Translational Lemmas, Polynomial Time, and $(\log n)^j$-Space

Ronald V. Book

Research Report #32

September 1974

This research was supported in part by the National Science Foundation under Grant GJ-30409.

ABSTRACT

Translational lemmas are stated in a general framework and then applied to specific complexity classes. Necessary and sufficient conditions are given for every set accepted by a Turing acceptor which operates in linear or polynomial time to be accepted by a Turing acceptor which operates in space $(\log n)^j$ for some $j \geq 1$.

Introduction

There has been a great deal of effort expended in attempting to determine how different complexity classes relate to one another. In this work certain "translational" techniques appear and reappear in different guises, and the notion of a set which is "class-complete" with respect to certain reducibilities plays an important role. The purpose of this paper is to draw attention to the use of these translational techniques and complete sets, and in particular to clarify the strategy used in [1, 2] where it is shown that certain classes are distinct (i.e., not equal to one another) without indicating whether one is contained in the other.

In Section 2 we state translational lemmas in general terms in order to describe the techniques used in [1, 2, 6, 7, 9, 12, 13]. In Section 3 we illustrate the use of these techniques by studying some specific complexity classes.

One of the important underlying questions in automata-based computational complexity is that of time-space tradeoffs: if a process takes a given amount of time (space) to perform, how much space (time) does it take? Recently it has been conjectured that for a suitable bounding function f, every set accepted by a deterministic or nondeterministic Turing acceptor which operates within time bound $f(n)$ can be accepted by a deterministic Turing machine which operates within space bound $(\log f(n))^j$, for $j = 2$. In Section 3 of this paper, we establish necessary and sufficient conditions for this to occur when $f(n) = n$ and $j \geq 1$ is any integer. In particular, it is shown that this relationship exists if and only if every set accepted by a Turing machine which operates within

polynomial time can be accepted by a Turing machine which operates within a space bound which is polynomial in log n. From these conditions it is shown that certain classes defined by time bounded acceptors are not equal to certain classes defined by space bounded acceptors.

Section 1.

The classes studied here are specified by deterministic and nondeterministic multitape Turing acceptors which operate within time bounds or space bounds. The functions f used to bound the amount of time used in a Turing acceptor's computation are such that for all $n \geq 0$, $f(n) \geq n$, and are "self-computable with respect to time" in the sense that there is a deterministic Turing machine M which, upon input w, runs for precisely $f(|w|)$ steps and halts.[1] The functions f used to bound the amount of space used in a Turing acceptor's computation are such that for all $n \geq 0$, $f(n) \geq \log n$, and are "self-computable with respect to space" in the sense that there is a deterministic Turing machine M which upon input w marks precisely $f(|w|)$ consecutive tape squares and halts.[2]

For a Turing acceptor M, $L(M)$ is the set of strings accepted by M. It is assumed that the reader is familiar with the notion of multitape Turing acceptors which operate within time bounds or space bounds. Both on-line and off-line Turing acceptors are considered. An on-line acceptor reads its input from left to right. An off-line acceptor can read its input in both directions. An auxiliary pushdown acceptor [4] is an off-line Turing acceptor which has an auxiliary storage tape which is restricted by the specified space bound and a pushdown store which is unrestricted.[3]

---

[1] For a string w, $|w|$ is the length of w.

[2] Functions which are "self-computable with respect to time" or "self-computable with respect to space" are often called "linearly honest".

[3] An auxiliary pushdown acceptor can be either deterministic or nondeterministic. When considering the class of sets accepted within a specified bound, there is no difference in the computational power.

Notation. Let f be a bounding function.

(i) DTIME(f) = {L(M) | M is a deterministic on-line multitape Turing acceptor which operates within time bound f}, and NTIME(f) = {L(M) | M is a nondeterministic on-line multitape Turing acceptor which operates within time bound f}.

(ii) DSPACE(f) = {L(M) | M is a deterministic off-line multitape Turing acceptor which operates within time bound f}, and NSPACE(f) = {L(M) | M is a nondeterministic off-line multitape Turing acceptor which operates within space bound f}.

(iii) APDA(f) = {L(M) | M is an auxiliary pushdown acceptor which operates within space bound f}.

Some of the classes considered here are defined by taking a union of complexity classes where the union is taken over the positive integers. A simple notation is adopted for the most frequently studied classes with hopes of making uniform the entire notational scheme.

Notation.

(i) Let $\text{DTIME}(\text{poly}(n)) = \bigcup_{k=1}^{\infty} \text{NTIME}(n^k)$, so that DTIME(poly(n)) is the class of sets accepted by deterministic Turing acceptors which operate in polynomial time.

(ii) Let $\text{NTIME}(\text{poly}(n)) = \bigcup_{k=1}^{\infty} \text{NTIME}(n^k)$, so that NTIME(poly(n)) is the class of sets accepted by nondeterministic Turing acceptors which operate in polynomial time.

(iii) Let $\text{DSPACE}(\text{poly}(\log n)) = \bigcup_{k=1}^{\infty} \text{DSPACE}((\log n)^k)$, and let $\text{APDA}((\text{poly}(\log n)) = \bigcup_{k=1}^{\infty} \text{APDA}((\log n)^k)$.

The class of sets accepted by deterministic Turing acceptors which operate within polynomial time is DTIME(poly(n)). Cobham [3] discussed the importance of the class of functions computed in polynomial time; the subclass of characteristic functions corresponds to DTIME(poly(n)). (In [1, 5, 7, 10] this class is referred to as P.) The class of sets accepted by nondeterministic Turing acceptors which operate in polynomial time is NTIME(poly(n)). Recently Cook [5] and Karp [10] have shown the importance of the class NTIME(poly(n)) in the study of concrete computational complexity. (In [1, 5, 7, 10] this class is referred to as NP.)

In [4] it is shown that for any bounding function f, APDA(f) = $\bigcup_{c>0}$ DTIME(2^{cf}). Thus, APDA(log n) = DTIME(poly(n)). Here the class APDA(poly(log n)) = $\bigcup_{j=1}^{\infty}$ DTIME($n^{(\log n)^j}$) is also of interest.

There are several well known results concerning time-space tradeoffs. In particular, for any bounding function f it is known that DTIME(f) $\subseteq$ NTIME(f) $\subseteq$ DSPACE(f) $\subseteq$ NSPACE(f) $\subseteq$ APDA(f) = $\bigcup_{c>0}$ DTIME(2^{cf}). It is not known which of these inclusions is proper, although at least one must be proper since DTIME(f) $\subsetneq$ $\bigcup_{c>0}$ DTIME(2^{cf}). Here the classes DTIME(poly(n)) and NTIME(poly(n)) are compared to the classes DSPACE(poly(log n)), APDA(poly(log n)), and for any $j \geq 1$, DSPACE($(\log n)^j$), NSPACE($(\log n)^j$), and APDA($(\log n)^j$). Recall that for any f, NSPACE(f) $\subseteq$ DSPACE(f^2) [13] (where $f^2(n) = (f(n))^2$). Thus, DSPACE(poly(log n)) = $\bigcup_{k=1}^{\infty}$ NSPACE($(\log n)^k$).

It should be noted that the methods used here can be applied to other classes specified by subelementary bounds.

Section 2.

In this section we describe the translational lemmas.

The scheme that we use is based on the notion of "reducibility" studied in recursive function theory [11]. We use the notion of "$\mathscr{C}$-reducibilities" for a class $\mathscr{C}$ of functions, that is, we specify that the reducibilities come from a given class $\mathscr{C}$ of functions. We use the notion of a set which is "complete for a class with respect to $\mathscr{C}$-reducibilities," emphasizing that completeness depends on the type of reducibility used. Finally, we define the notion of a class "closed under $\mathscr{C}$-reducibilities."

Definition. Let $\mathscr{C}$ be a class of functions.

(i) For $f \in \mathscr{C}$, a set L_1 is reducible to L_2 via f if $L_1 \subseteq$ domain(f) and $f^{-1}(L_2) = L_1$.

(ii) A class $\mathscr{L}_1$ of sets is $\mathscr{C}$-reducible to a class $\mathscr{L}_2$ of sets if $\mathscr{L}_1 \subseteq \mathscr{C}^{-1}(\mathscr{L}_2)$, i.e., for every $L_1 \in \mathscr{L}_1$ there exist $L_2 \in \mathscr{L}_2$ and $f \in \mathscr{C}$ such that L_1 is reducible to L_2 via f.

(iii) Let $\mathscr{L}$ be a class of sets. A set L_0 is $\mathscr{C}$-complete for if $L_0 \in \mathscr{L}$ and $\mathscr{L}$ is $\mathscr{C}$-reducible to $\{L_0\}$.

(iv) A class $\mathscr{L}$ of sets is closed under $\mathscr{C}$-reducibilities if $\mathscr{C}^{-1}(\mathscr{L}) \subseteq \mathscr{L}$.

These notions are by no means new. We define them here in order to emphasize the dependence on the type of reducibility used. Further, the notion of a class being "closed under $\mathscr{C}$-reducibilities"

underlies many of the arguments in the literature regarding "complete" sets for various complexity classes. However, this idea has not been made explicit previously; we believe that this is a useful concept. See [1, 2, 5, 7, 9, 10, 13] for examples.

The first translational lemma follows immediately from the definitions above. Its statement represents an attempt to abstract the use of translational techniques in [1, 2, 5, 9, 10, 12].

Lemma 2.1. Let $\mathscr{L}_1$, $\mathscr{L}_2$, and $\mathscr{L}_3$ be classes of sets, and let $\mathscr{C}$ be a class of functions. Suppose that $\mathscr{L}_1$ is $\mathscr{C}$-reducible to $\mathscr{L}_2$ and that $\mathscr{L}_3$ is closed under $\mathscr{C}$-reducibilities. If $\mathscr{L}_2 \subseteq \mathscr{L}_3$, then $\mathscr{L}_1 \subseteq \mathscr{L}_3$.

If $\mathscr{L}_2$ is a class which is closed under $\mathscr{C}$-reducibilities and $\mathscr{L}_1$ is a class containing a set L_0 which is $\mathscr{C}$-complete for $\mathscr{L}_1$, then $\mathscr{L}_1 \subseteq \mathscr{L}_2$ if and only if $L_0 \in \mathscr{L}_2$. This has been the main use of "class-complete" sets in the comparison of complexity classes found in the literature. See [2, 5, 7, 10, 13] for examples. We extend this technique to include its use in [1, 2].

Lemma 2.2. Let $\mathscr{C}$ be a class of functions. Let Ω be a class of sets such that there exists L_0 which is $\mathscr{C}$-complete for Ω. Let $\mathscr{L}$ be a class of sets such that for some index set I, $\mathscr{L} = \bigcup_{i \in I} \mathscr{L}_i$ where each $\mathscr{L}_i$ is a class of sets which is closed under $\mathscr{C}$-reducibilities. Then the following are equivalent:

(i) there is some i such that $L_0 \in \mathscr{L}_i$;

(ii) there is some i such that $\Omega \subseteq \mathscr{L}_i$;

(iii) $\Omega \subseteq \mathscr{L}$.

Proof. Since L_0 is $\mathscr{C}$-complete for Ω, for every $L \in \Omega$ there exists $f \in \mathscr{C}$ such that L is reducible to L_0 via f. If there is some i such that $L_0 \in \mathscr{L}_i$, then since $\mathscr{L}_i$ is closed under $\mathscr{C}$-reducibilities, this implies that every $L \in \Omega$ is in $\mathscr{L}_i$, so that (i) implies (ii). Since $\mathscr{L} = \bigcup_{i \in I} \mathscr{L}_i$, (ii) implies (iii) and (iii) implies (i). □

Lemma 2.3. Let $\mathscr{C}$ be a class of functions. Let Ω be a class of sets such that there exists L_0 which is $\mathscr{C}$-complete for Ω. Let $\mathscr{L}$ be a class of sets such that for some index I, $\mathscr{L} = \bigcup_{i \in I} \mathscr{L}_i$ where each $\mathscr{L}_i$ is a class of sets which is closed under $\mathscr{C}$-reducibilities. If for all $i \in I$, $\mathscr{L}_i \neq \mathscr{L}$, then $\Omega \neq \mathscr{L}$.

Proof. If $\Omega \subseteq \mathscr{L}$, then by Part (ii) of Lemma 2.2 there is some i such that $\Omega \subseteq \mathscr{L}_i \subsetneq \mathscr{L}$. Hence, $\Omega \neq \mathscr{L}$. □

The notions specified above have been used widely in the study of complexity classes. In particular, Lemma 3 describes the strategy used in [1, 2] (and in Section 3 of this paper) where it is shown that certain classes are distinct without showing that one is or is not contained in the other.

Section 3.

In this section we develop several results concerning the classes NTIME(poly(n)), NTIME(n^j) for $j \geq 1$, DSPACE(poly(log n)), etc. These results are established for their own sake as well as providing examples in the use of the translational lemmas of Section 2.

First we describe the classes of reducibilities that we use.

Notation.

(i) Let Π be the class of all functions f of the following form: for some finite alphabet Σ, some $c \notin \Sigma$, and some constant $k \geq 1$, f is defined for all $w \in \Sigma^*$ by $f(w) = wc^m$ where $|wc^m| = |w|^k$, i.e., $m = |w|^k - |w|$.

(ii) Let $\mathscr{F}$ be the class of all homomorphisms between free monoids, i.e., $f \in \mathscr{F}$ if and only if there exist finite alphabets Σ and Δ such that $f: \Sigma^* \to \Delta^*$ is a function with the property that $f(e) = e$ and for all $n \geq 1$ and all $a_1, \ldots, a_n \in \Sigma$, $f(a_1 \cdots a_n) = f(a_1) \cdots f(a_n)$.

The class Π is a subclass of the class of functions computed by Turing machines in polynomial time. In particular, every function in Π can be computed by an on-line deterministic Turing machine which operates in polynomial time and log n space. Notice that these functions are one-to-one. One uses such a function to "pad" the length of a string by an amount which is a polynomial in the length of the string being padded.

The class $\mathscr{F}$ has been used for many purposes in automata and formal language theory. A class is closed under $\mathscr{F}$-reducibilities if and only if it is closed under inverse homomorphism.

In order to compare classes defined by time-bounds and space-bounds, we need to show that the space-bounded classes are closed under Π-reducibilities and under $\mathcal{F}$-reducibilities. We accomplish this in the following lemmas.

Lemma 3.1. Each of the following classes is closed under Π-reducibilities: DSPACE(poly(log n)), APDA(poly(log n)), and for every $j \geq 1$, DSPACE($(\log n)^j$), NSPACE($(\log n)^j$), and APDA($(\log n)^j$).

Proof. We give a proof for the case of DSPACE($(\log n)^j$) for fixed j, the proofs for the cases NSPACE($(\log n)^j$) and APDA($(\log n)^j$) being the same.

Let L_1 and L_2 be two languages such that for some $f \in \Pi$, L_1 is reducible to L_2 via f. Thus, for some finite alphabet Σ, some symbol $c \notin \Sigma$, and some integer $k \geq 1$,

(i) $f: \Sigma^* \to (\Sigma \cup \{c\})^*$ is defined by $f(w) = wc^m$ where $|wc^m| = |w|^k$ for every $w \in \Sigma^*$;

(ii) $L_1 \subseteq \Sigma^*$; and

(iii) for all $w \in \Sigma^*$, $w \in L_1$ if and only if $f(w) \in L_2$.

Suppose that $L_2 \in$ DSPACE($(\log n)^j$), so that there is a deterministic off-line Turing acceptor M_2 such that $L(M_2) = L_2$ and M_2 operates within space bound $(\log n)^j$.

We must show that $L_1 \in$ DSPACE($(\log n)^j$). From M_2, construct a deterministic off-line Turing acceptor M_1 which behaves as follows. Upon input $w \in \Sigma^*$, M_1 first writes $|w|^k$ in binary on part of its storage tape (this takes $\log |w|^k$ tape squares), and then simulates M_2's computation on wc^m, where $|wc^m| = |w|^k$. M_1 performs

this simulation by reading w on its own input tape and simulating M_2's reading of c^m on that part of its storage tape which records $|w|^k$. That is, M_1 performs this simulation by reading w on its own input tape and simulating M_2's action on w. When M_2 wishes to move to the right of w in order to read c's, M_1 simulates the read head on M_2's input tape by keeping track of the position of M_2's head relative to the rightmost letter of w, on that part of its storage tape which records $|w|^k$ in binary, and simulates the activity on M_2's storage tapes on its own storage tapes. When M_2 moves back and forth within the c's, M_1 keeps track of the position of the head on M_2's input tape on that portion of M_1's storage tape which contains $|w|^k$ and simulates the activity on M_2's storage tapes on its own storage tapes.

We claim that M_1 accepts w if and only if M_2 accepts f(w), so that $L(M_1) = L_1$. Further, M_1 uses no more space on input w than M_2 did on f(w). Now M_2 operates within space bound $(\log n)^j$ where $n = |wc^m| = |w|^k$. But $(\log |w|^k)^j = k^j(\log |w|)^j$, so that M_1 operates within space bound $k^j(\log n)^j$. Since $DSPACE(k^j(\log n)^j) = DSPACE((\log n)^j)$, this means that $L_1 \in DSPACE((\log n)^j)$.

Now consider DSPACE(poly(log n)). Let L_1 and L_2 be two languages such that for some $f \in \Pi$, L_1 is reducible to L_2 via f. If $L_2 \in DSPACE(poly(\log n)) = \bigcup_{j=1}^{\infty} DSPACE((\log n)^j)$, then there is some k such that $L_2 \in DSPACE((\log n)^k)$. But we have shown that $DSPACE((\log n)^k)$ is closed under Π-reducibilities so that L_1 reduces to L_2 via f implies $L_1 \in DSPACE((\log n)^k) \subseteq DSPACE(poly(\log n))$. Thus DSPACE(poly(log n)) is closed under Π-reducibilities. Similarly, APDA(poly(log n)) is closed under Π-reducibilities. □

Lemma 3.2. Each of the following classes is closed uner $\mathscr{F}$-reducibilities: DSPACE(poly(log n)), APDA(poly(log n)), and for every $j \geq 1$, DSPACE($(\log n)^j$), NSPACE($(\log n)^j$), and APDA($(\log n)^j$).

Proof. We give a proof for the case of DSPACE($(\log n)^j$) for fixed j, the proofs for the cases NSPACE($(\log n)^j$) and APDA($(\log n)^j$) being the same.

Let L_1 and L_2 be two languages such that there is a homomorphism $h: \Sigma^* \to \Delta^*$ such that for all $w \in \Sigma^*$, $w \in L_1$ if and only if $h(w) \in L_2$, where $L_1 \subseteq \Sigma^*$ --that is, L_1 is reducible to L_2 via $h \in \mathscr{F}$. Suppose that $L_2 \in$ DSPACE($(\log n)^j$), so that there is a deterministic off-line Turing acceptor M_2 such that $L(M_2) = L_2$ and M_2 operates within space bound $(\log n)^j$. We must show that $L_1 \in$ DSPACE($(\log n)^j$). From M_2 construct a deterministic off-line Turing acceptor M_1 which behaves as follows. Upon input $a_1 \cdots a_n$, each $a_i \in \Sigma$, M_1 imitates the action of M_2 on $h(a_1 \cdots a_n) = h(a_1) \cdots h(a_n)$. Since there is a fixed bound k such that for all $a \in \Sigma$, $0 \leq |h(a)| \leq k$, M_1 can remember the entire string $h(a)$, $a \in \Sigma$, in its finite state control while imitating M_2 on $h(a)$. Now M_1 will use the same amount of space on $a_1 \cdots a_n$ as M_2 uses on $h(a_1 \cdots a_n)$. Since M_2 operates within space bound $(\log n)^j$ and $h(a_1 \cdots a_n) \leq kn$, this means that M_1 operates within space bound $(\log kn)^j \leq k^j (\log n)^j$. Thus $L_1 = L(M_1) \in$ DSPACE($k^j (\log n)^j$) = DSPACE($(\log n)^j$).

To show that DSPACE(poly(log n)) and APDA(poly(log n)) are closed under $\mathscr{F}$-reducibilities, one uses an argument just like that in Lemma 3.1. □

To compare NTIME(poly(n)) with space-bounded classes, we shall depend upon the fact that NTIME(poly(n)) is Π-reducible to NTIME(n) and that there is a set which is $\mathscr{F}$-complete for NTIME(n). These results are obtained in the following lemmas.

Lemma 3.3. The class NTIME(poly(n)) is Π-reducible to the class NTIME(n).

Proof. If $L_1 \in \text{NTIME}(\text{poly}(n)) = \bigcup_{j=1}^{\infty} \text{NTIME}(n^j)$, then there exists a $j \geq 1$ such that $L_1 \in \text{NTIME}(n^j)$, and hence, there exists a nondeterministic Turing machine M_1 such that $L(M_1) = L_1$ and such that M_1 operates within time bound n^j. Let Σ be a finite alphabet such that $L_1 \subseteq \Sigma^*$, and let $c \notin \Sigma$ be a new symbol. Let $L_2 = \{ wc^m \mid w \in L_1, \ |wc^m| = |w|^j \}$. From M_1 one can construct a nondeterministic machine M_2 to recognize L_2 as follows. Initially, M_2 reads input symbols from Σ and imitates M_1 on that initial portion of the input. While imitating M_1 on w, M_2 simultaneously checks whether the number of c's is precisely $|w|^j - |w|$. The imitation of M_1 on w takes at most $|w|^j$ steps since M_1 operates within time bound n^j, and having recorded $|w|$ while reading w, M_2 can check whether there are exactly $|w|^j - |w|$ c's in real time. Thus, M_2 can be made to operate in real time, i.e., within time bound n, so that $L_2 = L(M_2) \in \text{NTIME}(n)$. If $f: \Sigma^* \to (\Sigma \cup \{c\})^*$ is defined by $f(w) = wc^m$ where $m = |w|^j - |w|$, then $f \in \Pi$ and clearly L_1 is reducible to L_2 via f. Since L_1 was taken arbitrarily from NTIME(poly(n)), we have the result. □

Lemma 3.4. For every $j \geq 1$, there exists a language L_0 which is $\mathscr{F}$-complete for $NTIME(n^j)$.

Proof. This is established in [2]. The language given in [2] is the set of all strings of the form $\bar{a}_1 \bar{M} \bar{a}_2 \bar{M} \cdots \bar{a}_n \bar{M}$ where $\bar{M}$ encodes a nondeterministic multitape Turing acceptor which operates within time bound n^j, each $\bar{a}_i$ is the encoding of an input symbol of M, and $a_1 \cdots a_n$ is accepted by M. For further details, see [2]. □

To show that $NTIME(poly(n))$ is included in $DSPACE(poly(\log n))$, it is enough to show that $NTIME(n)$ is so included.

Theorem 3.5. The following are equivalent:

(i) $NTIME(poly(n)) \subseteq DSPACE(poly(\log n))$ (resp., $APDA(poly(\log n))$);

(ii) $NTIME(n) \subseteq DSPACE(poly(\log n))$ (resp., $APDA(poly(\log n))$);

(iii) there exists $j \geq 1$ such that $NTIME(n) \subseteq DSPACE((\log n)^j)$ (resp., $APDA((\log n)^j)$);

(iv) there exists $j \geq 1$ such that $NTIME(poly(n)) \subseteq DSPACE((\log n)^j)$ (resp., $APDA((\log n)^j)$).

Proof. We prove the result for $DSPACE(\ )$, the proof for $APDA(\ )$ being the same. It is immediate from the definitions that (i) implies (ii), (iii) implies (ii), (iv) implies (i), and (iv) implies (iii)

To prove that (ii) implies (iii), we use Lemma 2.2. By Lemma 3.4, there is a language L_0 which is $\mathscr{F}$-complete for $NTIME(n)$. By Lemma 3.2, for every $j \geq 1$, $DSPACE((\log n)^j)$ is closed under $\mathscr{F}$-reducibilities. Suppose $NTIME(n) \subseteq DSPACE(poly(\log n))$. Then by Lemma 2.2, there exists some $k \geq 1$ such that $NTIME(n) \subseteq DSPACE((\log n)^k)$.

To prove that (iii) implies (iv), we use Lemma 2.1. By Lemma 3.1, for every $j \geq 1$, $DSPACE((\log n)^j)$ is closed under Π-reducibilities. By Lemma 3.3, $NTIME(poly(n))$ is Π-reducible to $NTIME(n)$. Suppose that for some $k \geq 1$, $NTIME(n) \subseteq DSPACE((\log n)^k)$. By Lemma 2.1 this implies that $NTIME(poly(n)) \subseteq DSPACE((\log n)^k)$. □

From the equivalence of (i) and (ii) in Theorem 3.5, we see that $NTIME(poly(n)) \subseteq DSPACE(poly(\log n))$ if and only if $NTIME(n) \subseteq DSPACE(poly(\log n))$. By Lemma 3.4, there exists a language L_0 which is $\mathscr{T}$-complete for $NTIME(n)$, and by Lemma 3.2 $DSPACE(poly(\log n))$ is closed under $\mathscr{T}$-reducibilities. Thus we have the following result.

Corollary. There exists a language $L_0 \in NTIME(n)$ such that $NTIME(poly(n)) \subseteq DSPACE(poly(\log n))$ (resp., $APDA(poly(\log n))$) if and only if $L_0 \in DSPACE(poly(\log n))$ (resp., $APDA(poly(\log n))$).

The fact that the space-bounded machines were deterministic played no role in the proof of Theorem 3.5. Thus, if $NSPACE(\ )$ is substituted for $DSPACE(\ )$ throughout, the resulting statements still hold.

There are other classes which may play the role of $NTIME(n)$ in Theorem 3.5. In particular it is easy to prove that for any $j \geq 1$ the following are equivalent:

(i) $NTIME(n) \subseteq DSPACE((\log n)^j)$;

(ii) every language accepted by a simple nondeterministic Turing acceptor which operates within time bound $n \log n$ is in $DSPACE((\log n)^j)$;[4]

[4] A simple Turing machine has exactly one tape upon which input is written and work is performed, and has only one read-write head on that tape.

(iii) every language accepted by a nondeterministic on-line, one storage tape Turing acceptor which operates in linear time is in DSPACE($(\log n)^j$).

There are several results which follow from Theorem 3.5 and its proof.

Theorem 3.6. NTIME(poly(n)) $\neq$ DSPACE(poly(log n)) and NTIME(poly(n)) $\neq$ APDA(poly(log n)).

Proof. In [14] it is shown that for every $j \geq 1$, DSPACE($(\log n)^j$) $\subsetneq$ DSPACE($(\log n)^{j+1}$). Thus, for all j, DSPACE($(\log n)^j$) $\subsetneq$ DSPACE((poly(log n)). Similarly, by results in [4] and [8] it can be shown that for all j, APDA($(\log n)^j$) $\subsetneq$ APDA(poly(log n)). Thus this result follows from Theorem 3.5 and Lemma 2.3. □

Theorem 3.7. For every $j \geq 1$, NTIME(n^j) $\neq$ DSPACE(poly(log n)) and NTIME(n^j) $\neq$ APDA(poly(log n)).

Proof. As noted in the proof of Theorem 3.6 for every $j \geq 1$, DSPACE($(\log n)^j$) $\subsetneq$ DSPACE(poly(log n)). By Lemma 3.2, for every $j \geq 1$, DSPACE($(\log n)^j$) is closed under $\mathscr{F}$-reducibilities. By Lemma 3.4, for every $k \geq 1$, there is a set which is $\mathscr{F}$-complete for NTIME(n^k). From Lemma 2.3 we conclude that for every $k \geq 1$, NTIME(n^k) $\neq$ DSPACE(poly(log n)). The proof that NTIME(n^k) $\neq$ APDA(poly(log n)) is the same. □

Theorem 3.8. For all $j, k \geq 1$, NTIME(n^j) $\neq$ DSPACE($(\log n)^k$) and NTIME(n^j) $\neq$ APDA($(\log n)^k$.

Proof. Suppose that for some j, $k \geq 1$, $\text{NTIME}(n^j) \subseteq \text{DSPACE}((\log n)^k)$. Then $\text{NTIME}(n) \subseteq \text{DSPACE}((\log n)^k)$. By the equivalence of (iii) and (iv) in Theorem 3.5, this implies that $\text{NTIME}(\text{poly}(n)) \subseteq \text{DSPACE}((\log n)^k)$. Since for all j, $\text{NTIME}(n^j) \subsetneq \text{NTIME}(\text{poly}(n))$ [ ], we have $\text{NTIME}(n^j) \subsetneq \text{DSPACE}((\log n)^k)$. □

Again, if DSPACE( ) is replaced by NSPACE( ) in Theorem 3.8, then the resulting statements are true.

In Theorems 3.5-3.8 the classes specified by time bounds involve only nondeterministic acceptors. The results still hold if one considers deterministic acceptors instead of nondeterministic acceptors. However, the proofs must be altered slightly. In particular, Lemma 3.4 does not hold for deterministic acceptors. To obtain the deterministic counterparts of Theorems 3.5-3.8, one must note that there is a language $L_0 \in \text{DTIME}(n^2)$ with the property that for every $L \in \text{DTIME}(n)$ there is a function $f \in \mathcal{F}$ such that L is reducible to L_0 via f. This provides the appropriate counterpart of Lemma 3.4. We state the counterparts of Theorems 3.5-3.8 without proof.

Theorem 3.9. The following are equivalent:

(i) $\text{DTIME}(\text{poly}(n)) \subseteq \text{DSPACE}(\text{poly}(\log n))$;

(ii) $\text{DTIME}(n) \subseteq \text{DSPACE}(\text{poly}(\log n))$;

(iii) there exists $j \geq 1$ such that $\text{DTIME}(n) \subseteq \text{DSPACE}((\log n)^j)$;

(iv) there exists $j \geq 1$ such that $\text{DTIME}(\text{poly}(n)) \subseteq \text{DSPACE}((\log n)^j)$.

Theorem 3.10.

(i) DTIME(poly(n)) $\neq$ DSPACE(poly(log n)) ;

(ii) for every $j \geq 1$, DTIME(n^j) $\neq$ DSPACE(poly(log n)) ;

(iii) for every j, $k \geq 1$, DTIME(n^j) $\neq$ DSPACE($(\log n)^k$).

The results in Theorems 3.6, 3.7, 3.8 and 3.10 are of the same form. They state that two classes, specified in different ways, are not equal. But no information is given in the statement of the result or in the proof as to whether one class is a subclass of the other.

In [1] it is shown that NTIME(poly(n)) = DTIME(poly(n)) if and only if NTIME(n) $\subseteq$ DTIME(poly(n)). The proof (rephrased in terms of the concepts used here) is based on Lemma 3.3 and the observation that DTIME(poly(n)) is closed under Π-reducibilities. Greibach [7] has extended this result to show that NTIME(poly(n)) = DTIME(poly(n)) if and only if DTIME(poly(n)) contains every language of the form $h(L_1 \cap L_2)$ where L_1 and L_2 are linear context-free languages and h is a nonerasing homomorphism. The languages used in Greibach's proof are linear context-free and in DSPACE(log n). This leads to the following observation.

Proposition. For any $j \geq 1$, if DSPACE($(\log n)^j$) (resp., NSPACE($(\log n)^j$), APDA($(\log n)^j$)) contains the image under nonerasing homomorphism of DSPACE(log n), then NTIME(poly(n)) $\subseteq$ DSPACE($(\log n)^j$) (resp., NSPACE($(\log n)^j$), APDA($(\log n)^j$)).

Some of the inclusion relations between the classes studied here are illustrated in Figure 1. Some of the statements of inequality of classes are given in Figure 2. The statements that a class specified by time-bounded machines is not equal to a class specified by space-bounded machines are new results.

ACKNOWLEDGMENT

I wish to thank Celia Wrathall for many helpful comments on this work.

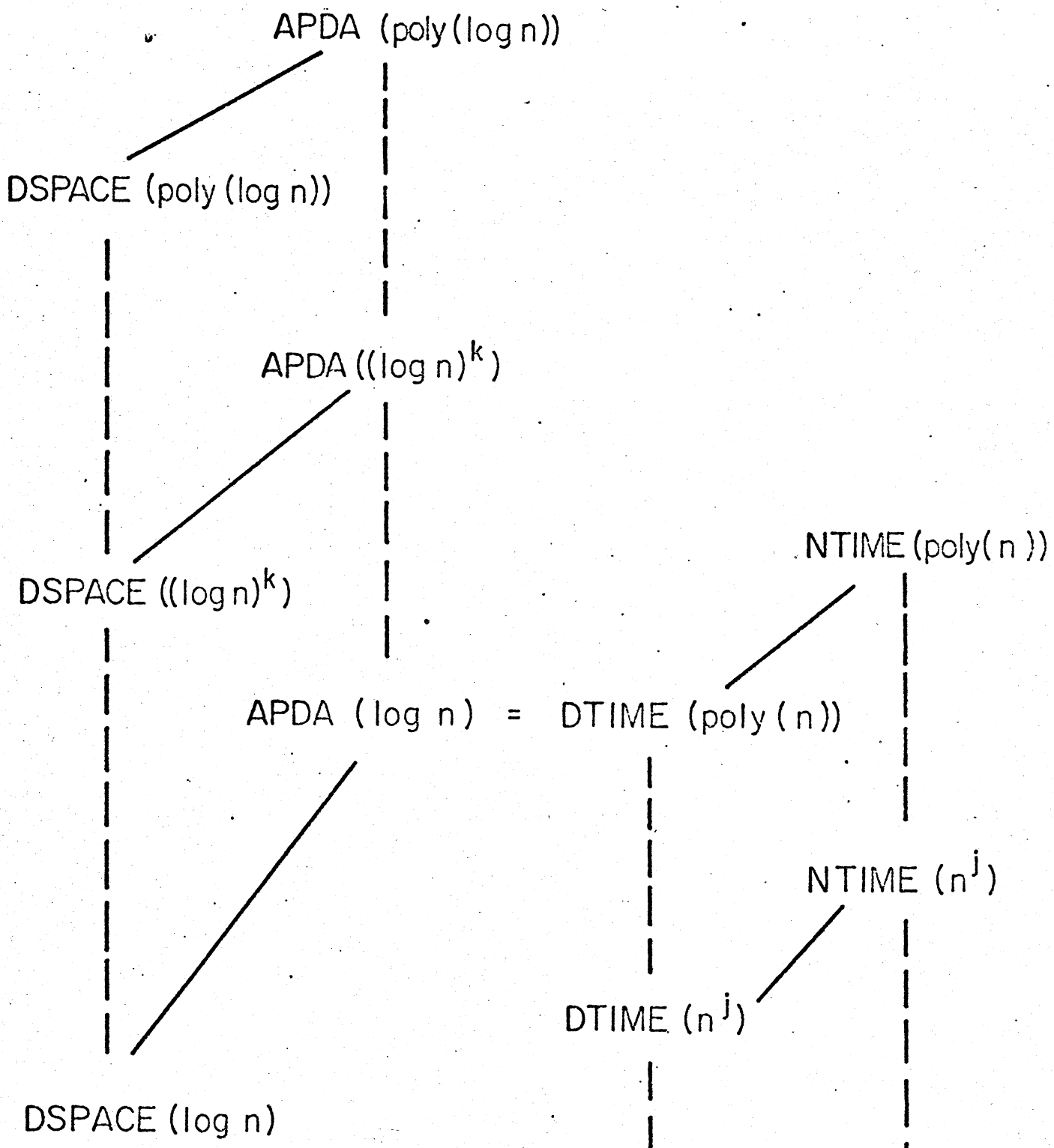

FIGURE 1

$\mathcal{L}_1$ — $\mathcal{L}_2$ INDICATES THAT $\mathcal{L}_1 \subseteq \mathcal{L}_2$

$\mathcal{L}_1$ - - $\mathcal{L}_2$ INDICATES THAT $\mathcal{L}_1 \subsetneq \mathcal{L}_2$

	APDA(poly(log n))	APDA((log n)^k, k>1	APDA(log n)	DSPACE(poly(log n))	DSPACE((log n)^k)	DSPACE(log n)	NTIME(poly(n))	NTIME(n^j), j>1	NTIME(n)	DTIME(poly(n))	DTIME(n^j)	DTIME(n)
APDA(poly(log n))												
APDA((log n)^k, k>1	≠											
APDA(log n)	≠	≠										
DSPACE(poly(log n))	?	≠	≠									
DSPACE((log n)^k)	≠	?	?	≠								
DSPACE(log n)	≠	≠	?	≠	≠							
NTIME(poly(n))	≠	?	?	≠	?	?						
NTIME(n^j), j>1	≠	≠	≠	≠	≠	≠	≠					
NTIME(n)	≠	≠	≠	≠	≠	≠	≠	≠				
DTIME(poly(n))	≠	≠	=	≠	?	?	?	≠	≠			
DTIME(n^j)	≠	≠	≠	≠	≠	≠	≠	?	?	≠		
DTIME(n)	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	≠	

FIGURE 2

$\mathcal{L}_1 \; ? \; \mathcal{L}_2$ INDICATES THAT IT IS NOT KNOWN WHETHER $\mathcal{L}_1 = \mathcal{L}_2$ OR $\mathcal{L}_1 \neq \mathcal{L}_2$.

References

1. R. Book, On languages accepted in polynomial time, SIAM J. Computing, 1 (1972), 281-287.

2. R. Book, Comparing complexity classes, J. Computer System Sci., 9 (1974).

3. A. Cobham, The intrinsic computational difficulty of functions, Proc. 1964 Congress for Logic, Math., and Phil. of Sci., North-Holland, 1964, 24-30.

4. S. Cook, Characterizations of pushdown machines in terms of time-bounded computers, JACM, 18 (1971), 4-18.

5. S. Cook, The complexity of theorem-proving procedures, Proc. Third ACM Symposium on Theory of Computing, (1971), 151-158.

6. S. Cook, A hierarchy for nondeterministic time complexity, Proc. Fourth ACM Symposium on Theory of Computing, (1972), 187-192.

7. S. Greibach, The hardest context-free language, SIAM J. Computing, 2 (1973), 304-310.

8. J. Hartmanis and R. Stearns, On the computational complexity of algorithms, Trans. Amer. Math. Soc., 117 (1965), 285-306.

9. O. Ibarra, A note concerning nondeterministic tape complexities, JACM, 19 (1972), 608-612.

10. R. Karp, Reducibilities among combinational problems, in R. Miller and J. Thatcher (eds.) Complexity of Computer Computations, Plenum Press, 1972, 85-104.

11. H. Rogers, Theory of Recursive Functions and Effective Computability, McGraw-Hill, 1967.

12. S. Ruby and P. C. Fischer, Translational methods and computational complexity, Conf. Record IEEE Sicth Annual Symp. on Switching Circuit Theory and Logical Design, (1965), 173-178.

13. W. Savitch, Relationships between nondeterministic and deterministic tape complexity, J. Comput. System Sci., 4 (1970), 177-192.

14. R. Stearns, J. Hartmanis, and P. Lewis, Hierarchies of memory limited computations, Conf. Record IEEE Sixth Annual Symposium on Switching Circuit Theory and Logical Design, (1965), 179-190.

Footnotes.

1. For a string w, $|w|$ is the length of w.

2. Functions which are "self-computable with respect to time" or "self-computable with respect to space" are often called "linearly honest."

3. An auxiliary pushdown acceptor can be either deterministic or nondeterministic. When considering the class of sets accepted within a specified bound, there is no difference in the computational power.

4. A simple Turing machine has exactly one tape upon which input is written and work is performed, and has only one read-write head on that tape.