$,

$$Time_{U'}(1^{h(ex)}) \le \begin{cases} T_1(h(ex)), \text{ if } 2 \cdot h(ex) + c_e \cdot Time_{M_e}(x) \le T_2(h(ex)); \\ 2 \cdot h(ex) + c_e \cdot Time_{M_e}(x), \text{ in any event.} \end{cases}$$

Finally, 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 U'.

Then

$$L(U) = \{x | \ 1^{h(x)} \in L(U')\}$$
$$= \{x | \ 1^{h(x)} \in L(U_0')\}$$
$$= L(U_0);$$

and, for every $e \in L^2_{p.c.}$,

$$\mathrm{Time}_U(ex) \le 2\cdot h(ex) + \mathrm{Time}_{U'}(1^{h(ex)})$$

$$\le \begin{cases} 2\cdot h(ex) + T_1(h(ex)), \text{ if } 2\cdot h(ex) + c_e\cdot \mathrm{Time}_{M_e}(x) \\ \qquad\qquad\qquad\qquad\qquad \le T_2(h(ex)); \\ 4\cdot h(ex) + c_e\cdot T_{M_e}(x), \text{ in any event.} \end{cases}$$

For any recursive $L \subset \{1\}^*$, we can use U just as in the proof of Theorem 13 to get a 2-tape TM acceptor M_{e_0} for $L\cdot 0^*$, with

$$\mathrm{Time}_{M_{e_0}}(x0^k) \le \begin{cases} d_1\cdot|e_0x0^k|, \text{ if } k \ge T(|x|); \\ d_1\cdot|e_0x0^{k+1}| + d_1\cdot\mathrm{Time}_U(e_0x0^{k+1}), \text{ if } k < T(|x|) \end{cases}$$

for some sufficiently large constant d_1 and some appropriate time bound T.

<u>Claim</u>. For each sufficiently long string $x \in L$, the following holds for every k:

$$\mathrm{Time}_{M_{e_0}}(x0^k) \le d_2\cdot T_1(h(e_0x0^{k+1})),$$

where $d_2 = 4d_1$.

<u>Proof</u>. Let $x \in L$ be so long that

$$(2 + c_{e_0}\cdot d_2)\cdot T_1(n + \ulcorner f^{-1}\urcorner(n)) \le T_2(n)$$

for every $n \ge h(e_0x)$. We establish the claim for x by induction on k running down from $k \ge T(|x|)$ to $k = 0$.

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

$$\mathrm{Time}_{M_{e_0}}(x0^k) \le d_1\cdot|e_0x0^k|$$

$$\le d_2\cdot T_1(h(e_0x0^{k+1})).$$

$k < T(|x|)$:

Assume $\mathrm{Time}_{M_{e_0}}(x0^{k+1}) \le d_2 \cdot T_1(h(e_0x0^{k+2}))$. Then

$$2 \cdot h(e_0x0^{k+1}) + c_{e_0} \cdot \mathrm{Time}_{M_{e_0}}(x0^{k+1})$$

$$\le 2 \cdot h(e_0x0^{k+1}) + c_{e_0} \cdot d_2 \cdot T_1(h(e_0x0^{k+2}))$$

$$= 2 \cdot h(e_0x0^{k+1}) + c_{e_0} \cdot d_2 \cdot T_1(h(e_0x0^{k+1}) + \lceil f^{-1} \rceil (h(e_0x0^{k+1})))$$

$$\le (2 + c_{e_0} \cdot d_2) \cdot T_1(h(e_0x0^{k+1}) + \lceil f^{-1} \rceil (h(e_0x0^{k+1})))$$

$$\le T_2(h(e_0x0^{k+1}))$$

(because x is so long).

Therefore,

$$\mathrm{Time}_U(e_0x0^{k+1}) \le 2 \cdot h(e_0x0^{k+1}) + T_1(h(e_0x0^{k+1}))$$
$$\le 3 \cdot T_1(h(e_0x0^{k+1})).$$

Therefore,

$$\mathrm{Time}_{M_{e_0}}(x0^k) \le d_1 \cdot |e_0x0^{k+1}| + d_1 \cdot \mathrm{Time}_U(e_0x0^{k+1})$$
$$\le d_1 \cdot |e_0x0^{k+1}| + 3d_1 \cdot T_1(h(e_0x0^{k+1}))$$
$$\le d_2 \cdot T_1(h(e_0x0^{k+1})). \quad \square$$

If H is a nondecreasing recursive function so large that $h(y) \le H(|y|)$ for all $y \in \{0,1\}^*$, then the claim gives the following for every sufficiently long $x \in L$:

$$\mathrm{Time}_{M_{e_0}}(x) \le d_2 \cdot T_1(h(e_0x0))$$
$$= d_2 \cdot T_1(h(e_0x) + \lceil f^{-1} \rceil (h(e_0x)))$$
$$\le T_2(h(e_0x))$$
$$\le \sum_{n' \le H(|e_0| + |x|)} T_2(n')$$

$$\leq \sum_{n' \leq H(2 \cdot |x|)} T_2(n').$$

It follows by Lemmas 4, 5 that $L \in \mathrm{NTIME}(\sum_{n' \leq H(2n)} T_2(n'))$. Since L is an arbitrary recursive language over $\{1\}$, this contradicts Lemma 9. □

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

We close with a list of open questions.

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

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

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

4. That a language L is not a member of $\mathrm{NTIME}(T_1)$ means only that every acceptor M for L has $\mathrm{Time}_M(x) > T_1(|x|)$ for strings $x \in L$ of <u>infinitely many lengths</u>. Stronger senses of lower bounds, requiring

that $\mathrm{Time}_M(x) > T_1(|x|)$ 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 [Blm67], [Lyn72], [GB74], for example.) It is known, for example, that there is a language L that requires $2^{|x|}$ many steps deterministically on almost every string $x \in L$ but that can be accepted deterministically within time $(2+\varepsilon)^n$ for any $\varepsilon > 0$. Our methods do not give such results for nondeterministic acceptance time complexity, so we leave it open whether there is a language $L \in \mathrm{NTIME}((2+\varepsilon)^n)$ that requires, even on nondeterministic machines, $2^{|x|}$ 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 13 is whether we can allow f to range over all one-one functions rather than just strictly increasing, recursively bounded ones. A plausible proof strategy is to design M' in the proof of Theorem 13 so that, in the case $k < T(|x|)$, it pads <u>or</u> <u>unpads</u> $ex0^k$ to $ex0^{k'}$ for some nondeterministically chosen $k' \neq k$. Under this strategy, however, 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? (Cf., Chapter Three.)

8. Is there any language in $\mathrm{NTIME}(T_2)$ that requires more time than the language $L_{T_2}(U_0)$ in the proof of Theorem 13?

9. In the conclusion of Lemma 11, can we exhibit a single language that must definitely belong to $\bigcap \{\mathrm{NTIME}(T_2) \mid T_2 \in \mathcal{S}_2\} - \bigcup \{\mathrm{NTIME}(T_1) \mid T_1 \in \mathcal{S}_1\}$? (Cf., Remark (i) following Corollary 15.)

CHAPTER THREE

SPACE SEPARATION THEOREMS FOR OFF-LINE TURING MACHINES

1. Basic definitions

To study Turing machine storage space complexity, we adopt a Turing machine model that has a read-only input tape and a single read-write worktape. The input string is received between the special endmarkers ¢, \$ on the input tape and is read by a single read-only input head which is allowed to move freely between the endmarkers. The worktape is infinite to the right only. For technical reasons, we allow any fixed finite number of freely moving, but initially left-adjusted, read-write heads on the worktape. The worktape heads can detect both each other and the left end of the worktape, and they are never required to write conflicting symbols on a single tape square in the same step or to shift left past the left end of the worktape. We refer to such an automaton as an <u>off-line TM</u>. An off-line TM with $m \geq 2$ symbols in its worktape alphabet (counting the blank symbol, which may be used without restriction even in overwrite instructions) and $\ell \geq 1$ worktape heads is called an <u>(m,ℓ)-machine</u> or just an <u>m-machine</u>. The deterministic restrictions of these automata are called <u>deterministic off-line TMs</u> and <u>deterministic (m,ℓ)-machines</u>, respectively.

An off-line TM can act as an <u>acceptor</u> by halting in some specified accepting state and with a blank worktape at the end of some computations.

<u>Definition</u>. Let M be any off-line TM acceptor. M <u>accepts</u> the string x

if there is some accepting computation by M on input x. M accepts the language $L(M) = \{x \mid M \text{ accepts string } x\}$.† For $x \in L(M)$, $\mathrm{Space}_M(x)$ is the minimum number of distinct worktape squares visited by the worktape heads of M in an accepting computation by M on x; for $x \notin L(M)$, $\mathrm{Space}_M(x) = \infty$. For $S{:}N \to N$, define

$$L_S(M) = \{x \mid \mathrm{Space}_M(x) \le S(|x|)\},$$

$$\mathrm{NSPACE}(S,m,\ell) = \{L \mid L = L(M) = L_S(M) \text{ for some } (m,\ell)\text{-machine } M\},$$

$$\mathrm{NSPACE}(S,m) = \bigcup \{\mathrm{NSPACE}(S,m,\ell) \mid \ell \ge 1\},$$

$$\mathrm{NSPACE}(S) = \bigcup \{\mathrm{NSPACE}(S,m,\ell) \mid m \ge 2,\ \ell \ge 1\},$$

$$\mathrm{DSPACE}(S,m,\ell) = \{L \mid L = L(M) = L_S(M) \text{ for some deterministic } (m,\ell)\text{-machine } M\},$$

$$\mathrm{DSPACE}(S,m) = \bigcup \{\mathrm{DSPACE}(S,m,\ell) \mid \ell \ge 1\},$$

$$\mathrm{DSPACE}(S) = \bigcup \{\mathrm{DSPACE}(S,m,\ell) \mid m \ge 2,\ \ell \ge 1\}.$$

We call $L_S(M)$ the S-cutoff of M, and we say M accepts within space S if $L(M) = L_S(M)$. If NSPACE(S) contains languages which are not regular, then S is a space bound. Every subscripted or primed S mentioned below is assumed to be a space bound.

Proposition 1. No space bound S satisfies $S(n) \in o(\log \log n)$.

Proof. See [HU69a]. □

It is well known that the NSPACE(S) and DSPACE(S) complexity classes are generally insensitive to machine model design variations. The $\mathrm{NSPACE}(S,m,\ell)$ and $\mathrm{DSPACE}(S,m,\ell)$ complexity classes, on the other hand, are sensitive to machine model design; but the differences are usually

†Acceptance is to be distinguished from "recognition." If the off-line TM M can accept either $L \subset \Sigma^*$ or $\Sigma^* - L$ (within space S), depending on accepting state designation, then M recognizes L (within space S).

minor. Following are comments on the effects of some common design variations.

1. Suppose that we redefine our (m,ℓ)-machine model so that its worktape heads cannot detect each other. If the resulting complexity classes are NSPACE'(S,m,ℓ), DSPACE'(S,m,ℓ), then we have

$$\text{NSPACE}'(S,m,\ell) = \text{NSPACE}(S,m,\ell),$$

$$\text{DSPACE}'(S,m,\ell) = \text{DSPACE}(S,m,\ell).$$

To see that detectability is no real advantage, it suffices to observe that detection can be simulated by the redesigned model. The trick is to make a temporary change under each worktape head in turn, letting each head discover which other heads' temporary changes take place on the square it scans.

2. Suppose we redesign our (m,ℓ)-machine model so that it cannot detect the left end of its worktape but instead halts without accepting if it shifts past that end. If the resulting complexity classes are NSPACE'(S,m,ℓ), DSPACE'(S,m,ℓ), then we have

$$\text{NSPACE}'(S,m,\ell) \subset \text{NSPACE}(S,m,\ell) \subset \text{NSPACE}'(S,m,\ell+1),$$

$$\text{DSPACE}'(S,m,\ell) \subset \text{DSPACE}(S,m,\ell) \subset \text{DSPACE}'(S,m,\ell+1).$$

Our model simulates the redesigned one simply by halting when it detects that the transition rules would lead to a shift off the end of the worktape. The redesigned model simulates ours by permanently stationing an extra worktape head at the leftmost worktape square. Detection of that square can then be effected by the trick of comment 1 above.

3. Suppose we redesign our (m,ℓ)-machine model so that its worktape is infinite in both directions. If the resulting complexity classes

are NSPACE'(S,m,ℓ), DSPACE'(S,m,ℓ), then we have

$$\text{NSPACE}'(S,m,\ell) \subset \text{NSPACE}(S,m,\ell+1) \subset \text{NSPACE}'(S,m,\ell+2),$$

$$\text{DSPACE}'(S,m,\ell) \subset \text{DSPACE}(S,m,\ell+1) \subset \text{DSPACE}'(S,m,\ell+2).$$

To simulate the redesigned model, our model must be able to provide new worktape squares for shifts past the left end of the worktape. It suffices to shift all the work to the right (making temporary use of the worktape head that needs the new tape square), and this is made possible by using an extra worktape head to mark the rightmost worktape square that has been visited. (Nothing to the right of this head need be shifted because it is all blank anyway.) The redesigned model simulates ours by permanently stationing an extra worktape head at the initial worktape square and treating that square as the left end of the worktape.

In the nondeterministic case we actually have

$$\text{NSPACE}'(S,m,\ell) \subset \text{NSPACE}(S,m,\ell)$$

because our model can nondeterministically guess where to start its simulation so that no shift past the left end of its own worktape is called for.

4. Suppose that we redefine acceptance by our (m,ℓ)-machine model so that a blank tape is <u>not</u> necessary. If the resulting complexity classes are NSPACE'(S,m,ℓ), DSPACE'(S,m,ℓ), then we have

$$\text{NSPACE}'(S,m,\ell) \subset \text{NSPACE}(S,m,\ell+1) \subset \text{NSPACE}'(S,m,\ell+2),$$

$$\text{DSPACE}'(S,m,\ell) \subset \text{DSPACE}(S,m,\ell+1) \subset \text{DSPACE}'(S,m,\ell+2).$$

Our model simulates the redesigned one simply by erasing its worktape when the transition rules call for acceptance. This is made possible by using an extra worktape head to mark the rightmost worktape square that has been visited. (Nothing to the right of this head need be erased because it is all blank anyway.) The redesigned model simulates ours by *checking* whether its worktape is blank before entering the accepting state. This is made possible by again using an extra worktape head to mark the rightmost worktape square that has been visited.

In the nondeterministic case we actually have

$$\mathrm{NSPACE}'(S,m,\ell) \subset \mathrm{NSPACE}(S,m,\ell)$$

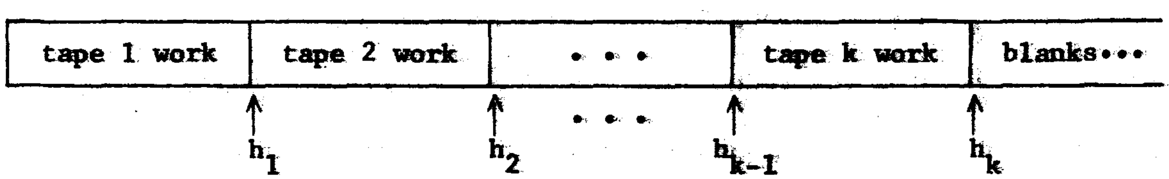
because our model can nondeterministically guess where to start erasing in preparation for acceptance.

5. Suppose we redesign our (m,ℓ)-machine model so that it has *k* worktapes. If the resulting complexity classes (obtained by counting the *total* number of visited worktape squares, the *total* alphabet size, and the *total* number of worktape heads) are $\mathrm{NSPACE}'(S,m,\ell)$, $\mathrm{DSPACE}'(S,m,\ell)$, then we have

$$\mathrm{NSPACE}'(S,m,\ell) \subset \mathrm{NSPACE}(S,m,\ell+k) \subset \mathrm{NSPACE}'(S,m,\ell+k),$$

$$\mathrm{DSPACE}'(S,m,\ell) \subset \mathrm{DSPACE}(S,m,\ell+k) \subset \mathrm{DSPACE}'(S,m,\ell+k).$$

Our model simulates the redesigned one by storing the concatenation of the visited portions of the k tapes on its one tape. The k extra heads $h_1,\ldots,h_k$ are used to delimit these k segments. New worktape squares are provided where needed by shifting work right, much as in comment 3 above.



The simulation of our model by the redesigned one is trivial.

6. Suppose that we redefine our (m,ℓ)-machine model so that the blank symbol is reserved for worktape squares that have not yet been visited. If the resulting complexity classes (obtained by counting only nonblank worktape symbols now) are NSPACE'(S,m,ℓ), DSPACE'(S,m,ℓ), then we have

$$\mathrm{NSPACE}'(S,m,\ell) \subset \mathrm{NSPACE}(S,m,\ell+1) \subset \mathrm{NSPACE}'(S,m+1,\ell+1),$$

$$\mathrm{DSPACE}'(S,m,\ell) \subset \mathrm{DSPACE}(S,m,\ell+1) \subset \mathrm{DSPACE}'(S,m+1,\ell+1).$$

Our model simulates the redesigned one by using its blank symbol for one of the ordinary symbols of the new model. An extra worktape head is used to mark the rightmost worktape square that has been visited, beyond which the blank represents the true blank symbol of the simulated machine. The redesigned model simulates ours by using an extra unrestricted symbol along with its restricted blank symbol to represent the unrestricted blank of our model.

The relations among design decisions revealed by considerations such as those above provide a convenient way of converting the results of this chapter to good results for any of the redesigned machine models. Slightly better results often are obtained by converting the original proofs, however, making better use of nondeterminism (cf., comments 3, 4 above) or of worktape heads not yet fully utilized, for example.

2. Basic containment relations

In this section we present all the known containment relations among the complexity classes defined in Section 1. The trivial relations are that no language is lost from a complexity class by allowing nondeterminism, additional space, additional worktape symbols, or additional worktape heads.

Only slightly less trivial is the use of the finite state control to save space.

Proposition 2. If $S_2(n)-S_1(n) \in O(1)$, then

$NSPACE(S_2,m,\ell) \subset NSPACE(S_1,m,\ell)$,

$DSPACE(S_2,m,\ell) \subset DSPACE(S_1,m,\ell)$.

It follows, for example, that the complexity class $DSPACE(n^{1/2},2,1)$ is not affected by how we round the square root. For convenience, therefore, we allow such an imprecise specification of a space bound when the precise specification is not relevant.

The basic relationship that $DSPACE(S_2) \subset DSPACE(S_1)$ whenever $S_2 \in O(S_1)$ appears in [SHL65]. It allows us to speak of DSPACE(log n) without specifying the base of the logarithm, for example. Our next proposition generalizes the relationship.

Proposition 3. If $S(n) \leq \delta \cdot S'(n)$ for some fixed rational number $\delta \leq \log_m m'$, then

$NSPACE(S,m,\ell) \subset NSPACE(S',m',\ell)$,

$DSPACE(S,m,\ell) \subset DSPACE(S',m',\ell)$.

Proof sketch. Say $\delta = i/j$ for positive integers i, j. As $m^i \leq m'^j$, we can encode the contents of i m-symbol-resolution worktape squares in j m'-symbol-resolution worktape squares. $\square$

Example. For k a positive integer,

$$\mathrm{NSPACE}(k \cdot S, m) = \mathrm{NSPACE}(S, m^k).$$

Additional worktape heads often can satisfy an apparent need for additional worktape symbols. Our technical reason for allowing several worktape heads is that additional worktape heads amount to much less additional space than additional symbols do. Proposition 3 establishes the close relationship between worktape symbols and a linear multiple of worktape space, and our next two propositions establish the close relationship between worktape heads and the logarithm of worktape space.

Proposition 4. For every $\epsilon > 0$,

$$\mathrm{NSPACE}(S,\ m,\ \ell+k) \subset \mathrm{NSPACE}(S + (k+1+\epsilon)\cdot\log_m S,\ m,\ \ell),$$

$$\mathrm{DSPACE}(S,\ m,\ \ell+k) \subset \mathrm{DSPACE}(S + (k+1+\epsilon)\cdot\log_m S,\ m,\ \ell).$$

Proof sketch. Let M be an $(m, \ell+k)$-machine that accepts within space $S(n)$. We wish to design an (m, ℓ)-machine M' that simulates M within space $S(n) + (k+1+\epsilon)\cdot\log_m S(n)$.

The first ℓ-1 heads of M can be simulated by the first ℓ-1 heads of M'. The position of each of the other k+1 heads of M can be stored by M' as the m-ary representation of that position. If these k+1 strings are properly delimited, the last head of M' can carry them around and access them to simulate all of the last k+1 heads of M. Since only finitely many (k+2) delimiting marks are required, a single extra bit inserted every j symbols of the list can be used in conjunction with finite-state memory to locate the marks. Each of these bits is set to 1 if and only if at least one of the k+2 delimiting marks should be located on one of the next j worktape squares. Since j and k are fixed, the precise locations of the marks <u>relative to the bits that are set to 1</u> can be maintained by finite-state memory. The list itself accounts for an extra space requirement of $(k+1)\cdot\log_m S(n)$, and the additional requirement for delimiters can be kept to an arbitrarily small fraction of that by choosing j large enough.

Clearly, M' is deterministic if M is. □

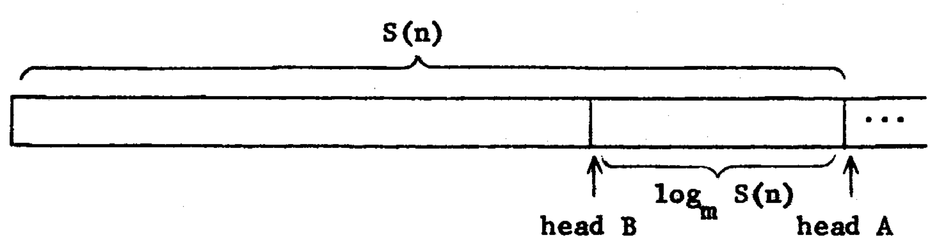
<u>Proposition 5</u>.

$$\mathrm{NSPACE}(S + k\cdot\log_m S,\ m,\ \ell) \subset \mathrm{NSPACE}(S,\ m,\ \ell+k+3),$$

$$\mathrm{DSPACE}(S + k\cdot\log_m S,\ m,\ \ell) \subset \mathrm{DSPACE}(S,\ m,\ \ell+k+3).$$

<u>Proof sketch</u>. Let M be an (m,ℓ)-machine that accepts within space $S(n) + k\cdot\log_m S(n)$. We wish to design an $(m,\ell+k+3)$-machine M' that simulates M within space $S(n)$.

If S happens to be easy to compute, then M' can start by stationing head A at worktape square $S(n)$ and head B at worktape square $S(n) - \log_m S(n)$.



The worktape of M is conceptually parsed into an initial segment of length $S(n) - \log_m S(n)$ and k+1 "pages," each of length $\log_m S(n)$. The initial segment will always reside in the first $S(n) - \log_m S(n)$ worktape squares of M'. One page at a time can reside in the worktape squares of M' delimited by heads B and A. Each page not residing there can be stored as a worktape head position, the page being the m-ary representation of the position.

In its simulation, M' tries to use ℓ of its worktape heads to behave like M'. A record of the current location of each page (resident or stored as some head position) and the page currently scanned by each head of M is maintained in the finite-state control of M'. When the heads of M scan different pages or move from page to page, "paging" is required. None of the ℓ simulating heads is moved from its proper location, but the one free head is used (with some help from head B) to store away the page that is currently resident. The required page is then loaded (again with help from head B), leaving the head that stored it free.

The simulation for arbitrary S differs in that heads A and B are restationed during the simulation according to how much space M has actually used. When M has used s' worktape squares, with

$$(s-1) + k\cdot\log_m (s-1) < s' \leq s + k\cdot\log_m s,$$

head A will be at position s and head B will be at position $s - \log_m s$. Restationing is required only when some simulating head is coincident with head A (and scanning the final page), so a second head is temporarily free to help the usual free head to determine whether head B must be restationed. (Head A is restationed every time s changes to s+1, but head B is restationed only when $\lfloor\log_m s\rfloor = \lfloor\log_m (s+1)\rfloor$.) Adjusting the pages on restationing is easily managed with some help from the finite-state control.

Clearly, M' is deterministic if M is. $\sqcap$

Our final basic containment relationship is the well known result of [Sav70].

<u>Proposition 6</u>. If $\log n = O(S(n))$, then

$$\mathrm{NSPACE}(S) \subset \mathrm{NSPACE}(S^2).$$

3. Notions of honesty

Qualitatively, a function is "honest" with respect to space if it can be computed without using space that is too much greater than both its argument and its value. It is easy to check that functions of practical interest are extremely honest. In fact, all of the common functions are of the following type. (See [Rit63].)

Definition. A function $f:N \to N$ is linear space honest if

$$\{bin(k)\#bin(f(k)) \mid k \in N\} \in DSPACE(n),$$

where we use the notation bin(k) for the binary representation of k (high-order bit first, say). Equivalently, f is linear space honest if

$$\{1^k 0^{f(k)} \mid k \in N\} \in DSPACE(\log n).$$

Our main goal in this chapter is to discover weak separation conditions for the complexity classes defined in Section 1; e. g., we seek a generally sufficient condition on S_1, S_2 for the nonemptiness of $NSPACE(S_2) - NSPACE(S_1)$. Well known "gap" theorems, however, show that any reasonable separation condition must include some sort of honesty requirement on at least one of the space bounds involved. (See [Bor72], [Con72], [Con73], [Yng71].) The most commonly used ([Ib72], [Sav70], [BGIW70], [Bk72]) notion of an honest space bound is the one we adopt.

Definition. If $f:N \to N$ does not belong to $O(1)$ and M is a deterministic off-line TM acceptor with $L(M) = 1^*$ and $Space_M(x) = f(|x|)$, then f is fully constructable and M fully constructs f.

Proposition 7.

(i) Every fully constructable function is a space bound.

(ii) Let f be linear space honest. If $\log n \in o(f(n))$, then f is fully constructable by a (2,1)-machine. If $\log n \in O(f(n))$, then f is still fully constructable.

(iii) Let S be fully constructable by an (m,ℓ)-machine, and let M be an (m,ℓ)-machine. Then $L_S(M) \in NSPACE(S,m,\ell+1)$; and if M is deterministic, then $L_S(M) \in DSPACE(S,m,\ell+1)$.

(iv) There are fully constructable space bounds in $O(\log \log n)$. (Cf., Proposition 1.)

(v) If S is fully constructable by an (m,ℓ)-machine and $1 \in o(S(n))$, then S satisfies

$$\log_m n - \ell\cdot\log_m \log_m n - S(n) \in O(1),$$

from which it follows that

$$\log_m n - S(n) \in O(\log \log n),$$

$$\log n \in O(S(n)).$$

Proof. (i) Let f be fully constructable. Since $f(n) \notin O(1)$, the language

$$\{0^k 1^j 0^k \mid j \geq 1,\ k \leq f(j+2k)\} \in DSPACE(f)$$

is not regular.

(ii) Assume f is linear space honest and $\log n \in o(f(n))$. Every language accepted deterministically within space n can actually be recognized (cf., footnote on page 43) within space n [HU69a], so let the (2,1)-machine M deterministically recognize $\{bin(k)\#bin(f(k)) \mid k \in N\}$ within space $O(n)$. To fully construct $f(n)$, compute according to the transition rules of M successively on

bin(n)#bin(0), bin(n)#bin(1), bin(n)#bin(2),...

until bin(f(n)) is discovered, and then convert bin(f(n)) to unary. As both log n and log f(n) belong to o(f(n)), this does fully construct f(n) for all sufficiently large n; the differences can be handled by the finite-state control. With care, the entire program can be carried out by a (2,1)-machine.

If only $\log n \in O(f(n))$, then using a large enough worktape alphabet keeps the search for bin(f(n)) smaller than f(n) itself.

(iii) An acceptor for $L_S(M)$ fully constructs space S and then computes according to the transition rules of M within that space. The extra head is left by the first phase to delimit the space used.

(iv) Define $f(n) = \min\{k \mid n \text{ is \underline{not} divisible by } k\}$, $S(n) = \log f(n)$. Obviously S is fully constructable. Let $\pi(k)$ be the number of primes smaller than k. The least n with $f(n) \geq k$ is the least common multiple of $\{k' \mid k' < k\}$, which certainly exceeds $2^{\pi(k)}$. Hence, $\pi(f(n)) \leq \log_2 n$. According to the prime number theorem [NZ60], $k/\log k \in O(\pi(k))$; so

$$\begin{aligned} f(n)^{1/2} &\in O(f(n)/\log f(n)) \\ &\subset O(\pi(f(n))) \\ &\subset O(\log n). \end{aligned}$$

Therefore,

$$S(n) = \log f(n) \in O(\log \log n).$$

(v) Suppose the d-state (m,ℓ)-machine M fully constructs S. We show below that $S(n) = S(n + kn!)$ for every positive integer k whenever $\log_m n - \ell \cdot \log_m \log_m n - S(n) > \log_m d$. It follows that

$$\log_m n - \ell \cdot \log_m \log_m n - S(n) \notin O(1) \Rightarrow 1 \notin o(S(n)).$$

Let $S(i,n)$ be the number of distinct worktape squares visited through the i^{th} time that M scans an input endmarker on either end of the input while computing on input 1^n, and let

$$Q(i,n) \in \{¢,\$\} \times \{1,\ldots,d\} \times \{1,\ldots,m\}^{S(n)} \times \{1,\ldots,S(n)\}^{\ell}$$

describe the total state of M at the end of that time. (If there is no i^{th} endmarked total state, then $Q(i,n)$ = undefined.)

For $\log_m - \ell \cdot \log_m \log_m n - S(n) > \log_m d$, we show by induction on i that $S(i,n) = S(i, n + kn!)$ and $Q(i,n) = Q(i, n + kn!)$. That $S(i,n) = S(i, n + kn!)$ for all i implies $S(n) = S(n + kn!)$.

Since M has a fixed initial state, $Q(1,n) = Q(1, n + kn!)$ and $S(1,n) = S(1, n + kn!) = 1$.

Suppose $Q(i,n) = Q(i, n + kn!)$ and $S(i,n) = S(i, n + kn!)$. To prove $Q(i+1,n) = Q(i+1, n + kn!)$ and $S(i+1,n) = S(i+1, n + kn!)$, we consider four cases:

Case 1: $Q(i,n)$ = undefined. Obviously,

$Q(i+1,n) = Q(i+1, n + kn!)$ = undefined,

$S(i+1,n) = S(i,n) = S(i, n + kn!) = S(i+1, n + kn!)$.

Case 2: $Q(i,n)$ is defined, but $Q(i+1,n)$ = undefined. Since $n + kn! \geq n$, the computation continuations are identical.

Case 3: $Q(i+1,n)$ is defined and involves the same endmarker as $Q(i,n)$. Since $n + kn! \geq n$, the computations are identical from the i^{th} endmarked total state up to the $(i+1)^{st}$ endmarked total state.

Case 4: $Q(i+1,n)$ is defined and involves the other endmarker. In going from $Q(i,n)$ to $Q(i+1,n)$, M first reaches each input square $j \in \{1,\ldots,n\}$ in some memory state $f(j)$. Because

$$d \cdot m^{S(n)} \cdot S(n)^{\ell} < n$$

is implied by

$$\log_m n - \ell \cdot \log_m \log_m n - S(n) > \log_m d,$$

however, $f(j_1) = f(j_2)$ for some $j_1 < j_2 \in \{1,\ldots,n\}$. Clearly, therefore, increasing the input length by any multiple of $j_2 - j_1$ results in the same next endmarked total state and no new memory states. Certainly kn! is a multiple of $j_2 - j_1$. □

Criteria slightly different from ours for "acceptance within space S" have been proposed. Book [Bk72] requires that _every accepting computation_ on input x involve no more than $S(|x|)$ worktape squares, and Ibarra [Ib72] requires that _every_ computation on input x involve no more than $S(|x|)$ worktape squares. The significance of the proof of part (iii) of Proposition 7 is that the complexity classes determined by fully constructable space bounds are hardly affected by these differences.

While part (iv) is surprising, part (v) of Proposition 7 redeems our intuition that a radix count of the input length fully constructs nearly the smallest possible fully constructable space bound. The result implies that one may substitute the innocuous hypothesis $1 \in o(S(n))$ whenever the apparently more arbitrary condition $\log n \in O(S(n))$ arises for a fully constructable space bound S.

Hopcroft and Ullman [HU69a] work with space bounds that are merely "constructable."

Definition. If M is a deterministic off-line TM acceptor with

$$S(n) = \max\{\mathrm{Space}_M(x) \mid x \in L(M),\ |x| = n\},$$

then S is _constructable_ and M _constructs_ S.

An interesting corollary of Proposition 7(v) and the existence of constructable space bounds $S(n) \in o(\log n)$ with $1 \in o(S(n))$ [SHL65] is that not every constructable space bound is fully constructable. Because we cannot prove Proposition 7(iii) for constructable space bounds, however, we choose not to use the concept.

4. Conventional separation results

Counting and diagonalization arguments ([HU69a] and [SHL65], respectively) have been used to prove separation results among the NSPACE(S) and DSPACE(S) complexity classes. (See Corollaries 9, 11 below.) In this section we sketch more careful versions of these arguments to show what conditions they yield for separation among the more refined classes NSPACE(S,m,ℓ) and DSPACE(S,m,ℓ). For details, the reader is referred to [HU69a], [SHL65], [HU69b].

Theorem 8. If S_2 is fully constructable by an (m,ℓ)-machine, then there is a language over $\{0,1\}$ in

$$\begin{aligned}&\mathrm{DSPACE}(S_2,m,\ell+1)\\ &\quad - (\bigcup\{\mathrm{NSPACE}(S_1,m,\ell) \mid S_2'(n)-2\cdot S_1(n) \notin O(1)\} \cup\\ &\qquad \bigcup\{\mathrm{DSPACE}(S_1,m,\ell-1) \mid S_2'(n)-S_1(n) \notin O(1)\}),\end{aligned}$$

where $S_2'(n) = \min\{S_2(n), \log_m n - \ell\cdot\log_m \log_m n\}$.

Proof sketch. Define

$$L = \{x \mid x = uyu^R \in \{0,1\}^* \text{ for some } u \text{ with}$$
$$|u| = \min\{\lfloor |x|/2 \rfloor, m^{S_2(|x|)}\cdot S_2(|x|)^{\ell}\}\},$$

where u^R is the reverse of string u. (Note here that

$$|u| \le m^{S_2'(|x|)}\cdot S_2'(|x|)^{\ell}.)$$

We first show that $L \in \mathrm{DSPACE}(S_2,m,\ell+1)$. An acceptor for L can first lay out space $S_2(|x|)$, using the extra head to delimit that space. The delimited space can be used as a counter to compare successive characters from the two ends of the input string. Because ℓ extra heads are available, the counter is large enough to count up to $m^{S_2(|x|)}\cdot S_2(|x|)^{\ell}$.

Next, we show that L $\notin$ NSPACE(S_1,m,ℓ) unless $S_2'(n)-2\cdot S_1(n) \in O(1)$. By the reasoning of [HU69a], for a d-state (m,ℓ)-machine to accept L within space S_1, we must have

$$2^{m^{S_2'(n)}\cdot S_2'(n)^{\ell}} \le 4^{(d\cdot m^{S_1(n)}\cdot S_1(n)^{\ell})^2}.$$

Taking logarithms twice gives

$$S_2'(n) + \ell\cdot\log S_2'(n) + cnstnt_1 \le 2\cdot(S_1(n) + \ell\cdot\log S_1(n)) + cnstnt_2,$$

so that $S_2'(n)-2\cdot S_1(n) \in O(1)$.

Finally, we use similar reasoning to show that L $\notin$ DSPACE(S_1,m,ℓ-1) unless $S_2'(n)-S_1(n) \in O(1)$. The behavior of a <u>deterministic</u> off-line TM at an input boundary can be described as a <u>function</u> from the storage states into the storage states plus the "accept" and "nonaccepting no-return" outcomes. For a deterministic d-state (m,ℓ-1)-machine to accept L within space S_1, therefore, we must have

$$2^{m^{S_2'(n)}\cdot S_2'(n)^{\ell}} \le (2 + d\cdot m^{S_1(n)}\cdot S_1(n)^{\ell-1})^{d\cdot m^{S_1(n)}\cdot S_1(n)^{\ell-1}}.$$

Taking logarithms twice gives

$$S_2'(n) + \ell\cdot\log S_2'(n) + cnstnt_1 \le S_1(n) + \ell\cdot\log S_1(n) + cnstnt_2,$$

so that $S_2'(n)-S_1(n) \in O(1)$. □

<u>Corollary 9</u> [HU69a]. If S_2 is fully constructable, then

$$\mathrm{DSPACE}(S_2) - \bigcup\{\mathrm{NSPACE}(S_1) \mid \min\{S_2(n), \log n\} \notin O(S_1(n))\}$$

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

<u>Proof</u>. Take m so large that S_2 is fully constructable by an (m,1)-machine. For $S_2'(n) = \min\{S_2(n), \log_m n - \log_m \log_m n\}$, Theorem 8 gives a language over $\{0,1\}$ that bears witness to the noncontainment in

$\bigcup \{\text{NSPACE}(S_1) \mid \min\{S_2(n), \log n\} \notin O(S_1(n))\}$

$= \bigcup \{\text{NSPACE}(S_1) \mid S_2' \notin O(S_1)\}$

$= \bigcup \{\text{NSPACE}(k \cdot S_1, m, 1) \mid k \in N,\ S_2' \notin O(S_1)\}$

$= \bigcup \{\text{NSPACE}(S_1, m, 1) \mid S_2' \notin O(S_1)\}$

$\subset \bigcup \{\text{NSPACE}(S_1, m, 1) \mid S_2'(n) - 2 \cdot S_1(n) \notin O(1)\}$

$\not\supset \text{DSPACE}(S_2, m, 2)$

$\subset \text{DSPACE}(S_2)$. $\square$

Theorem 10. If S_2 is fully constructable by an $(m, \ell+2)$-machine, then there is a language over $\{0,1\}$ in

$$\text{DSPACE}(S_2, m, \ell+3) - \bigcup \{\text{DSPACE}(S_1, m, \ell) \mid S_2(n) - 2 \cdot S_1(n) - \ell \cdot \log_m S_1(n) - \log_m n \notin O(1)\}.$$

Proof sketch. Design a deterministic $(m, \ell+3)$-machine M to operate as follows on input ex, where e is the description of a deterministic (m, ℓ)-machine M_e:

1. Lay out space $S_2(|ex|)$, using head A to bound it.
2. Use ℓ worktape heads to carry out a simulation of M_e on input ex, using head B to bound the simulation. (The simulation requires no more space than $c_e + S_{M_e}(ex)$, where c_e depends only on e; and it requires no additional worktape heads to read the description of M_e because that description can be carried around and read by one of the ℓ heads already being used.) Meanwhile, use head A to keep m-ary count of the simulated steps, and use head C to mark the high-order end of the counter.

$S_2(|ex|)$

simulation (ℓ heads)	free space	m-ary counter	blanks•••
	↑ head B	↑ head C	↑ head A

3. If the simulation is completed before the simulation and the count run out of free space, then complement the outcome of the simulation; otherwise, just accept.

Now suppose M_e is a deterministic d-state (m,ℓ)-machine that accepts within space S_1, where $\Delta(n) \notin O(1)$ for

$$\Delta(n) = S_2(n) - 2 \cdot S_1(n) - \ell \cdot \log_m S_1(n) - \log_m n.$$

Take x so that $\Delta(|ex|) > c_e + d$. If $ex \in L(M_e)$, then it must be accepted within $d \cdot |ex| \cdot m^{S_1(|ex|)} \cdot S_1(|ex|)^{\ell}$ steps by M_e. (Otherwise, a total state would repeat, and M_e would loop forever on ex.) Since

$$\begin{aligned}
&c_e + S_1(|ex|) + \log_m (d \cdot |ex| \cdot m^{S_1(|ex|)} \cdot S_1(|ex|)^{\ell}) \\
&= c_e + \log_m d + S_2(|ex|) - \Delta(|ex|) \\
&\le S_2(|ex|) - (\Delta(|ex|) - (c_e + d)) \\
&\le S_2(|ex|),
\end{aligned}$$

M does discover that $ex \in L(M_e)$ and arrange $ex \notin L(M)$. On the other hand, if $ex \notin L(M_e)$, then M will certainly accept ex. Therefore, $L(M_e) \neq L(M)$. □

<u>Corollary 11</u> [SHL65]. If S_2 is fully constructable, then

$$\mathrm{DSPACE}(S_2) - \bigcup \{\mathrm{DSPACE}(S_1) \mid S_2(n) \notin O(S_1(n) + \log n)\}$$

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

<u>Proof</u>. Take m so large that S_2 is fully constructable by an (m,1)-

machine. Theorem 10 gives a language over $\{0,1\}$ that bears witness to the noncontainment in

$$\bigcup \{\mathrm{DSPACE}(S_1) \mid S_2(n) \notin O(S_1(n) + \log n)\}$$
$$= \bigcup \{\mathrm{DSPACE}(k\cdot S_1, m, 1) \mid k \in N,\ S_2(n) \notin O(S_1(n) + \log n)\}$$
$$= \bigcup \{\mathrm{DSPACE}(S_1, m, 1) \mid S_2(n) \notin O(S_1(n) + \log n)\}$$
$$\subset \bigcup \{\mathrm{DSPACE}(S_1, m, 1) \mid S_2(n) - 2\cdot S_1(n) - \log_m S_1(n) - \log_m n \notin O(1)\}$$
$$\not\supset \mathrm{DSPACE}(S_2, m, 4)$$
$$\subset \mathrm{DSPACE}(S_2). \quad \square$$

By Proposition 6, one more corollary is implicit in Corollary 11.

Corollary 12. If S_2 is fully constructable, then

$$\mathrm{DSPACE}(S_2) - \bigcup \{\mathrm{NSPACE}(S_1) \mid S_2(n) \notin O(S_1(n)^2 + (\log n)^2)\}$$

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

Remark. The original arguments of [HU69a] and [SHL65] were for S_2 merely constructable. For S_2 constructable by an (m,ℓ)-machine M with $L(M) \subset \Sigma^*$, accordingly, a witness language over $\Sigma \times \{0,1\}$ is obtained in each of the above results.

5. Padding whole languages

For space bounds above log n, the separation results given by Corollary 11 are very good. For S_2 fully constructable and $\log n \in o(S_2(n))$, in fact, since $S_2 \in O(S_1)$ implies $\mathrm{DSPACE}(S_2) \subset \mathrm{DSPACE}(S_1)$ (Proposition 3), it follows from Corollary 11 that $\mathrm{DSPACE}(S_2) - \mathrm{DSPACE}(S_1) \neq \emptyset$ if and only if $S_2 \notin O(S_1)$. Corollary 12, on the other hand, is relatively weak and does not even separate $\mathrm{NSPACE}(n^4)$ from $\mathrm{NSPACE}(n^2)$, for example.

Using the padding trick of Ruby and P. Fischer [RF65], Ibarra [Ib72] has refined some of the separation results given by Corollary 12. The basic trick is illustrated by the following lemma, where we write p(L) for $\{x10^k \mid x \in L, |x10^k| = p(|x|)\}$ when $L \subset \{0,1\}^*$ and $p:N \to N$ satisfies $p(n) > n$.

Lemma 13. If $p(n) > n$ is linear space honest and $\log n \in O(S(n))$, then

$$p(L) \in \mathrm{NSPACE}(S) \Leftrightarrow L \in \mathrm{NSPACE}(S \circ p).$$

Proof. Every language accepted deterministically within space n can actually be recognized (cf., footnote on page 43) within space n [HU69a], so let M deterministically recognize $\{\mathrm{bin}(k)\#\mathrm{bin}(f(k)) \mid k \in N\}$ within space n.

($\Rightarrow$) Suppose M_1 accepts p(L) within space S. Design M_2 to operate as follows on input string x:

1. Write down $\mathrm{bin}(|x|)$ and then compute according to the transition rules of M successively on

   $$\mathrm{bin}(|x|)\#\mathrm{bin}(0),\ \mathrm{bin}(|x|)\#\mathrm{bin}(1),\ \mathrm{bin}(|x|)\#\mathrm{bin}(2),\ldots$$

   until $\mathrm{bin}(p(|x|))$ is discovered. As $p(n) \geq n$, this can be accomplished within space proportional to $\log p(|x|) \in O(S(p(|x|)))$.

2. Compute according to the transition rules of M_1 on input $x10^{p(|x|)-|x|-1}$. By hypothesis, this can be accomplished within space proportional to $S(p(|x|))$ in acceptance.

Clearly, M_2 accepts L within space proportional to $S \circ p$.

($\Leftarrow$) Suppose M_2 accepts L within space $S(p(n))$. Design M_1 to operate as follows on input string $x10^k$:

1. Write down $\mathrm{bin}(|x|)\#\mathrm{bin}(|x|+1+k)$ and then compute according to the transition rules of M to determine whether $|x10^k| = p(|x|)$. This can be accomplished within space proportional to $\log p(|x|) = \log |x10^k| \in O(S(|x10^k|))$ in acceptance.
2. If $|x10^k| = p(|x|)$, then compute according to the transition rules of M_2 on input x. By hypothesis, this can be accomplished within space $S(p(|x|)) = S(|x10^k|)$ in acceptance.

Clearly, M_1 accepts p(L) within space proportional to S. $\square$

The following theorem shows how Lemma 13 is used to improve known separation results. The formulation is a variation of Ibarra's [Ib72].

Theorem 14. Let sets $\mathcal{S}_1$, $\mathcal{S}_2$ of space bounds be given, with $\log n \in O(S(n))$ for every $S \in \mathcal{S}_1 \cup \mathcal{S}_2$. Say $p_1(n) > n, \ldots, p_\ell(n) > n$ are linear space honest functions with $S_1 \circ p_{i+1} \in O(S_2 \circ p_i)$ whenever $1 \le i < \ell$, $S_1 \in \mathcal{S}_1$, $S_2 \in \mathcal{S}_2$.
If $L \in \bigcap \{\mathrm{NSPACE}(S_2 \circ p_\ell) |\ S_2 \in \mathcal{S}_2\} - \bigcup \{\mathrm{NSPACE}(S_1 \circ p_1) |\ S_1 \in \mathcal{S}_1\}$,
then $p_i(L) \in \bigcap \{\mathrm{NSPACE}(S_2) |\ S_2 \in \mathcal{S}_2\} - \bigcup \{\mathrm{NSPACE}(S_1) |\ S_1 \in \mathcal{S}_1\}$ for some i.

Proof. For $1 \le i \le \ell$, let

$$C(i,1) = \bigcup \{NSPACE(S_1 \circ p_i) \mid S_1 \in \mathcal{S}_1\},$$

$$C(i,2) = \bigcap \{NSPACE(S_2 \circ p_i) \mid S_2 \in \mathcal{S}_2\}.$$

Suppose $L \in C(\ell,2) - C(1,1)$. By Proposition 3, $NSPACE(S_1 \circ p_{i+1}) \subset NSPACE(S_2 \circ p_i)$ whenever $1 \le i < \ell$, $S_1 \in \mathcal{S}_1$, $S_2 \in \mathcal{S}_2$; so, for $1 \le i < \ell$,

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

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 $L \in C(\ell,2)$ that $L \in C(1,1)$, a contradiction. For some i, therefore, we must have

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

$$= \bigcap \{NSPACE(S_2 \circ p_i) \mid S_2 \in \mathcal{S}_2\} - \bigcup \{NSPACE(S_1 \circ p_i) \mid S_1 \in \mathcal{S}_1\}.$$

By Lemma 13,

$$p_i(L) \in \bigcap \{NSPACE(S_2) \mid S_2 \in \mathcal{S}_2\} - \bigcup \{NSPACE(S_1) \mid S_1 \in \mathcal{S}_1\}$$

for that same i. $\square$

Examples [Ib72]. $NSPACE((\log n)^{1.99}) \subsetneq NSPACE((\log n)^2)$.

$$NSPACE(n^{1.99}) \subsetneq NSPACE(n^2).$$

$$NSPACE((1.99)^n) \subsetneq NSPACE(2^n).$$

The main result of this chapter, Theorem 18 below, is a general separation result which not only subsumes and improves on most of the specific results of [Ib72], but which improves on diagonalization even for DSPACE in many cases. The method of proof is that of Theorem 13 of Chapter Two of this thesis. That proof was inspired by [Se73], which was inspired in turn by [Ib72].

6. Program codes and recursion

For precision, let us now choose an appropriate program coding for off-line TMs. With each off-line TM having input alphabet $\{0,1\}$, we associate a distinct program code from $\{0,1\}^*$; and we do this in agreement with the easily-satisfied conditions listed below. We use the notation $L^{m,\ell}_{p.c.}$ for the set of program codes for (m,ℓ)-machines and $L_{p.c.}$ for the set of all program codes. We denote by M_e the off-line TM with program code e.

Condition 1. No program code is a prefix or suffix of another, and $L^{m,\ell}_{p.c.}$ is regular for each m, ℓ.

Condition 2. For each fixed m, ℓ, there is an (m,ℓ)-machine U_0 (a "universal simulator") with

$$L(U_0) = \{ex \mid e \in L^{m,\ell}_{p.c.},\ x \in L(M_e)\},$$

$$\mathrm{Space}_{U_0}(ex) \le c_e + \mathrm{Space}_{M_e}(x) \text{ if } e \in L^{m,\ell}_{p.c.},$$

where c_e depends only on e. Furthermore, U_0 has only one computation on ex if M_e is deterministic.

Condition 3. There is a recursive function $f: L_{p.c.} \to L_{p.c.}$ such that $f: L^{m,\ell}_{p.c.} \to L^{m,\ell}_{p.c.}$ for each m, ℓ and such that $M_{f(e)}$ deterministically writes e at the front of its worktape and thereafter acts according to the transition rules of M_e.

Most common instruction-by-instruction or state-by-state codings of off-line TM programs can be tailored to satisfy these conditions. The only trick is to design the universal simulator of Condition 2 so that one of the $\ell \ge 1$ simulating worktape heads carries with it and references an appropriate version of the program code e. (A similar trick is used below in the proof of Theorem 15.)

Conditions 2 and 3 allow us to prove a version of the fixed point theorem (recursion theorem) of recursive function theory.

Theorem 15. For each (m,ℓ)-machine M with $L(M) \subseteq \{0,1\}^*$, there is an (m,ℓ)-machine M_{e_0} with

$$L(M_{e_0}) = \{x \mid e_0 x \in L(M)\},$$

$$\exists c(\mathrm{Space}_{M_{e_0}}(x) \leq c + \mathrm{Space}_M(e_0 x)).$$

Furthermore, M_{e_0} is deterministic if M is.

Proof. Let U_0 be the universal simulator of Condition 2 for m, ℓ, let f be as in Condition 3, and let e_2 be the program code for M. Take M_{e_1} to be an (m,ℓ)-machine that operates as follows given x on its input tape and e on its worktape:

1. Convert e to $f(e)$.
2. Convert $f(e)$ to $h(f(e))$, where h is the "doubling homomorphism" with $h(0) = 00$, $h(1) = 11$.
3. Simulate U_0 on $e_2 f(e)x$. To do this, commit e_2 to finite-state memory and carry $h(f(e))$ around with one of the worktape heads of U_0. The redundant symbols in $h(f(e))$ can be modified to mark the ends of the string and the position of the input head of U_0 when it is in that portion of the tape.

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

1. Write e_1 on the worktape.

2. Convert e_1 to $f(e_1) = e_0$.

3. Convert e_0 to $h(e_0)$.

4. Simulate U_0 on e_2e_0x.

Thus,

$$x \in L(M_{e_0}) \Leftrightarrow e_2e_0x \in L(U_0)$$

$$\Leftrightarrow e_0x \in L(M),$$

$$\mathrm{Space}_{M_{e_0}}(x) \le c + \mathrm{Space}_M(e_0x),$$

where c is c_{e_2} plus the number of worktape squares required for steps 1, 2, 3. □

7. Another general separation result

The general separation result that we prove in this section (Theorem 18) amounts to a dramatic refinement of the following very weak "separation" result.

Lemma 16. For no recursive space bound S does NSPACE(S) contain all the recursive languages over $\{1\}$.

Proof. If S is recursive, then the diagonal language

$$\{1^n \mid 1^n \notin L_S(M_e),\ 1e = \text{bin}(n)\}$$

is a recursive language not in NSPACE(S). □

One more technical lemma is all we need for the proof of Theorem 18.

Lemma 17. If S/2 is fully constructable by a (2,1)-machine, then some deterministic (m,ℓ)-machine recognizes

$$L = \{1^j 0^k \mid k \ge m^{S(j)} \cdot S(j)^{\ell-1}\}$$

within space $\log_m n - (\ell-1) \cdot \log_m \log_m n$.

Proof. It is easier to get an $(m,\ell+1)$-machine that does the job. Given a fixed position s of the extra head, we can use the other ℓ heads within space s to count through $s^{\ell-1}$ cycles of an m-ary counter that counts up to m^s, while checking whether $k < m^s \cdot s^{\ell-1}$. By trying successive positions s, we can find position s_0, the least $s \ge 1$ with $k < m^s \cdot s^{\ell-1}$.

Certainly S is fully constructable by a (2,1)-machine. Since $m \ge 2$ and $\ell \ge 1$, we can leave the extra head at position s_0 and try to lay out space S(j) without reaching that position. We succeed if and only if $k \ge m^{S(j)} \cdot S(j)^{\ell-1}$.

Suppose n is so large that

$$(\log_m n - (\ell-1)\cdot\log_m \log_m n)^{\ell-1} > (\log_m n)^{\ell-1}/m.$$

If

$$s_0 > 2 + \log_m n - (\ell-1)\cdot\log_m \log_m n,$$

then

$$m^{s_0-1}\cdot(s_0-1)^{\ell-1} > n = j + k \geq k$$

by substitution, contradicting the minimality of s_0. Therefore,

$$s_0 \leq 2 + \log_m n - (\ell-1)\cdot\log_m \log_m n$$

for all sufficiently large n, and the simple method of Proposition 2 yields an $(m,\ell+1)$-machine that recognizes L within space $\log_m n - (\ell-1)\cdot\log_m \log_m n$.

To get rid of the extra head, we make two observations. The first is that the boundary head can be used in the first phase to run the m-ary counter without losing its place. The second is that the boundary head can be used even to lay out space S(j) in the second phase. The reason is that, since S/2 is fully constructable by a (2,1)-machine, S is fully constructable by a (2,1)-machine that leaves every other tape square redundant by always writing and reading aa rather than just a for every worktape symbol a. We can modify redundant tape squares to mark the head position and the ends of the used space. □

Theorem 18. If S_2 is fully constructable by an (m,ℓ)-machine, then each of the following set differences contains a language over $\{0,1\}$:

$$\text{NSPACE}(S_2,m,\ell+3) - \bigcup\{\text{NSPACE}(S_1,m,\ell+2) \mid 1 \in o(S_2(n)-S_1(n+1))\},$$

$$\text{DSPACE}(S_2,m,\ell+3) - \bigcup\{\text{DSPACE}(S_1,m,\ell+2) \mid 1 \in o(S_2(n)-S_1(n+1))\}.$$

Proof. Let S_2 be fully constructable by an (m,ℓ)-machine, and let U_0 be the universal simulator of Condition 2 for m, $\ell+2$. By Proposition 7(iii), $L_{S_2}(U_0) \in NSPACE(S_2, m, \ell+3)$, and intuitively it should be very nearly the hardest language in that class. We prove that $L_{S_2}(U_0) \notin NSPACE(S_1, m, \ell+2)$ for any space bound S_1 with $1 \in o(S_2(n)-S_1(n+1))$.

Suppose that the $(m,\ell+2)$-machine M_1 accepts $L_{S_2}(U_0)$ within space S_1, where $1 \in o(S_2(n)-S_1(n+1))$. Then $1 \in o(S_2(n))$, so that

$$\log_m n - \ell\cdot\log_m \log_m n - S_2(n) \in O(1)$$

by Proposition 7(v). Therefore, it is no loss of generality to assume also that

$$S_1(n) \geq \log_m n - (\ell+1)\cdot\log_m \log_m n.$$

To summarize the behavior of M_1,

$$L(M_1) = L_{S_2}(U_0) \subseteq L(U_0)$$

and, for $e \in L^{m,\ell+2}_{p.c.}$,

$$c_e + Space_M(x) \leq S_2(|ex|) \Rightarrow Space_{M_1}(ex) \geq S_1(|ex|).$$

Let $L \subset \{1\}^*$ be any recursive language over $\{1\}$. Because L is recursive, we can take a deterministic $(2,1)$-machine that accepts L within some space bound $S/2$ that is fully constructable by a $(2,1)$-machine. Let $S'(n) = m^{S(n)}\cdot S(n)^{\ell+1}$, and design an $(m,\ell+2)$-machine M' that operates as follows:

1. Check that the input string is of the form $ex0^k$ for some $e \in L^{m,\ell+2}_{p.c.}$, some $x \in \{1\}^*$. By Condition 1 this requires no space.

2. Determine whether $k \geq S'(|x|)$. By Lemma 17 this requires no more than

$$\log_m |x0^k| - (\ell+1)\cdot\log_m \log_m |x0^k|$$
$$\le \log_m |ex0^{k+1}| - (\ell+1)\cdot\log_m \log_m |ex0^{k+1}|$$
$$\le S_1(|ex0^{k+1}|)$$

worktape squares.

3. If $k \ge S'(|x|)$, then compute on input x according to the transition rules of M. This requires no more than

$$S(|x|) \le \log_m S'(|x|) - (\ell+1)\cdot\log_m \log_m S'(|x|)$$
$$\le \log_m k - (\ell+1)\cdot\log_m \log_m k$$
$$\le \log_m |ex0^{k+1}| - (\ell+1)\cdot\log_m \log_m |ex0^{k+1}|$$
$$\le S_1(|ex0^{k+1}|)$$

worktape squares for $x \in L(M)$.

4. If $k < S'(|x|)$, then compute on input $ex0^{k+1}$ according to the transition rules of U_1, committing the final 0 to finite-state memory. This requires no more than $S_1(|ex0^{k+1}|)$ worktape squares for acceptance.

To summarize the behavior of M' on $ex0^k$,

$k \ge S'(|x|) \Rightarrow$ behave like M on x;

$k < S'(|x|) \Rightarrow$ behave like U_1 on $ex0^{k+1}$.

To summarize the timing for $ex0^k \in L(M')$,

$$\text{Space}_{M'}(ex0^k) \le S_1(|ex0^{k+1}|).$$

Applying the recursion theorem (Theorem 15) to M', we get a program code e_0 for an $(m,\ell+2)$-machine that accepts

$$L(M_{e_0}) = \{x0^k |\ e_0x0^k \in L(M')\} \subset 1^*0^*$$

within space

$$\text{Space}_{M_{e_0}}(x0^k) \le c + \text{Space}_{M'}(e_0x0^k)$$

for some constant c.

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

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

Proof. Let $x \in \{1\}^*$ be so long that

$$S_2(n)-S_1(n+1) \geq c_{e_0} + c$$

for every $n \geq |e_0x|$. We establish the claim for x by induction on k running down from $k \geq S'(|x|)$ to $k = 0$.

$k \geq S'(|x|)$:

$$x0^k \in L(M_{e_0}) \Leftrightarrow e_0x0^k \in L(M')$$

(by choice of e_0)

$$\Leftrightarrow x \in L(M) = L$$

(because by definition M' behaves like M in this case).

$k < S'(|x|)$: Assume $x0^{k+1} \in L(M_{e_0}) \Leftrightarrow x \in L$.

$$x0^k \in L(M_{e_0}) \Leftrightarrow e_0x0^k \in L(M')$$

(by choice of e_0)

$$\Leftrightarrow e_0x0^{k+1} \in L(U_1) \subset L(U_0)$$

(because by definition M' behaves like U_1 in this case)

$$\Leftrightarrow x0^{k+1} \in L(M_{e_0})$$

(because U_0 is universal over $L^{m,\ell+2}_{p.c.}$)

$$\Leftrightarrow x \in L$$

$$x \in L \Rightarrow x0^{k+1} \in L(M_{e_0})$$

(by induction hypothesis)

$\Rightarrow e_0x0^{k+1} \in L(M')$

(by choice of e_0)

$\Rightarrow \mathrm{Space}_{M'}(e_0x0^{k+1}) \leq S_1(|e_0x0^{k+2}|)$

(by space usage of M')

$\Rightarrow c_{e_0} + \mathrm{Space}_{M_{e_0}}(x0^{k+1}) \leq c_{e_0} + c + \mathrm{Space}_{M'}(e_0x0^{k+1})$

(by choice of e_0, c)

$\leq c_{e_0} + c + S_1(|e_0x0^{k+2}|)$

$\leq S_2(|e_0x0^{k+1}|)$

(because x is so long)

$\Rightarrow e_0x0^{k+1} \in L_{S_2}(U_0) = L(U_1)$

(by choice of U_0)

$\Rightarrow e_0x0^k \in L(M')$

(because by definition M' behaves like U_1 in this case)

$\Rightarrow x0^k \in L(M_{e_0})$

(by choice of e_0). □

Finally, M_{e_0} can be modified to use its finite-state control to reject padded inputs (those not members of $\{1\}^*$) and to agree with M on short ones (those not sufficiently long for the claim) without using the worktape. This gives an off-line TM that accepts L = L(M) within space $S_1(|e_0|+n+1)$. Because

$$S_1(|e_0|+n+1) \in O(S_2(|e_0|+n)) \subset O(\sum_{n'\leq 2n} S_2(n')),$$

Proposition 3 gives $L \in \mathrm{NSPACE}(\sum_{n' \leq 2n} S_2(n'))$. But $L \subseteq \{1\}^*$ was chosen arbitrarily, and by Lemma 16 not every such recursive $L \subset \{1\}^*$ can belong to the particular class $\mathrm{NSPACE}(\sum_{n' \leq 2n} S_2(n'))$, a contradiction.

A similar proof shows that

$$L_{S_2}(U_0') \in \mathrm{DSPACE}(S_2, m, \ell+3)$$

$$- \bigcup \{\mathrm{DSPACE}(S_1, m, \ell+2) \mid 1 \in o(S_2(n) - S_1(n+1))\},$$

where U_0' computes according to the transition rules of U_0 in deterministic computations but halts without accepting whenever U_0 would take a nondeterministic step. $\square$

8. Applications of the general separation results

Our first application brings together Corollaries 9, 11, and 12 with the related consequences of Theorem 18. It is the latter (part (iii) of Corollary 19) that subsume and improve on the specific results of [Ib72].

Corollary 19. Let S_2 be fully constructable.

(i) If $S_2(n) \in O(\log n)$, then $\mathrm{DSPACE}(S_2) \not\subset \bigcup \{\mathrm{NSPACE}(S_1) \mid S_2 \notin O(S_1)\}$.

(ii) $\mathrm{DSPACE}(S_2) \not\subset \bigcup \{\mathrm{DSPACE}(S_1) \mid S_2 \notin O(S_1)\}$,

$\mathrm{DSPACE}(S_2) \not\subset \bigcup \{\mathrm{NSPACE}(S_1) \mid S_2 \notin O(S_1^{\,2})\}$.

(iii) $\mathrm{NSPACE}(S_2) \not\subset \bigcup \{\mathrm{NSPACE}(S_1) \mid S_1(n+1) \in o(S_2(n))\}$,

$\bigcup \{\mathrm{NSPACE}(S_1) \mid S_1(n+1) \in O(S_2(n)),\ S_1(n) \in o(S_2(n))\} \subsetneqq \mathrm{NSPACE}(S_2)$.

(In the case $S_2(n+1) \in O(S_2(n))$, note that $S_1(n+1) \in O(S_2(n))$ is implied by $S_1(n) \in o(S_2(n))$.)

Furthermore, there are languages over $\{0,1\}$ that bear witness to these facts.

Proof. (i) This part is a slightly weakened form of Corollary 9.

(ii) By Corollaries 9, 11, and 12, we can take languages $L_{00}, L_{01}, L_{10} \subset \{0,1\}^*$ such that

$$L_{00} \in \mathrm{DSPACE}(S_2(n+2)) - \bigcup \{\mathrm{NSPACE}(S_1(n+2)) \mid \min\{S_2(n), \log n\} \notin O(S_1(n))\},$$

$$L_{01} \in \mathrm{DSPACE}(S_2(n+2)) - \bigcup \{\mathrm{DSPACE}(S_1(n+2)) \mid S_2(n) \notin O(S_1(n) + \log n)\},$$

$$L_{10} \in \mathrm{DSPACE}(S_2(n+2)) - \bigcup \{\mathrm{NSPACE}(S_1(n+2)) \mid S_2(n) \notin O(S_1(n)^2 + (\log n)^2)\}.$$

Clearly the language

$$\{00x \mid x \in L_{00}\} \cup \{01x \mid x \in L_{01}\} \cup \{10x \mid x \in L_{10}\}$$

belongs to DSPACE($S_2(n)$) but not to

$$\bigcup \{\mathrm{NSPACE}(S_1(n)) \mid \min\{S_2(n), \log n\} \notin O(S_1(n))\}, \text{ or}$$

$$\bigcup \{\mathrm{DSPACE}(S_1(n)) \mid S_2(n) \notin O(S_1(n) + \log n)\}, \text{ or}$$

$$\bigcup \{\mathrm{NSPACE}(S_1(n)) \mid S_2(n) \notin O(S_1(n)^2 + (\log n)^2)\}.$$

(Cf., Lemma 13.) It is easy to verify that

$$S_2 \notin O(S_1) \Rightarrow \min\{S_2(n), \log n\} \notin O(S_1(n)) \vee S_2(n) \notin O(S_1(n) + \log n),$$

$$S_2 \notin O(S_1^{\,2}) \Rightarrow \min\{S_2(n), \log n\} \notin O(S_1(n)) \vee S_2(n) \notin O(S_1(n)^2 + (\log n)^2).$$

(iii) Take m so large that S_2 is fully constructable by an (m,ℓ)-machine. Theorem 18 gives a language over $\{0,1\}$ that bears witness to the noncontainment in

$$\begin{aligned} &\bigcup \{\mathrm{NSPACE}(S_1) \mid S_1(n+1) \in o(S_2(n))\} \\ &= \bigcup \{\mathrm{NSPACE}(k \cdot S_1, m, 3) \mid k \in N,\ S_1(n+1) \in o(S_2(n))\} \\ &= \bigcup \{\mathrm{NSPACE}(S_1, m, 3) \mid S_1(n+1) \in o(S_2(n))\} \\ &\subset \bigcup \{\mathrm{NSPACE}(S_1, m, 3) \mid 1 \in o(S_2(n) - S_1(n+1))\} \\ &\not\supset \mathrm{NSPACE}(S_2, m, 4) \\ &\subset \mathrm{NSPACE}(S_2). \end{aligned}$$

Containment in the second assertion of part (iii) holds by Proposition 3. To prove that the containment is proper, appeal to the first assertion to get a language $L \subset \{0,1\}^*$ in

$$\mathrm{NSPACE}(S_2(n+2)) - \bigcup \{\mathrm{NSPACE}(S_1(n+1)) \mid S_1(n+1) \in O(S_2(n)),\ S_1(n) \in o(S_2(n))\}$$

and then apply Theorem 14 with

$\mathcal{S}_1 = \{S_1 | \ S_1(n+1) \in O(S_2(n)),\ S_1(n) \in o(S_2(n))\}$,

$\mathcal{S}_2 = \{S_2\}$,

$p_1(n) = n + 1$,

$p_2(n) = n + 2$. □

Examples. For any function G tending to infinity ($1 \in o(G(n))$), however slowly,

$$\mathrm{NSPACE}(\log n/G(n)) \subsetneq \mathrm{NSPACE}(\log n),$$
$$\mathrm{NSPACE}((\log n)^2/G(n)) \subsetneq \mathrm{NSPACE}((\log n)^2),$$
$$\mathrm{NSPACE}(n^2/G(n)) \subsetneq \mathrm{NSPACE}(n^2),$$
$$\mathrm{NSPACE}(2^n/G(n)) \subsetneq \mathrm{NSPACE}(2^n),$$
$$\mathrm{NSPACE}(2^{2^n}) \subsetneq \mathrm{NSPACE}(2^{2^{n+1}}).$$

Feldman and Owings [FO73] have observed that deterministic linear bounded automata are more powerful than deterministic linear bounded automata with a fixed worktape alphabet; i. e.,

$$\mathrm{DSPACE}(n,m) \subsetneq \mathrm{DSPACE}(n).$$

Our next application generalizes that observation, showing, for example, that it holds even for nondeterministic linear bounded automata.

Corollary 20. Let S be fully constructable.

(i) If $S(n) \in o(\log n)$, then $\mathrm{NSPACE}(S,m) \not\supseteq \mathrm{DSPACE}(S)$.

(ii) $\mathrm{DSPACE}(S,m) \subsetneq \mathrm{DSPACE}(S)$.

(iii) If $S(n+1) \in O(S(n))$ and $1 \in o(S(n))$, then $\mathrm{NSPACE}(S,m) \subsetneq \mathrm{NSPACE}(S)$.

Furthermore, there are languages over $\{0,1\}$ that bear witness to these facts.

Proof. (i) Take m so large that S is fully constructable by an (m,1)-

machine. By Proposition 1, $S(n) \in o(\log(\log_2 n))$ so Theorem 8 gives a language over $\{0,1\}$ that bears witness to the noncontainment in

$$
\begin{aligned}
NSPACE(S,m) &\subset NSPACE(S,m+1,1)\\
&\not\supset DSPACE(3\cdot S,m+1,2)\\
&\subset DSPACE(S).
\end{aligned}
$$

(ii) Part (i) handles the case $S(n) \in o(\log n)$, so assume $S(n) \notin o(\log n)$. Take m so large that S is fully constructable by an (m,1)-machine and $S(n) - \log_m n \notin O(1)$. Appealing to Theorem 10, we get a language over $\{0,1\}$ that bears witness to the proper containment in

$$
\begin{aligned}
DSPACE(S,m) &\subset DSPACE(S,m+1,1)\\
&\subsetneq DSPACE(4\cdot S,m+1,4)\\
&\subset DSPACE(S).
\end{aligned}
$$

(iii) Take m so large that S is fully constructable by an (m,1)-machine. Because $S(n+1) \in O(S(n))$ and $1 \in o(S(n))$, we can take k so large that $1 \in o(k\cdot S(n)-S(n+1))$. Theorem 15 gives a language over $\{0,1\}$ that bears witness to the proper containment in

$$
\begin{aligned}
NSPACE(S,m) &\subset NSPACE(S,m+1,3)\\
&\subsetneq NSPACE(k\cdot S,m+1,4)\\
&\subset NSPACE(S). \ \square
\end{aligned}
$$

Examples. $NSPACE(\log_2 n, m) \subsetneq NSPACE(\log n)$,

$NSPACE(n,m) \subsetneq NSPACE(n)$,

$NSPACE(2^n,m) \subsetneq NSPACE(2^n)$.

Ibarra [Ib73a] has shown that $DHEADS(k) \subsetneq DHEADS(k+2)$, where DHEADS(k) (NHEADS(k), respectively) denotes the class of languages over $\{0,1\}$ accepted by deterministic (nondeterministic, respectively) two-way finite automata with $k \geq 1$ heads.[†] The fact

$$\mathrm{NSPACE}(\log_2 n, m) \subsetneq \mathrm{NSPACE}(\log n)$$

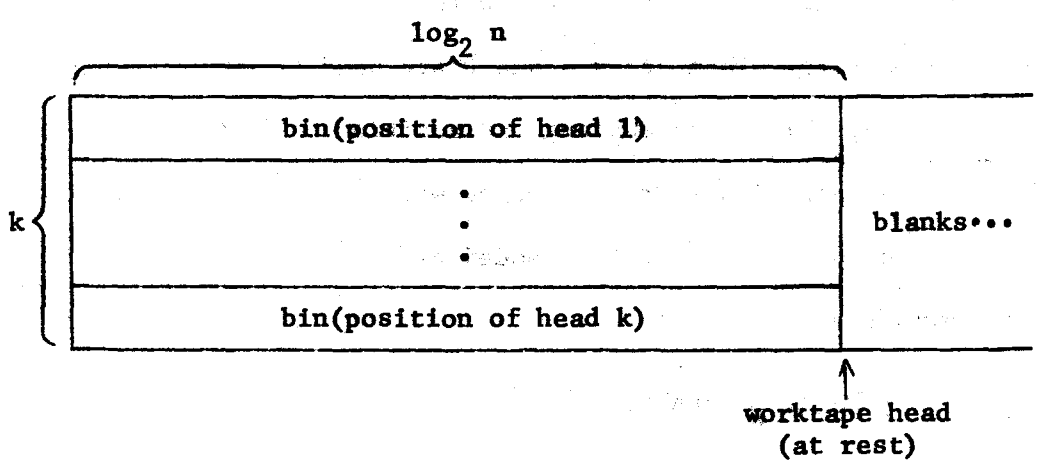
is just what is needed to establish a similar hierarchy theorem for NHEADS. To this end, we prove a lemma relating NHEADS to NSPACE(log n) and DHEADS to DSPACE(log n).

<u>Lemma 21</u> [Ha72].

$$\mathrm{NHEADS}(k) \subset \mathrm{NSPACE}(\log_2 n, 2^k, 1) \subset \mathrm{NHEADS}(k+4),$$

$$\mathrm{DHEADS}(k) \subset \mathrm{DSPACE}(\log_2 n, 2^k, 1) \subset \mathrm{DHEADS}(k+4).$$

<u>Proof</u>. Let M be a two-way finite automaton with k heads. A $(2^k,1)$-machine M' can simulate M by behaving like a "k-track" (2,1)-machine, using the respective tracks of its worktape to hold the binary representations of the positions of the k heads of M. Clearly, M' is deterministic if M is.



Let M' be a $(2^k,1)$-machine that accepts within space $\log_2 n$. To simulate M', a two-way finite automaton M can encode each of the k

[†]Two-way finite automata with k heads can be described as off-line TMs that do not use their worktapes but that have k two-way read-only <u>input</u> heads. We assume the k heads cannot detect each other, but nothing we say here actually depends on that convention.

"tracks" of the worktape of M' as a head position whose binary representation is the track contents. An additional head, head A, can keep count of the position of the worktape head of M'.

To read the contents of track i of the worktape square M' scans, M begins by positioning head B coincident with head A and head C coincident with the head whose position encodes the contents of track i. (This requires help from head D since the heads cannot detect each other.) Then heads C and D are used to successively halve the position that encodes track i (dropping remainders), while decrementing the position of head B once for each halving. The remainder of the last division before B reaches the endmarker is the contents of track i of the worktape square scanned by M'.

To change the contents of track i of the worktape square M' scans, M begins by positioning head B coincident with head A and head C on the first input tape square. Then heads C and D are used to successively double the position, while decrementing the position of head B once for each doubling. The position that is reached when B reaches the endmarker is the power of 2 which must be added or subtracted from the position encoding track i. □

Corollary 22. $\mathrm{NHEADS}(k) \subsetneq \mathrm{NHEADS}(k+2)$.

Proof. For each k, Corollary 20(iii) guarantees the existence of some k' with

$$\begin{aligned}\mathrm{NHEADS}(k) &\subset \mathrm{NSPACE}(\log_2 n,\ 2^k,\ 1)\\ &\subsetneq \mathrm{NSPACE}(\log_2 n,\ 2^{k'},\ 1)\\ &\subset \mathrm{NHEADS}(k'+4).\end{aligned}$$

According to [Ib73a] this is enough. $\square$

Recently, Ibarra and Sahni ([Ib73b], [IS73]) refined some specific instances of our Corollary 20 to show that a single additional worktape symbol sometimes helps. The following more directly proven corollary generalizes their results. Note that this is a case where the consequences of Theorem 18 are stronger than those of Theorem 10 for <u>deterministic</u> machines.

<u>Corollary 23</u>. Let S be fully constructable.

(i) If $S(n) \in o(\log n)$, then

$$\mathrm{DSPACE}(S,m) \subsetneq \mathrm{DSPACE}(S,m+1,1)$$

for all sufficiently large m.

(ii) If $S(n+1)-S(n) \in o(S(n))$, then

$$\mathrm{NSPACE}(S,m) \subsetneq \mathrm{NSPACE}(S,m+1,1),$$

$$\mathrm{DSPACE}(S,m) \subsetneq \mathrm{DSPACE}(S,m+1,1)$$

for all sufficiently large m. (If S is actually linear space honest and $\log n \in o(S(n))$, then $m = 2$ is sufficiently large.)

Furthermore, there are languages over $\{0,1\}$ that bear witness to these facts.

<u>Proof</u>. Take m so large that S is fully constructable by an (m,1)-machine. (If S is actually linear space honest and $\log n \in o(S(n))$, then $m = 2$ will do, according to Proposition 7(ii).) Take rational numbers δ_1, δ_2 with $1 < \delta_1 < \delta_2 < \log_m (m+1)$. (Note then that $\delta_2 \cdot S$ is fully constructable by an (m,1)-machine, too.)

(i) By Proposition 1, $S(n) \notin o(\log \log n)$, so Theorem 8 gives a language over $\{0,1\}$ that bears witness to the proper containment in

$$\begin{aligned}\mathrm{DSPACE}(S,m) &\subset \mathrm{DSPACE}(\delta_1\cdot S,m,1)\\ &\subsetneq \mathrm{DSPACE}(\delta_2\cdot S,m,3)\\ &\subset \mathrm{DSPACE}(S,m+1,1).\end{aligned}$$

(ii) From $S(n+1)-S(n) \in o(S(n))$ and $S(n) \notin O(1)$, it is easy to show that $1 \in o(S(n))$ and hence that $1 \in o(\delta_2\cdot S(n)-\delta_1\cdot S(n+1))$. Therefore, Theorem 18 gives a language over $\{0,1\}$ that bears witness to the proper containment in

$$\begin{aligned}\mathrm{NSPACE}(S,m) &\subset \mathrm{NSPACE}(\delta_1\cdot S,m,3)\\ &\subsetneq \mathrm{NSPACE}(\delta_2\cdot S,m,4)\\ &\subset \mathrm{NSPACE}(S,m+1,1).\end{aligned}$$

The proof for DSPACE is identical. □

Examples. Within each of the space bounds $(\log_2 n)^2$, $n^{1/2}$, $n^{3.2}\cdot\log\log n$, $2^{n^{1/2}}$, every additional worktape symbol increases the computing power of nondeterministic and deterministic off-line TMs.

Finally, we go one step further and show that additional worktape heads sometimes increase computing power.

Corollary 24. Let S be fully constructable.

(i) If $S(n) \in o(\log n)$, then

$$\mathrm{DSPACE}(S,m,\ell) \subsetneq \mathrm{DSPACE}(S,m)$$

for all sufficiently large m and all ℓ.

(ii) If $S(n+1)-S(n) \in O(\log S(n))$ and $1 \in o(S(n))$, then

$$\mathrm{NSPACE}(S,m,\ell) \subsetneq \mathrm{NSPACE}(S,m),$$

$$\mathrm{DSPACE}(S,m,\ell) \subsetneq \mathrm{DSPACE}(S,m)$$

for all sufficiently large m and all ℓ. (If S is actually linear space honest and $\log n \in o(S(n))$, then $m = 2$ is sufficiently large.)

Furthermore, there are languages over $\{0,1\}$ that bear witness to these facts.

Proof. Take m so large that S is fully constructable by an (m,1)-machine. (If S is actually linear space honest and $\log n \in o(S(n))$, then $m = 2$ will do, according to Proposition 7(ii).) Look at any ℓ.

(i) By Proposition 1, $S(n) \notin o(\log\log n)$, so Theorem 8 gives a language over $\{0,1\}$ that bears witness to the proper containment in

$$\begin{aligned}\mathrm{DSPACE}(S,m,\ell) &\subset \mathrm{DSPACE}(S - (1/2)\cdot\log_m S,\ m,\ \ell+4)\\ &\subsetneqq \mathrm{DSPACE}(S,m,\ell+6)\\ &\subset \mathrm{DSPACE}(S,m).\end{aligned}$$

(Directly adapting the proof of Theorem 8 would give

$$\mathrm{DSPACE}(S,m,\ell) \subsetneqq \mathrm{DSPACE}(S,m,\ell+3).)$$

(ii) Take $c > 0$ so large that $S(n+1)-S(n) \le c\cdot\log_m S(n)$. Since $1 \in o(S(n))$, Theorem 18 gives a language over $\{0,1\}$ that bears witness to the proper containment in

$$\begin{aligned}\mathrm{NSPACE}(S,m,\ell) &\subset \mathrm{NSPACE}(S - d\cdot\log_m S,\ m,\ \ell+\lfloor c\rfloor+4)\\ &\subsetneqq \mathrm{NSPACE}(S,\ m,\ \ell+\lfloor c\rfloor+5)\\ &\subset \mathrm{NSPACE}(S,m),\end{aligned}$$

where $c < d < \lfloor c\rfloor+1$. (Note that $\lfloor c\rfloor = 0$ if $1 \in O(1 - (S(n+1)-S(n))/\log_m S(n))$.) The proof for DSPACE is identical. $\square$

Examples. Within each of the space bounds $(\log_7 n)^2$, $n^{1/2}$, n, $n\cdot(\log_2 n)^{1/2}$, every five additional worktape heads increase the computing power of nondeterministic and deterministic off-line TMs. Some greater number of additional worktape heads increases computing power within space $n \cdot \log_2 n$.

9. Witness languages over a one-letter alphabet

The witness languages provided by the general separation results above are subsets of $\{0,1\}^*$. In this section we investigate conditions for the witness languages to be subsets of just $\{1\}^*$. For sublogarithmic space bounds (Theorem 8), we know of no such conditions; but both Theorem 10 and Theorem 18 can be modified to give languages over just $\{1\}$.

Theorem 25. If S_2 is fully constructable by an $(m,\ell+2)$-machine, then there is a language over $\{1\}$ in

$$\mathrm{DSPACE}(S_2,m,\ell+3) - \bigcup \{\mathrm{DSPACE}(S_1,m,\ell) \mid 1 \in o(S_2(n) - 2\cdot S_1(n) - \ell\cdot \log_m S_1(n) - \log_m n)\}$$

Proof sketch. To adapt the proof of Theorem 10 and get a diagonal language over just $\{1\}$, we must, in limited space, somehow obtain a description of an (m,ℓ)-machine to simulate on the input 1^n. Furthermore, each description must arise for infinitely many n. (We cannot get by with the condition of Theorem 10 because that would require each description to arise for a string of every sufficient length.) Because $S_2(n) \geq \log_m n$ in the nontrivial case, a suitable approach is to obtain the description from the m-ary representation of the input length (e. g., by dropping the low-order 0's). □

Theorem 26. Let $f(n) \in O(n) - O(1)$ be nondecreasing and linear space honest. If S_2 is fully constructable by an $(m,\ell+2)$-machine and $\log n \in o(S_2(n))$, then each of the following set differences contains a language over $\{1\}$:

$$\mathrm{NSPACE}(S_2,m,\ell+6) - \bigcup \{\mathrm{NSPACE}(S_1,m,\ell) \mid S_2(n) - S_1(n + f(n)) \geq 4\cdot\log_m n\},$$

$$\mathrm{DSPACE}(S_2,m,\ell+6) - \bigcup \{\mathrm{DSPACE}(S_1,m,\ell) \mid S_2(n) - S_1(n + f(n)) \geq 4\cdot\log_m n\}.$$

Proof. Let $f':N \to N$ be the strictly increasing function with range $\{n \mid f(n) > f(n-1)\}$. Define injections g, $h:\{0,1\}^* \to N$ by

$$\mathrm{bin}(g(x)) = 1x,$$

$$h(x0) = h(x) + f(h(x)),$$

$$h(x1) = f'(m^{g(x1)}) + m^{g(x1)} - 1,$$

$$h(\lambda) = f'(m) + m - 1.$$

Use the fact that $f(n) \in O(n)$ to take j so large that $h(x0) \leq j\cdot h(x)$. Note that, because f is linear space honest and $f(n) \in O(n)$, the conversion of $1^{h(x10^k)}$ to $x10^k$ (or, more conveniently, something like $x1\#\mathrm{bin}(k)$) can be accomplished within space proportional to log n.

Claim 1. The result of this conversion requires only space

$$\log_m n - G(n)$$

for some G with $1 \in o(G(n))$.

Proof. Because f is nondecreasing and unbounded, we can take some G_1 with $1 \in o(G_1(n))$ such that

$$\begin{aligned} h(x10^k) &\geq h(x1) + k\cdot G_1(k)\cdot f(h(x1)) \\ &\geq k\cdot G_1(k)\cdot f(h(x1)) \\ &\geq k\cdot G_1(k)\cdot f(f'(m^{g(x1)})) \\ &\geq k\cdot G_1(k)\cdot m^{g(x1)}. \end{aligned}$$

Therefore,

$$\log_m h(x10^k) \geq \log_m k + \log_m G_1(k) + g(x1)$$

$$\geq \log_m k + G_2(k) + |x1| + G_3(|x1|)$$
$$\geq (\log_m k + |x1|) + G(h(x10^k)),$$

where

$$G_2(n) = \log_m G_1(n),$$
$$G_3(n) = \min\{g(x1)-|x1| \mid n = |x1|\},$$
$$G(n) = \min\{G_2(k)+G_3(|x1|) \mid h(x10^k) \geq n\}$$

all tend to infinity. The result of the conversion from $1^{h(x10^k)}$ to $x10^k$ requires only space $|x1| + \log_m k$, so the claim follows. □

An additional worktape head can be used to separate x1 from the m-ary representation of k, and additional space $\log_m n$ can provide for an m-ary counter up to k.

Let S_2 with $\log n \in o(S_2(n))$ be fully constructable by an $(m,\ell+2)$-machine, and let U_0 be the universal simulator of Condition 2 for m, $\ell+2$. Design an $(m,\ell+5)$-machine U_0' 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_0.

By the considerations above, whenever step 2 alone uses more space than the conversion of step 1 (and $S_2(h(x))$ is enough space since $\log n \in o(S_2(n))$), the whole program can be carried out at the extra cost (over just step 2) of the three additional worktape heads (one to separate the parts of the representation of x, one to separate the entire (left-adjusted) representation from the rest of the worktape, and one to scan the representation) plus $2 \cdot \log_m h(x) - G(h(x))$ worktape squares. By Proposition 7(iii), $L_{S_2}(U_0') \in \mathrm{NSPACE}(S_2,m,\ell+6)$. We prove that $L_{S_2}(U_0') \notin \mathrm{NSPACE}(S_1,m,\ell)$ for any space bound S_1 with

$S_2(n) - S_1(n + f(n)) \geq 4 \cdot \log_m n$.

Suppose the (m,ℓ)-machine U_1' accepts $L_{S_2}(U_0')$ within space S_1, where $S_2(n) - S_1(n + f(n)) \geq 4 \cdot \log_m n$. Because $\log n \in o(S_2(n))$, it is no loss of generality to assume also that $S_1(n) \geq \log_2 n$. To summarize the behavior of U_1',

$$L(U_1') = L_{S_2}(U_0') \subset L(U_0')$$

and, for $e \in L^{m,\ell+2}_{p.c.}$,

$$2 \cdot \log_m h(ex) - G(h(ex)) + c_e + \text{Space}_{M_e}(x) \leq S_2(h(ex)) \Rightarrow$$

$$\text{Space}_{U_1'}(1^{h(ex)}) \leq S_1(h(ex)).$$

Finally, design an $(m,\ell+2)$-machine U_1 to operate as follows on input $x \in \{0,1\}^*$:

1. Compute the m-ary representation of $h(x)$.
2. Simulate U_1' on $1^{h(x)}$.

If enough space $(S_2(h(x)))$ is used, then the extra cost (over just the computation by U_1' directly on $1^{h(x)}$) is no more than $2 \cdot \log_m h(x)$ worktape squares (half for an m-ary representation of $h(x)$ and the other half for an m-ary counter up to $h(x)$ to keep track of the input head position on $1^{h(x)}$) plus the two additional worktape heads. Hence,

$$\begin{aligned} L(U_1) &= \{x \mid 1^{h(x)} \in L(U_1')\} \\ &= \{x \mid 1^{h(x)} \in L_{S_2}(U_0')\} \\ &\subset \{x \mid 1^{h(x)} \in L(U_0')\} \\ &= L(U_0); \end{aligned}$$

and, for every $e \in L^{m,\ell+2}_{p.c.}$,

$$2 \cdot \log_m h(ex) - G(h(ex)) + c_e + \text{Space}_{M_e}(x) \leq S_2(h(ex)) \Rightarrow$$

$$\mathrm{Space}_{U_1}(ex) \le 2\cdot\log_m h(ex) + S_1(h(ex)).$$

For any recursive $L \subset \{1\}^*$, we can use U_1 just as in the proof of Theorem 18 to get an $(m,\ell+2)$-machine M_{e_0} with

$$x0^k \in L(M_{e_0}) \Leftrightarrow \begin{cases} x \in L, \text{ if } k \ge S'(|x|); \\ e_0x0^{k+1} \in L(U_1), \text{ if } k < S'(|x|); \end{cases}$$

$$\mathrm{Space}_{M_{e_0}}(x0^k) \le d + 2\cdot\log_m h(e_0x0^{k+1}) + S_1(h(e_0x0^{k+1})), \text{ if } x0^k \in L(M_{e_0})$$

for some appropriate space bound S', some constant d, and every $x \in \{1\}^*$. (This uses $S_1(n) \ge \log_2 n$.)

Claim 2. For each sufficiently long string $x \in \{1\}^*$, the following holds for every k:

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

Proof. Let $x \in \{1\}^*$ be so long that

$$G(n) \ge 2\cdot\log_m j + c_{e_0} + d$$

for every $n \ge h(e_0x)$. We establish the claim for x by induction on k running down from $k \ge S'(|x|)$ to $k = 0$.

$k \ge S'(|x|)$:

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

$k < S'(|x|)$: Assume $x0^{k+1} \in L(M_{e_0}) \Leftrightarrow x \in L$.

$$\begin{aligned} x0^k \in L(M_{e_0}) &\Rightarrow e_0x0^{k+1} \in L(U_1) \subset L(U_0) \\ &\Rightarrow x0^{k+1} \in L(M_{e_0}) \\ &\Rightarrow x \in L. \end{aligned}$$

$$x \in L \Rightarrow x0^{k+1} \in L(M_{e_0})$$

$$\Rightarrow \mathrm{Space}_{M_{e_0}}(x0^{k+1}) \le d + 2\cdot\log_m h(e_0x0^{k+2}) + S_1(h(e_0x0^{k+2}))$$

$$\Rightarrow 2\cdot\log_m h(e_0x0^{k+1}) - G(h(e_0x0^{k+1})) + c_{e_0} + \mathrm{Space}_{M_{e_0}}(x0^{k+1})$$

$$\le 4\cdot\log_m h(e_0x0^{k+1}) - G(h(e_0x0^{k+1})) + 2\cdot\log_m j + c_{e_0} + d$$

$$+ S_1(h(e_0x0^{k+2}))$$

$$\le 4\cdot\log_m h(e_0x0^{k+1}) + S_1(h(e_0x0^{k+1}) + f(h(e_0x0^{k+1})))$$

(because x is so long)

$$\le S_2(h(e_0x0^{k+1}))$$

$$\Rightarrow \mathrm{Space}_{U_1}(e_0x0^{k+1}) < \infty$$

$$\Rightarrow e_0x0^{k+1} \in L(U_1) \subset L(U_0)$$

$$\Rightarrow x0^k \in L(M_{e_0}). \ \square$$

If H is a nondecreasing recursive function so large that $h(y) \le H(|y|)$ for all $y \in \{0,1\}^*$, then Claim 2 gives the following for every sufficiently long $x \in L$:

$$\mathrm{Space}_{M_{e_0}}(x) \le d + 2\cdot\log_m h(e_0x0) + S_1(h(e_0x0))$$

$$\le d + 2\cdot\log_m j + 2\cdot\log_m h(e_0x) + S_1(h(e_0x) + f(h(e_0x)))$$

$$\le S_2(h(e_0x))$$

$$\le \sum_{n' \le H(|e_0| + |x|)} S_2(n')$$

$$\le \sum_{n' \le H(2\cdot|x|)} S_2(n')$$

It follows that $L \in \mathrm{NSPACE}\left(\sum_{n' \le H(2n)} S_2(n')\right)$. Since L is an arbitrary recursive language over {1}, this contradicts Lemma 16.

As in Theorem 18, a similar proof works for DSPACE. $\square$

Remark. The proof of Theorem 26 does not really require f to be linear space honest. It is enough to be able to compute f within space belonging to $o(S_2)$. If S_2 happens to be linear space honest, for example, $f(n) = \log^* S(n)$ will work.

For the particular function f defined by $f(n) = 2n$, we can get a result that extends all the way down to $O(\log n)$.

Theorem 27. Let $S_2(n)$ be fully constructable by a $(2,\ell)$-machine, $1 \in o(S_2(n) - \log_2 n)$, $\ell \geq 3$. Then each of the following set differences contains a language over $\{1\}$:

$$\mathrm{NSPACE}(S_2,2,\ell+1) - \bigcup \{\mathrm{NSPACE}(S_1,2,\ell) \mid 1 \in o(S_2(n)-S_1(2n))\},$$

$$\mathrm{DSPACE}(S_2,2,\ell+1) - \bigcup \{\mathrm{DSPACE}(S_1,2,\ell) \mid 1 \in o(S_2(n)-S_1(2n))\}.$$

Proof. Define an injection $h:(0+1)^*01(1^*0^*) \to N$ by

$$h(e01x0^k) = 2^k \cdot 3^j \cdot (6i+1),$$

where

$$x \in \{1\}^*,$$

$$1x = \mathrm{bin}(i),$$

$$1e = \mathrm{bin}(j).$$

Note that, with care, conversion of $1^{h(e01x0^k)}$ to (the reverse of) $e01x0^k$ can be accomplished within space $\log_2 h(e01x0^k)$ by a $(2,3)$-machine that leaves a worktape head marking each end of the string; this will account for the requirement $\ell \geq 3$.

Let U_0 be the universal simulator of Condition 2 for $2, \ell$. Design a $(2,\ell)$-machine U_0' to simulate U_0 on the input $e \cdot 1^{h(01x0^k)}$ when it receives the actual input $1^{h(e01x0^k)}$. Because $h(e01x0^k)/h(01x0^k)$ is an integer that depends only on e, this can be done at the extra cost of only d_e worktape squares, where d_e depends only on e, whenever at least space $\log_2 h(e01x0^k)$ is used (so that e can be computed from $1^{h(e01x0^k)}$). By Proposition 7(iii), $L_{S_2}(U_0') \in NSPACE(S_2,2,\ell+1)$. We prove that $L_{S_2}(U_0') \notin NSPACE(S_1,2,\ell)$ for any space bound S_1 with $1 \in o(S_2(n)-S_1(2n))$.

Suppose the $(2,\ell)$-machine U_1' accepts $L_{S_2}(U_0')$ within space S_1, where $1 \in o(S_2(n)-S_1(2n))$. Because $1 \in o(S_2(n) - \log_2 n)$, it is no loss of generality to assume $S_1(n) \geq \log_2 n$. To summarize the behavior of U_1',

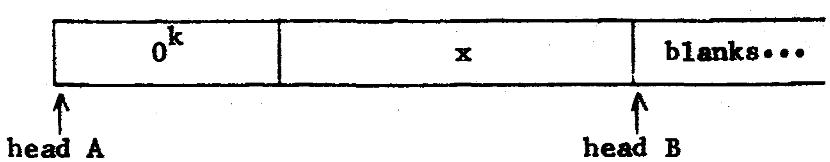
$$L(U_1') = L_{S_2}(U_0') \subset L(U_0')$$

and, for $e \in L^{2,\ell}_{p.c.}$ and $x \in \{1\}^*$,

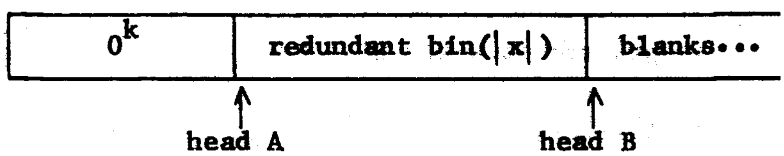
$$\max\{\log_2 h(e01x0^k),\ d_e + c_e + Space_{M_e}(1^{h(01x0^k)})\} \leq S_2(h(e01x0^k)) \Rightarrow$$
$$Space_{U_1'}(1^{h(e01x0^k)}) \leq S_1(h(e01x0^k)).$$

Let $L \subset \{1\}^*$ be any recursive language over $\{1\}$. Because L is recursive, we can take a deterministic (2,1)-machine M that accepts L within some space bound S that is fully constructable by a (2,1)-machine. Design a $(2,\ell)$-machine M' that operates as follows on the input string 1^n:

1. Use heads A, B, C to write down (the reverse of) $e01x0^k$ with $x \in \{1\}^*$, $h(e01x0^k) = n$, if possible. This requires no more than $\log_2 h(e01x0^k) \leq S_1(h(e01x0^{k+1}))$ worktape squares.
2. Check that $e \in L^{2,\ell}_{p.c.}$, and then erase all but x and 0^k.



3. Within the space occupied by x (for x sufficiently long), use heads A, B to compute a version of bin($|x|$) that has every second and third tape square redundant. (Cf., proof of Lemma 17.) As in the proofs of Proposition 4, Theorem 15, and Lemma 17, one of these sets of redundant tape squares can be used to mark the two ends of the string so that it can be carried around and referenced by head A without confusion. The other set can be used as a binary counter up to $|x|$.



4. Use head A in an attempted full construction of space $S(|x|)$ in the additional space occupied by 0^k. All necessary input data can be obtained from the redundant version of bin($|x|$). There is success iff $S(|x|) \leq k$.

5. If $S(|x|) \leq k$, then use head A to erase the tape out to head B except for the redundant version of bin($|x|$), and then compute as M would on the input x. This requires no more than

$$\begin{aligned} S(|x|) + |x| &\leq k + |x| \\ &\leq \log_2 h(e01x0^{k+1}) \\ &\leq S_1(h(e01x0^{k+1})) \end{aligned}$$

worktape squares.

6. If $S(|x|) > k$, then completely erase the tape out to head B, freeing all ℓ worktape heads, and then compute as $U_1{}'$ would on the input $1^{h(e01x0^{k+1})}$ (which is just twice the length of the

actual input). This requires no more than $S_1(h(e01x0^{k+1}))$ worktape squares for acceptance.

To summarize the behavior of M' on $1^{h(e01x0^k)}$,

$k \geq S(|x|) \Rightarrow$ behave like M on x;

$k < S(|x|) \Rightarrow$ behave like U_1' on $1^{h(e01x0^{k+1})}$.

For $1^{h(e01x0^k)} \in L(M')$,

$\text{Space}_{M'}(1^{h(e01x0^k)}) \leq S_1(h(e01x0^{k+1}))$.

The recursion theorem (Theorem 15) does not apply as stated, so we adapt it. Let e_2 be the program code for M'. Take M_{e_1} to be a $(2,\ell)$-machine that operates as follows, given $1^{h(01x0^k)}$ (where $x \in \{1\}^*$) on its input tape and $e \in L^{2,\ell}_{p.c.}$ on its worktape:

1. Convert e to f(e), where f is as in Condition 3.
2. Convert f(e) to the string $y \in \{1\}^*$ of length 3^j, where $\text{bin}(j) = 1 \cdot f(e)$.
3. Simulate U_0 on $e_2 \cdot 1^{h(f(e)01x0^k)}$. To do this, commit e to finite-state memory and carry y around with one of the worktape heads of U_0, modifying symbols of y to mark the ends of the string and the position in y that indicates which, if any, of the 3^j copies of $1^{h(01x0^k)}$ that compose $1^{h(f(e)01x0^k)}$ U_0 is currently scanning on its input tape.

Let $e_0 = f(e_1)$. Then M_{e_0} operates as follows on input $1^{h(01x0^k)}$, where $x \in \{1\}^*$:

1. Write e_1 on the worktape.
2. Convert e_1 to $f(e_1) = e_0$.

3. Convert e_0 to the string $y \in \{1\}^*$ of length 3^j, where $\text{bin}(j) = 1e_0$.

4. Simulate U_0 on $e_2 \cdot 1^{h(e_0 01x0^k)}$.

Thus,

$$1^{h(01x0^k)} \in L(M_{e_0}) \Leftrightarrow e_2 \cdot 1^{h(e_0 01x0^k)} \in L(U_0)$$
$$\Leftrightarrow 1^{h(e_0 01x0^k)} \in L(M'),$$

$$\text{Space}_{M_{e_0}}(1^{h(01x0^k)}) \le c + \text{Space}_{M'}(1^{h(e_0 01x0^k)}),$$

where c is c_{e_2} plus the number of worktape squares required for steps 1, 2, 3.

<u>Claim</u>. For each sufficiently long string $x \in \{1\}^*$, the following holds for every k:

$$1^{h(01x0^k)} \in L(M_{e_0}) \Leftrightarrow x \in L.$$

<u>Proof</u>. Let $x \in \{1\}^*$ be so long that

$$S_2(n) - S_1(2n) \ge d_{e_0} + c_{e_0} + c$$

for every $n \ge h(e_0 01x)$. We establish the claim for x by induction on k running down from $k \ge S(|x|)$ to $k = 0$.

$k \ge S(|x|)$:

$$1^{h(01x0^k)} \in L(M_{e_0}) \Leftrightarrow 1^{h(e_0 01x0^k)} \in L(M')$$
$$\Leftrightarrow x \in L(M) = L.$$

$k < S(|x|)$: Assume $1^{h(01x0^{k+1})} \in L(M_{e_0}) \Leftrightarrow x \in L$.

$$1^{h(01x0^k)} \in L(M_{e_0}) \Rightarrow 1^{h(e_0 01x0^k)} \in L(M')$$

$$\Rightarrow 1^{h(e_0 01x0^{k+1})} \in L(U_1') \subset L(U_0')$$

$$\Rightarrow 1^{h(01x0^{k+1})} \in L(M_{e_0})$$

$$\Rightarrow x \in L.$$

$$\begin{aligned}
x \in L \Rightarrow\ & 1^{h(01x0^{k+1})} \in L(M_{e_0}) \\
\Rightarrow\ & 1^{h(e_0 01x0^{k+1})} \in L(M') \\
\Rightarrow\ & \mathrm{Space}_{M'}(1^{h(e_0 01x0^{k+1})}) \le S_1(h(e_0 01x0^{k+2})) \\
\Rightarrow\ & \max\{\log_2 h(e_0 01x0^{k+1}),\ d_{e_0} + c_{e_0} + \mathrm{Space}_{M_{e_0}}(1^{h(01x0^{k+1})})\} \\
& \le \max\{\log_2 h(e_0 01x0^{k+1}),\ d_{e_0} + c_{e_0} + c \\
& \qquad + \mathrm{Space}_{M'}(1^{h(e_0 01x0^{k+1})})\ \} \\
& \le d_{e_0} + c_{e_0} + c + S_1(h(e_0 01x0^{k+2})) \\
& \le S_2(h(e_0 01x0^{k+1})) \\
& \quad \text{(because } x \text{ is so long)} \\
\Rightarrow\ & 1^{h(e_0 01x0^{k+1})} \in L(U_1') \\
\Rightarrow\ & 1^{h(e_0 01x0^{k})} \in L(M') \\
\Rightarrow\ & 1^{h(01x0^{k})} \in L(M_{e_0}).\ \square
\end{aligned}$$

Finally, M_{e_0} can be modified to use its finite-state control to reject padded inputs (those of even length) and to satisfy the claim for short ones without using the worktape. If $1e_0 = \mathrm{bin}(j)$, then this gives an off-line TM that accepts $\{1^{6i+1} \mid x \in L,\ 1x = \mathrm{bin}(i)\}$ within space $S_1(2 \cdot 3^j \cdot h(011^n)) \in O(S_2(3^j \cdot h(011^n)))$. From this it is easy to derive a fixed recursive space bound S_3 for which $L \in \mathrm{NSPACE}(S_3)$. Since L is an arbitrary recursive language over $\{1\}$, this contradicts Lemma 16.

As in Theorem 18, a similar proof works for DSPACE. $\square$

<u>Corollary 28</u>. Let S_2, S be fully constructable; and let $f(n) \in O(n)-O(1)$ be nondecreasing and linear space honest. There are languages over $\{1\}$ that bear witness to the following proper containments:

(i) $\bigcup \{\text{DSPACE}(S_1) |\ S_1 \in o(S_2)\} \subsetneqq \text{DSPACE}(S_2)$,

$\bigcup \{\text{NSPACE}(S_1) |\ S_1^2 \in o(S_2)\} \subsetneqq \text{DSPACE}(S_2)$

whenever $\log n \in o(S_2(n))$.†

(ii) $\bigcup \{\text{NSPACE}(S_1) |\ S_1(n + f(n)) \in o(S_2(n)),\ S_1(n) \in O(S_2(n))\}$

$\subsetneqq \text{NSPACE}(S_2)$

whenever $\log n \in o(S_2(n))$.

(iii) $\bigcup \{\text{NSPACE}(S_1) |\ S_1(2n) \in O(S_2(n)),\ S_1(n) \in o(S_2(n))\} \subsetneqq \text{NSPACE}(S_2)$,

$\bigcup \{\text{DSPACE}(S_1) |\ S_1(2n) \in O(S_2(n)),\ S_1(n) \in o(S_2(n))\} \subsetneqq \text{DSPACE}(S_2)$

whenever $1 \in o(S_2(n))$.

(iv) $\text{DSPACE}(S,m) \subsetneqq \text{DSPACE}(S)$

whenever $1 \in o(S(n))$.

(v) $\text{NSPACE}(S,m) \subsetneqq \text{NSPACE}(S)$

whenever $\log n \in o(S(n))$,

$S(n + f(n)) \in O(S(n))$.

(vi) $\text{NSPACE}(S,m) \subsetneqq \text{NSPACE}(S)$

whenever $1 \in o(S(n))$,

---

†The technique of [MM71] can be used to show that each of the following set differences contains a language over a one-letter alphabet if S_2 is fully constructable:

$\text{DSPACE}(S_2) - \bigcup \{\text{DSPACE}(S_1) |\ S_1 \text{ fully constructable, } S_2 \notin O(S_1)\}$,

$\text{DSPACE}(S_2) - \bigcup \{\text{NSPACE}(S_1) |\ S_1 \text{ fully constructable, } S_2 \notin O(S_1^2)\}$.

$S(2n) \in O(S(n))$.

(vii) $NSPACE(S,m) \subsetneq NSPACE(S,m+1,1)$,

$DSPACE(S,m) \subsetneq DSPACE(S,m+1,1)$

whenever $\log n \in o(S(n))$,

$S(n + f(n)) - S(n) \in o(S(n))$,

m is sufficiently large.

(If S is fully constructable by an (m,1)-machine, then m is sufficiently large; if S is actually linear space honest, then m = 2 is sufficiently large.)

(viii) $NSPACE(S,m) \subsetneq NSPACE(S,m+1,1)$,

$DSPACE(S,m) \subsetneq DSPACE(S,m+1,1)$

whenever $1 \in o(S(n))$,

$S(2n)-S(n) \in o(S(n))$,

m is sufficiently large.

(If S is fully constructable by an (m,1)-machine, then m is sufficiently large.)

(ix) $NSPACE(S,m,\ell) \subsetneq NSPACE(S,m)$ for all ℓ,

$DSPACE(S,m,\ell) \subsetneq DSPACE(S,m)$ for all ℓ

whenever $\log n \in O(\log S(n))$,

$S(n + f(n)) - S(n) \in O(\log S(n))$,

m is sufficiently large.

(If S is fully constructable by an (m,1)-machine, then m is sufficiently large; if S is actually linear space honest, then m = 2 is sufficiently large.)

(x) $NSPACE(\delta \cdot S,2,\ell) \subsetneq NSPACE(\delta \cdot S,2)$ for all ℓ,

$DSPACE(\delta \cdot S, 2, \ell) \subsetneq DSPACE(\delta \cdot S, 2)$ for all ℓ

whenever $1 \in o(S(n))$,

$S(2n) - S(n) \in O(\log S(n))$,

δ is rational and sufficiently large.

(If $\delta \cdot S$ is fully constructable by a (2,1)-machine and $1 \in o(\delta \cdot S(n) - \log_2 n)$, then δ is sufficiently large.)

Proof. (i) Use Theorem 25. (See Corollary 19(ii).)

(ii) Use Theorem 26. (See Corollary 19(iii).)

(iii) Use Theorem 27. (See Corollary 19(iii).)

(iv) Use Theorem 25. (See Corollary 20(ii).)

(v) Use Theorem 26. (See Corollary 20(iii).)

(vi) Use Theorem 27. (See Corollary 20(iii).)

(vii) Use Theorem 26. (See Corollary 23(i).)

(viii) Use Theorem 27. (See Corollary 23(i).)

We explicitly prove parts (ix), (x) to show just how few additional worktape heads can increase the power to accept languages over a one-letter alphabet.

(ix) Take m so large that S is fully constructable by an (m,1)-machine, and look at any ℓ. Take k so large that

$$4 \cdot \log_m n + S(n + f(n)) - S(n) \le k \cdot \log_m S(n).$$

Since $S + k \cdot \log_m S$ is fully constructable by an (m,3)-machine, Theorem 26 gives a language over $\{1\}$ that bears witness to the proper containment in

$$NSPACE(S, m, \ell) \subsetneq NSPACE(S + k \cdot \log_m S, m, \ell+6)$$
$$\subset NSPACE(S, m, \ell+k+9)$$

$\subset NSPACE(S,m)$.

The proof for DSPACE is identical.

(x) Take rational δ so large that $\delta\cdot S$ is fully constructable by a (2,1)-machine and $\delta\cdot S(n) \geq \log_2 S$. (This uses $1 \in o(S(n))$.) Look at any $\ell \geq 3$. Take $c > 0$ so large that $\delta\cdot(S(2n)-S(n)) \leq c\cdot\log_2 S(n)$. Since $\delta\cdot S + (\lfloor c\rfloor+1)\cdot\log_2 S$ is fully constructable by a (2,1)-machine, Theorem 27 gives a language over $\{1\}$ that bears witness to the proper containment in

$$\begin{aligned} NSPACE(\delta\cdot S,2,\ell) &\subsetneq NSPACE(\delta\cdot S + (\lfloor c\rfloor+1)\cdot\log_2 S,\ 2,\ \ell+1) \\ &\subset NSPACE(\delta\cdot S,\ 2,\ \ell+\lfloor c\rfloor+3) \\ &\subset NSPACE(\delta\cdot S,2). \end{aligned}$$

The proof for DSPACE is identical. $\sqcap$

__Remarks.__ (i) For any common, well-behaved space bound S with $S(n+1)-S(n) \in o(S(n))$ or $S(n+1)-S(n) \in o(\log S(n))$, there is a nondecreasing and linear space honest function $f(n) \notin O(1)$ with $S(n+f(n))-S(n) \in o(S(n))$ or $S(n+f(n))-S(n) \in o(\log S(n))$, respectively.

(ii) Although we cannot presently prove a general hierarchy theorem for space bounds below $\log n$ (for languages over a one-letter alphabet), we can exhibit a nonregular language over $\{1\}$ in $DSPACE(\log\log n)$. The language is $L = \{1^n \mid n \in A\}$, where

$f(n) = \min\{k \mid n \text{ is } \underline{\text{not}} \text{ divisible by } k\}$

$A = \{n \mid f(n) \text{ is a power of } 2\}$.

(A. Meyer points out that J. Hartmanis once mentioned a quite similar example in a private communication to him.) An acceptor for L can be obtained from the deterministic off-line TM that fully constructs the space bound of Proposition 7(iv), so $L \in \mathrm{DSPACE}(\log\log n)$. To prove L nonregular, it certainly suffices to find a positive integer n_p for each prime $p > 2$, such that

$$A \cap \{m \cdot n_p \mid 1 \le m < p\} = \emptyset \ne A \cap \{m \cdot n_p \mid 1 \le m\}.$$

Just take n_p to be the least common multiple of the positive integers not exceeding 2^k, where $2^k < p < 2^{k+1}$. For $1 \le m < p$, then,

$$2^k < f(m \cdot n_p) \le p < 2^{k+1},$$

so that $m \cdot n_p \notin A$. Yet, the least common multiple of the positive integers smaller than 2^{k+1} is a multiple of n_p that does belong to A.

<u>Examples</u>. There are languages over a one-letter alphabet that bear witness to the following proper containments:

$\mathrm{NSPACE}(2^n/\log^* n) \subsetneq \mathrm{NSPACE}(2^n)$ (part (ii) with $f(n) = \log^* \log^* n$).

$\bigcup \{\mathrm{NSPACE}(S) \mid S(n) \in o(\log n)\} \subsetneq \mathrm{NSPACE}(\log n)$ (minimum example of part (iii)).

$\mathrm{DSPACE}(2^n, m) \subsetneq \mathrm{DSPACE}(2^n)$ (part (iv))

$\mathrm{NSPACE}(n^{8.13}/(\log_{2.21} n)^{5.6}, m) \subsetneq \mathrm{NSPACE}(n^{8.13}/(\log_{2.21} n)^{5.6}, m+1, 1)$ (part (vii)).

$\mathrm{NSPACE}((\log_{2.21} n)^{5.6}, m) \subsetneq \mathrm{NSPACE}((\log_{2.21} n)^{5.6}, m+1, 1)$ (part (vii) or part (viii)).

$\mathrm{NSPACE}(2^{n/\log^* n}, m) \subsetneq \mathrm{NSPACE}(2^{n/\log^* n}, m+1, 1)$ (near-maximum example of part (vii); $f(n) = \log^* \log^* n$).

$\mathrm{NSPACE}(n^{1/\delta}, m, \ell) \subsetneq \mathrm{NSPACE}(n^{1/\delta}, m, \ell + \lfloor 4 \cdot \delta \rfloor + 10)$ for every rational $\delta \ge 1$ (proof of part (ix)).

$\mathrm{NSPACE}(n \cdot \log_2 n/\log^* n, m, \ell) \subsetneq \mathrm{NSPACE}(n \cdot \log_2 n/\log^* n, m, \ell + 13)$

(near-maximum example of part (ix)).

$\mathrm{NSPACE}(\log_2 n,\ 2^k,\ \ell) \subsetneq \mathrm{NSPACE}(\log_2 n,\ 2^k,\ \ell+5)$ for every $\ell \geq 3$

(proof of part (x)).

$\mathrm{NSPACE}((\log_2 n)(\log_2 \log_2 n),\ 2^k,\ \ell) \subsetneq \mathrm{NSPACE}((\log_2 n)(\log_2 \log_2 n),$

$2^k,\ \ell+6)$ for every $\ell \geq 3$ (near-maximum example of part (x)).

<u>Corollary 29</u>. Each of the following set differences contains a language over a one-letter alphabet:

NHEADS(k+5) - NHEADS(k),

DHEADS(k+5) - DHEADS(k).

<u>Proof</u>. Corollary 28 gives a language over $\{1\}$ that bears witness to the proper containment in

$$\begin{aligned} \mathrm{NHEADS}(k) &\subset \mathrm{NSPACE}(\log_2 n,\ 2^k,\ 1) \text{ (by Lemma 21)} \\ &\subsetneq \mathrm{NSPACE}(\log_2 n,\ 2^k,\ 6) \\ &\subset \mathrm{NSPACE}(\log_2 n,\ 2^{k+1},\ 1) \\ &\subset \mathrm{NHEADS}(k+5) \text{ (by Lemma 21).} \end{aligned}$$

The argument for DHEADS is identical. □

10. Open questions

Our most general open questions, of course, concern necessary and sufficient conditions for containment and separation among the NSPACE(S,m,ℓ), DSPACE(S,m,ℓ) complexity classes.

1. For containment we ask in particular how close the truth comes to the "ideal" result

$$m_2^{S_2(n)} \cdot S_2(n)^{\ell_2} \in O(m_1^{S_1(n)} \cdot S_1(n)^{\ell_1}) \Rightarrow$$
$$\mathrm{NSPACE}(S_2,m_2,\ell_2) \subset \mathrm{DSPACE}(S_1,m_1,\ell_1).$$

This very strong statement would immediately yield and perfect all the results of Section 2. It would also yield NSPACE(S,m,ℓ) = DSPACE(S,m,ℓ), however, so it seems extremely likely that the truth stops somewhat short of the statement.

2. For separation we ask how close the truth comes to the "ideal" result that, for S_2 fully constructable, there must be a language in

$$\mathrm{DSPACE}(S_2,m_2,\ell_2) - \bigcup \{\mathrm{NSPACE}(S_1,m_1,\ell_1) \mid$$
$$m_2^{S_2(n)} \cdot S_2(n)^{\ell_2} \notin O(m_1^{S_1(n)} \cdot S_1(n)^{\ell_1})\}.$$

This very strong statement would immediately yield and perfect all the separation results of Section 4.

3. Lemma 21 illustrates the relationship between additional <u>input</u> heads and additional space log n. If we consider a model that has $k \geq 1$ read-only heads on its input tape, then the open statements above could be rephrased in terms of quantities of the form $m^{S(n)} \cdot S(n)^{\ell} \cdot n^k$ rather than just $m^{S(n)} \cdot S(n)^{\ell}$. Then they would include and perfect

also Lemma 21, Corollary 22, and the work of [Ib72].

The following specific instances of the above questions are just beyond the frontier of our knowledge:

4. DSPACE(n,2,1) = DSPACE($n/\log_2 3$, 3, 1)? Proposition 3 comes close to an affirmative answer.

5. DSPACE(n,2,1) = DSPACE($n - \log_2 n$, 2, 2)? Propositions 4, 5 come close to an affirmative answer.

6. NSPACE(log n) $\subset$ DSPACE($(\log n)^2/\log^* n$)? Proposition 6 comes close to an affirmative answer.

7. NSPACE(S,m,ℓ) = DSPACE(S,m,ℓ)? Everybody expects a negative answer, but our study offers no clear and convincing evidence for one. A negative answer to any of open questions 8, 9, 10, 11, 19, 20 would do, though.

8. DSPACE(log n) $\not\subset$ NSPACE($(\log_2 n)/2$, 2, 1)? Theorem 8 comes close to an affirmative answer; e. g., DSPACE(log n) $\not\subset$ NSPACE($(\log_2 n)/3$, 2, 1).

9. DSPACE($(\log n)(\log^* n)$) $\not\subset$ NSPACE(log n)? Corollary 19(i) comes close to an affirmative answer.

10. NSPACE(2^{2^n}) $\subsetneq$ NSPACE($2^{2^{n+1}}/\log^* n$)? Corollary 19(iii) comes close to an affirmative answer, and Corollary 19(ii) gives an affirmative answer for the DSPACE analogue.

11. NSPACE($(\log^* n)^n$, 2, 1) $\subsetneq$ NSPACE($(\log^* n)^n$)? Corollary 20(iii) comes close to an affirmative answer, and Corollary 20(ii) gives an affirmative answer for the DSPACE analogue.

12. $\mathrm{DSPACE}(\log_2 n, 2, 1) \subset \mathrm{DHEADS}(4)$? Lemma 21 comes close to an affirmative answer.

13. $\mathrm{DHEADS}(k) \subsetneq \mathrm{DHEADS}(k+1)$?

14. $\mathrm{DHEADS}(k+1) \not\subset \mathrm{NHEADS}(k)$? For the particular case $k = 2$, we suspect that $\{1^n \mid k \in N, n = 2^{2^k}\} \notin \mathrm{NHEADS}(2)$, but the suspicion does not generalize.

15. $\mathrm{DSPACE}(2^n,m) \subsetneq \mathrm{DSPACE}(2^n,m+1)$? Corollaries 20, 23(ii) both come close to an affirmative answer.

16. $\mathrm{DSPACE}(n(\log_2 n)(\log^* n), 2, 1) \subsetneq \mathrm{DSPACE}(n(\log_2 n)(\log^* n), 2)$? Corollary 24(ii) comes close to an affirmative answer.

17. $\mathrm{DSPACE}(\log_2 n, 2, 1) \subsetneq \mathrm{DSPACE}(\log_2 n, 2, 5)$? The proof of Corollary 24(ii) comes close to an affirmative answer.

18. $\mathrm{DSPACE}(n - (\log_2 n)^{1/2}, 2, 1) \subsetneq \mathrm{DSPACE}(n,2,1)$?

19. Does $\mathrm{NSPACE}(2^{2^{n+1}}) - \mathrm{NSPACE}(2^{2^n})$ contain a language over a <u>one-letter alphabet</u>? Corollary 28(ii) comes close to an affirmative answer.

20. Does $\mathrm{NSPACE}(2^n) - \mathrm{NSPACE}(2^n,2,1)$ contain a language over a <u>one-letter alphabet</u>? Corollary 28(v) comes close to an affirmative answer.

21. Does $\mathrm{DSPACE}(n \cdot \log_2 n, 2) - \mathrm{DSPACE}(n \cdot \log_2 n, 2, 1)$ contain a language over a <u>one-letter alphabet</u>? Corollary 28(ix) comes close to an affirmative answer.

22. Does DSPACE($(\log_2 n)^2$, 2) - DSPACE($(\log_2 n)^2$, 2, 1) contain a language over a <u>one-letter alphabet</u>? Corollary 28(x) comes close to an affirmative answer.

23. Does DHEADS(k+2) - DHEADS(k) contain a language over a <u>one-letter alphabet</u>?

Finally, we list a few miscellaneous open questions.

24. Is there a hierarchy of languages over $\{1\}$ for space bounds below log n?

25. For S fully constructable by an (m,ℓ)-machine and U_0 the universal simulator of Condition 2 for m, ℓ, is there any language in NSPACE(S,m,ℓ+1) that requires more space (on an $(m,\ell+1)$-machine) than the S-cutoff of U_0?

26. Even if $L \in$ NSPACE(S_2) - NSPACE(S_1), there may be an off-line TM that accepts infinitely many strings $x \in L$ within space $S_1(|x|)$. When can we find an infinite language $L \in$ NSPACE(S_2) such that every off-line TM that accepts L requires more than space $S_1(|x|)$ on <u>all</u> <u>but</u> <u>finitely</u> <u>many</u> strings $x \in L$?

27. Is there some conceptually simple language in $\bigcup$ {NHEADS(k)| $k \geq 1$} or $\bigcup$ {DHEADS(k)| $k \geq 1$} which is not in NHEADS(k) or DHEADS(k) for any small k (say k = 3)? If, for X a matrix of strings over $\{0,1\}$, we define

   r(X) = row-wise concatenation of X,

   c(X) = column-wise concatenation of X,

then some good candidates are

$\{r(X)c(X) \mid X \text{ is a } k \times 2 \text{ matrix}\}$,

$\{r(X)c(X) \mid X \text{ is a } k \times 2 \text{ matrix for } \underline{\text{some}}\ k\}$,

$\{r(X)c(X) \mid X \text{ is a } k \times k \text{ matrix}\}$,

$\{r(X)c(X) \mid X \text{ is a } k \times k \text{ matrix for } \underline{\text{some}}\ k\}$.

What are the complexities of these languages?

28. If S is fully constructable by an (m,ℓ)-machine, does

$$\log_m n - (\ell-1)\cdot\log_m \log_m n - S(n) \in O(1)$$

necessarily hold? Proposition 7(v) comes close to an affirmative answer.

BIBLIOGRAPHY

[Aan74] Aanderaa, S. O., On k-tape versus (k-1)-tape real time computation, SIAM-AMS Colloquia on Applied Mathematics, 7(1974), To appear.

[AU72] Aho, A. V. and Ullman, J. D., The Theory of Parsing, Translation, and Compiling; Volume I: Parsing, Prentice-Hall, Englewood Cliffs, N. J., 1972.

[AU73] Aho, A. V. and Ullman, J. D., The Theory of Parsing, Translation, and Compiling; Volume II: Compiling, Prentice-Hall, Englewood Cliffs, N. J., 1973.

[BG70] Book, R. V. and Greibach, S. A., Quasi-realtime languages, Mathematical Systems Theory, Vol. 4, No. 2, June 1970, pp. 97-111.

[BGIW70] Book, R. V., Greibach, S. A., Ibarra, O. H. and Wegbreit, B., Tape-bounded Turing acceptors and principal AFLs, Journal of Computer and System Sciences, Vol. 4, No. 6, December 1970, pp. 622-625.

[BGW70] Book, R. V., Greibach, S. A. and Wegbreit, B., Time- and tape-bounded Turing acceptors and AFLs, Journal of Computer and System Sciences, Vol. 4, No. 6, pp. 606-621.

[Bk72] Book, R. V., On languages accepted in polynomial time, The SIAM Journal on Computing, Vol. 1, No. 4, December 1972, pp. 281-287.

[Blm67] Blum, M., A machine-independent theory of the complexity of recursive functions, Journal of the Association for Computing Machinery, Vol. 14, No. 2, April 1967, pp. 322-336.

[Bor72] Borodin, A., Computational complexity and the existence of complexity gaps, Journal of the Association for Computing Machinery, Vol. 19, No. 1, January 1972, pp. 158-174.

[Ck71] Cook, S. A., The complexity of theorem-proving procedures, Proceedings of Third Annual ACM Symposium on Theory of Computing, Shaker Heights, Ohio, 1971, pp. 151-158.

[Ck73] Cook, S. A., A hierarchy for nondeterministic time complexity, Journal of Computer and System Sciences, Vol. 7, No. 4, August 1973, pp. 343-353.

[Con72] Constable, R. L., The operator gap, Journal of the Association for Computing Machinery, Vol. 19, No. 1, January 1972, pp. 175-183.

[Con73] 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.

[F167] Floyd, R. W., Nondeterministic algorithms, Journal of the Association for Computing Machinery, Vol. 14, No. 4, October 1967, pp. 636-644.

[FMR72] Fischer, P. C., Meyer, A. R. and Rosenberg, A. L., Real-time simulation of multihead tape units, Journal of the Association for Computing Machinery, Vol. 19, No. 4, October 1972, pp. 590-607.

[FO73] Feldman, E. D. and Owings, J. C. Jr., A class of universal linear bounded automata, Information Sciences, Vol. 6, No. 2, April 1973, pp. 187-190.

[FR74] Fischer, M. J. and Rabin, M. O., Super-exponential complexity of Presburger arithmetic, Project MAC Technical Memorandum 43, Massachusetts Institute of Technology, February 1974.

[GB74] Gill, J. and Blum, M., On almost everywhere complex recursive functions, Journal of the Association for Computing Machinery, Vol. 21, No. 3, July 1974, pp. 425-435.

[Ha72] Hartmanis, J., On non-determinancy in simple computing devices, Acta Informatica, Vol. 1, Fasc. 4, 1972, pp. 336-344.

[HaS65] Hartmanis, J. and Stearns, R. E., On the computational complexity of algorithms, Transactions of the American Mathematical Society, Vol. 117, May 1965, pp. 285-306.

[HeS66] Hennie, F. C. and Stearns, R. E., Two-tape simulation of multitape Turing machines, Journal of the Association for Computing Machinery, Vol. 13, No. 4, October 1966, pp. 533-546.

[HU69a] Hopcroft, J. E. and Ullman, J. D., Some results on tape-bounded Turing machines, Journal of the Association for Computing Machinery, Vol. 16, No. 1, January 1969, pp. 168-177.

[HU69b] Hopcroft, J. E. and Ullman, J. D., Formal Languages and Their Relation to Automata, Addison-Wesley, Reading, Mass., 1969.

[Hun73] Hunt, H. B. III, The equivalence problem for regular expressions with intersection is not polynomial in tape, Technical Report 73-161, Department of Computer Science, Cornell University, March 1973.

[Ib72] Ibarra, O. H., A note concerning nondeterministic tape complexities, Journal of the Association for Computing Machinery, Vol. 19, No. 4, October 1972, pp. 608-612.

[Ib73a] Ibarra, O. H., On two-way multihead automata, Journal of Computer and System Sciences, Vol. 7, No. 1, February 1973, pp. 28-36.

[Ib73b] Ibarra, O. H., A hierarchy theorem for polynomial-space recognition, Department of Computer Science, University of Minnesota, Minneapolis, October 1, 1973.

[IS73] Ibarra, O. H. and Sahni, S. K., Hierarchies of Turing machines with restricted tape alphabet size, Department of Computer Science, University of Minnesota, Minneapolis, October 12, 1973.

[Krp72] Karp, R. M., Reducibility among combinatorial problems, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, ed., Plenum Press, New York, 1972, pp. 85-103.

[Lyn72] Lynch, N. A., Relativization of the theory of computational complexity, Project MAC Technical Report 99, Massachusetts Institute of Technology, June 1972.

[Mey73] Meyer, A. R., Weak monadic second order theory of successor is not elementary-recursive, Project MAC Technical Memorandum 38, Massachusetts Institute of Technology, December 1973.

[MM71] 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.

[MS72] Meyer, A. R. and Stockmeyer, L. J., The equivalence problem for regular expressions with squaring requires exponential space, Proceedings of 13th Annual Symposium on Switching & Automata Theory, College Park, Maryland, 1972, pp. 125-129.

[NZ60] Niven, I. and Zuckerman, H. S., An Introduction to the Theory of Numbers, Wiley, New York, 1960.

[RF65] Ruby, S. and Fischer, P. C., Translational methods and computational complexity, IEEE Conference Record on Switching Circuit Theory and Logical Design, Ann Arbor, Michigan, 1965, pp. 173-178.

[Rit63] Ritchie, R. W., Classes of predictably computable functions, Transactions of the American Mathematical Society, Vol. 106, January 1963, pp. 139-173.

[Rog67] Rogers, H. Jr., Theory of Recursive Functions and Effective Computability, McGraw-Hill, New York, 1967.

[Sav70] Savitch, W. J., Relationships between nondeterministic and deterministic tape complexities, Journal of Computer and System Sciences, Vol. 4, No. 2, April 1970, pp. 177-192.

[SFM73] Seiferas, J. I., Fischer, M. J. and Meyer, A. R., Refinements of the nondeterministic time and space hierarchies, Proceedings of 14th Annual Symposium on Switching & Automata Theory, Iowa City, Iowa, 1973, pp. 130-137.

[SHL65] Stearns, R. E., Hartmanis, J. and Lewis, P. M. II, Hierarchies of memory limited computations, IEEE Conference Record on Switching Circuit Theory and Logical Design, Ann Arbor, Michigan, 1965, pp. 179-190.

[SM73] Stockmeyer, L. J. and Meyer, A. R., Word problems requiring exponential time: preliminary report, Proceedings of Fifth Annual ACM Symposium on Theory of Computing, Austin, Texas, 1973, pp. 1-9.

[St74] Stockmeyer, L. J., The complexity of decision problems in automata theory and logic, Project MAC Technical Report 133, Massachusetts Institute of Technology, June 1974.

[Yam62] Yamada, H., Real-time computation and recursive functions not real-time computable, IRE Transactions on Electronic Computers, Vol. EC-11, No. 6, December 1962, pp. 753-760.

[Yng71] Young, P., Easy constructions in complexity theory: speed-up and gap theorems, Computer Science Department Technical Report 57, Purdue University, July 1971.

APPENDIX I

TIME DIAGONALIZATION

Diagonalization is a technique for constructing a language that is not in some given class. If the members of the class can be described by character strings, then the simplest diagonal construction includes the string x just if the language described by that string does <u>not</u> include x. A useful variant on this idea is to include xy iff the language described by its prefix x does not.

To diagonalize over a <u>complexity</u> class, a good approach is to describe languages by encoding the programs of the resource-bounded automata that accept them. Then the construction can be performed by employing a "universal simulator" that can simulate any automaton from its program code. The simulation can then be examined, with disagreement in mind, to decide whether or not it leads to acceptance.

Because we shall be interested also in an <u>upper</u> bound on the complexity of the diagonal language, we will want the construction to be effective and as efficient as possible. This calls for an efficient universal simulator. To diagonalize over DTIME(T), we can use the universal simulation technique of Hennie and Stearns [HeS66] to simulate t steps of the deterministic TM acceptor with program code e by $c_e \cdot t \cdot \log t$ steps of the simulator, where the constant c_e depends only on e. The diagonalization technique used by Hartmanis and Stearns [HaS65] then shows that $\mathrm{DTIME}(T_2) - \mathrm{DTIME}(T_1)$ is nonempty whenever T_2 is a running time and $T_2 \notin O(T_1 \log T_1)$. For further details, see [HaS65], [HeS66], [Con73], and the sketch below of a nondeterministic diagonalization over

DTIME(T_1).

For <u>nondeterministic</u> TM acceptors, we can use the technique of Book, Greibach, and Wegbreit [BGW70] (Lemma 8 of Chapter Two of this thesis) to get a more efficient universal simulation. (Doing so requires that their technique be effective, and it is.) In the following proposition, we use the details of this simulation in a diagonal construction in the style of [HaS65], [HeS66].

<u>Proposition</u>. If T_2 is a running time with $T_2 \notin O(T_1)$, then $\mathrm{NTIME}(T_2) - \mathrm{DTIME}(T_1) \neq \emptyset$.

<u>Proof sketch</u>. (We assume familiarity with the proof sketch in Chapter Two of Lemma 8.) We construct a TM acceptor M that diagonalizes over DTIME(T_1) within time bound T_2. Given an input ex (with e a program code), M performs the (nondeterministic) Lemma 8 simulation of M_e on ex and simultaneously operates clocks for the running times $T_2/2$ and T_2. 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.

If the outcome is the second one and that fact is discovered after t steps by M, where $T_2(|ex|)/2 \leq t \leq T_2(|ex|)$, then M accepts ex. There is no other way for M to accept.

Now suppose M_e deterministically accepts L(M) within time T_1 for some particular program code e. Without loss of generality, assume that

M_e halts only when it accepts. There is some constant c such that M will get through computations of length t by M_e within $c \cdot t$ of its own steps. Since $T_2 \notin O(T_1)$, we can take $x \in \{0,1\}^*$ so that $c \cdot T_1(|ex|) < T_2(|ex|)/2$. If M_e deterministically accepts ex, then it does so within $T_1(|ex|)$ steps, and there can be no longer legal computation. By design, then, $ex \in L(M_e)$ implies $ex \notin L(M)$. If M_e does not accept ex, then there is a legal computation of every length; therefore, $ex \notin L(M_e)$ implies $ex \in L(M)$. The contradiction establishes $L(M) \notin DTIME(T_1)$. □

To diagonalize over NTIME(T) is more difficult. The problem is that discovering that M_e does not accept ex within t steps seems to require examining all legal lines of simulated computation up to t steps. This is a deterministic process which apparently may take exponentially longer than simulating a single legal line, so the diagonal construction yields $DTIME(T_2) - NTIME(T_1) \neq \emptyset$ whenever T_2 is a running time and $\log T_2 \notin O(T_1)$.

None of the above diagonal constructions actually depends on T_1, and they all produce languages over $\{0,1\}$; so Theorem 1 of Chapter Two summarizes the results.

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 $L \in DTIME(T_2)$ is a language over just $\{1\}$. If the complement of L is finite, then L is regular and $L \in DTIME(n)$. If the complement of L is infinite, on the other hand, then our convention that only acceptance time matters guarantees that $L \in \mathrm{DTIME}(T_1)$ 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, $L \in \bigcup \{\mathrm{DTIME}(T_1) \mid T_2 \notin O(T_1 \log T_1)\}$.

Strengthening the "lim inf" condition on T_1 (e. g., $T_2 \notin O(T_1 \log T_1)$) to a "lim" condition (e. g., $T_1 \log T_1 \in o(T_2)$) is one way to get separation results for languages over a one-letter alphabet. A diagonal language over $\{1\}$ can then be constructed by trying to differ on input 1^n from $M_{f(n)}$, where $f(n)$ is obtained from, say, the binary representation of the exponent of 2 in the prime factorization of n. The results are given by Theorem 2 of Chapter Two.

As far as we know, diagonalization alone yields no results better than those of Theorems 1, 2 of Chapter Two for TM acceptance time complexity. If we change our definition to take into account the time spent in nonaccepting computations, however, then we can use the diagonal technique of [MM71] to get by with "lim inf" conditions in Theorem 2. I. e., for

$\mathrm{NTIME}'(T) = \{L \mid$ L is accepted by some TM that never computes for more than $T(n)$ steps on an input of length $n\}$,

$\mathrm{DTIME}'(T) = \{L \mid$ L is accepted by some deterministic TM that never computes for more than $T(n)$ steps on an input of length $n\}$,

the set differences

$$\mathrm{DTIME}'(T_2) - \bigcup \{\mathrm{DTIME}'(T_1) \mid T_2 \notin O(T_1 \log T_1)\},$$

$$\mathrm{NTIME}'(T_2) - \bigcup \{\mathrm{DTIME}'(T_1) \mid T_2 \notin O(T_1)\},$$

$$DTIME'(T_2) - \bigcup \{NTIME'(T_1) \mid \log T_2 \notin O(T_1)\}$$

do contain languages over a one-letter alphabet if T_2 is a running time. For T a running time, we clearly have

$$NTIME(T) = NTIME'(T),$$

$$DTIME(T) = DTIME'(T),$$

so we do get languages over $\{1\}$ in the set differences of Theorem 1 of Chapter Two if we insist that the unions range only over running times T_1.

APPENDIX II

A PROGRAM CODING FOR CHAPTER TWO

Let $L^{k}_{p.c.}$ = [program], where

[program] = <u>begin program</u> [instruction]* <u>end program</u>

[instruction] = <u>begin instruction</u> [requirement]
[rewrite]
[shifts]
[next] <u>end instruction</u>
+ <u>begin instruction</u> <u>accept</u> <u>end instruction</u>

[requirement] = <u>begin requirement</u> $[\text{symbol}]^k$ <u>end requirement</u>

[rewrite] = <u>begin rewrite</u> $[\text{symbol}]^k$ <u>end rewrite</u>

[symbol] = <u>begin symbol</u> <u>tally</u>* <u>end symbol</u>

[shifts] = <u>begin shifts</u> $(\underline{\text{left}} + \underline{\text{right}} + \underline{\text{still}})^k$ <u>end shifts</u>

[next] = <u>begin next</u> [instruc. no.]* <u>end next</u>

[instruc. no.] = <u>begin instruc. no.</u> <u>tally</u>* <u>end instruc. no.</u>

<u>begin program</u>	= 00001	<u>end program</u>	= 00010
<u>begin instruction</u>	= 00011	<u>end instruction</u>	= 00100
<u>begin requirement</u>	= 00101	<u>end requirement</u>	= 00110
<u>begin rewrite</u>	= 00111	<u>end rewrite</u>	= 01000
<u>begin symbol</u>	= 01001	<u>end symbol</u>	= 01010
<u>begin shifts</u>	= 01011	<u>end shifts</u>	= 01100
<u>begin next</u>	= 01101	<u>end next</u>	= 01110
<u>begin instruc. no.</u>	= 01111	<u>end instruc. no.</u>	= 10000
<u>accept</u>	= 10001	<u>tally</u>	= 10010

<u>left</u>	= 10011	<u>right</u>	= 10100
<u>still</u>	= 10101		

For $e \in L^{k}_{p.c.}$, M_e is the TM that begins executing at the first "instruction" in e. The execution of an instruction involves the following steps:

1. Determine whether the symbols being scanned on the k tapes match the "requirement." (The blank, the 0, and the 1 match <u>tally</u>^{0}, <u>tally</u>^{1}, and <u>tally</u>^{2}, respectively, by convention.)
2. If the "requirement" was met, then overwrite the scanned symbols by the "rewrite."
3. If the "requirement" was met, then shift the heads by the "shifts."
4. If the "requirement" was met, then go next to some instruction whose position is given by "next." (Halt if there is none.)
5. If the "requirement" was not met, then go next to the next instruction in sequence. (Halt if there is none.)

It is easy to verify that this program coding satisfies Conditions 1, 2, 3 of Chapter Two.

BIOGRAPHICAL NOTE

The author was born on February 21, 1947, in Columbus, Ohio, where he attended Bexley High School (Class of 1965). As an undergraduate at M. I. T. (Class of 1969), he majored in mathematics and was inducted into the Xi Chapter of Phi Beta Kappa in 1971. He began his graduate years in the Department of Electrical Engineering as an NSF Graduate Fellow and later became a Research Assistant at Project MAC.

On January 23, 1971, he married Diane Salhanick, a Boston University graduate from Fall River, Massachusetts. On August 13, 1974, she bore him a son Gershon.

He has accepted a position as Assistant Professor of Computer Science at The Pennsylvania State University, beginning in September, 1974.

CS-TR Scanning Project
Document Control Form

Date : 3 / 14 / 96

Report # LCS-TR-137

Each of the following should be identified by a checkmark:
Originating Department:

☐ Artificial Intellegence Laboratory (AI)
☒ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:

Document Information

Number of pages: 126 (137-images)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☐ Single-sided or
☒ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☐ Unknown ☐ Other:

Check each if included with document:

☐ DOD Form ☒ Funding Agent Form ☒ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☒ Other: BIBLIOGRAPHIC DATA SHEET, 4 PAGES OF CORRECTIONS

Page Data:

Blank Pages (by page number): FOLLOW TITLE PAGE, AND PAGES 2, 3, 4, 121

Photographs/Tonal Material (by page number):

Other (note description/page number):

Description : Page Number:

IMAGE MAP: (1-126) UN#'ED TITLE PAGE, UN# BLANK, 2, UN# BLANK, 3, UN# BLK, 4, UN# BLK, 5-121, UN# BLANK.
(127-137) SCANCONTROL, COVER, FUNDING AGENT, 4 PAGES OF CORRECTION'S, TRGT'S (3)

Scanning Agent Signoff:
Date Received: 3 / 14 / 96 Date Scanned: 3 / 22 / 96 Date Returned: 3 / 25 / 96

Scanning Agent Signature: Michael W. Cook

Rev 9/94 DS/LCS Document Control Form cstrform.vsd

SOME MORE CORRECTIONS TO MAC TR-137

Joel I. Seiferas

December 30, 1974

p. 9: Delete the six lines following the statement of Corollary 15.

pp. 33-34: Delete starting with the fourth word of line 9 from the bottom of p. 33 through line 10 of p. 34.

p. 35, line 14: Change "$- 1$" to "$+ 1$".

p. 35, line 15: Change "$f(1)$" to "$f(1) + 2$".

p. 35, lines 16, 20: Change "$f(n)+n-1$" to "$f(n)+n+1$".

p. 35, line 22: Change this line to

$$"f(n) + \lceil f^{-1}\rceil (f(n)) = f(n) + n$$
$$< f(n) + n + 1$$
$$< (f(n)+1) + (n+1)$$
$$= (f(n)+1) + \lceil f^{-1}\rceil (f(n)+1),".$$

p. 36: Delete the first three lines.

p. 37, line 5: Change "$T_{M_e}(x)$" to "$\mathrm{Time}_{M_e}(x)$".