ILLiad Request Printout

Transaction Number: 427393
Username: 100934397 Name: William Gasarch
ISSN/ISBN:
NotWantedAfter: 11/09/2009
Accept Non English: Yes
Accept Alternate Edition: No
Request Type: Article - Article

Loan Information

LoanAuthor:
LoanTitle:
LoanPublisher:
LoanPlace:
LoanDate:
LoanEdition:
NotWantedAfter: 11/09/2009

Article Information

PhotoJournalTitle: SIAM journal of Computing
PhotoJournalVolume: 16
PhotoJournalIssue:
Month:
Year: 1987
Pages: 613-627
Article Author: Gasarch
Article Title: Oracles for Deterministic versus Alternating classes

Citation Information

Cited In:
Cited Title:
Cited Date:
Cited Volume:
Cited Pages:

OCLC Information

ILL Number:
OCLC Number:
Lending String: Direct Request
Original Loan Author:
Original Loan Title:
Old Journal Title:
Call Number: UMCP EPSL Periodical Stacks QA76.S555 v.16 (1987)
Location:

Notes

9/11/2009 7:42:04 AM System 1. No Matching Bib/2. No ISBN, ISSN, or OCLCNo in request.

SIAM J. COMPUT.
Vol. 16, No. 4, August 1987

ORACLES FOR DETERMINISTIC VERSUS ALTERNATING CLASSES*

WILLIAM GASARCH†

Abstract. We construct oracles that force all possible relationships between NP and $EXP_k = DTIME(2^{O(n^k)})$. We generalize these results to obtain a theorem about oracles that force relationships between deterministic and nondeterministic (and alternating) classes, from which many corollaries follow. The corollaries are interesting because we compare a powerful type of machine (e.g., nondeterministic, alternating) to a less powerful type of machine that can use more time. One of our corollaries is that the result $DTIME(n \log^*(n)) \subseteq \Sigma_2 - TIME(n)$ does not relativize.

Key words. oracle, NP, alternation

AMS(MOS) subject classifications. 68Q15, 03O15

1. Introduction.

In a series of papers Book [7], [8] develops and applies padding techniques to a variety of problems. All the results obtained are inequalities (e.g., $NP \neq EXP_k = DTIME(2^{O(n^k)})$) without additional information; it is not known whether one class contains another. All the techniques relativize; hence oracle results demonstrate their limitation (see [2]). In particular, we show that such techniques will not suffice to give additional information about the nature of the inequalities Book obtained.

A technique or theorem that remains valid or true when all the machines are replaced by machines with oracles is said to *relativize*. Virtually all techniques in recursion theory relativize, since usually such techniques are only concerned with the input-output behavior of the machines, and could just as well be dealing with an oracle machine's input-output behavior. Baker, Gill and Solovay [2] showed that such techniques will not suffice to solve $P = ?NP$ by showing there exists oracles A and B such that $P^A \neq NP^A$ and $P^B = NP^B$. Oracles have been used to show that recursion theoretic techniques will not suffice to resolve questions about the classes R [29], U [15], $NP \cap \text{co-}NP$ [2], Σ_2^p [3], ZPP [21], PP, $\#P$ [1], exponential time [14], [17], various parallel classes [35], Σ_n^p, and $PSPACE$ [36] (easier proofs of the results in [36] are in [16]). Other topics shown hard to explore by purely recursion theoretic means include the Berman–Hartmanis Conjecture [23], P-immunity, P-simplicity [5], [18], complete sets for R or $NP \cap \text{co-}NP$ [32], and the existence of sparse sets in $NP - P$ [24].

We construct oracles that induce all possible relationships (with immunity) between NP and EXP_k. Other results involving exponential time, of a different nature, appear in [17]. Some of the oracle constructions we use for NP and EXP_k have appeared in [12]; however, here we modify them to obtain results about other complexity classes. A general theorem about oracles that force relationships between deterministic and nondeterministic (or alternating) classes is presented. This theorem has many corollaries. These corollaries involve comparing two complexity classes under an oracle. The comparisons are interesting because we compare a powerful type of machine (e.g., alternating) to a less powerful type of machine that can use more time. The classes considered include EXP_k, $D\#P$ (machines with a $\#P$ oracle (see [33],

* Received by the editors December 24, 1985; accepted for publication (in revised form) September 30, 1986. This work is part of the author's Ph.D. dissertation. This work was supported by National Science Foundation grants MCS-80-05-05386, MCS-82-00269 and MCS-80-10707.

† Department of Computer Science, University of Maryland, College Park, Maryland 20742.

[34])), $DTIME(c^n)$, $NTIME(d^n)$, $DTIME(n^c)$, $BTIME(n^d)$, and alternating classes [19]. One of our corollaries is that the result $DTIME(n \log^*(n)) \subseteq \Sigma_2\text{-}TIME(n)$ [28] does not relativize.

In many oracle constructions in the literature, a set A is constructed to force $L^A \notin K^A$, where $L^{()}$ is an oracle dependent language and $K^{()}$ is a relativized complexity class. However, it is possible that L^A can still be approximated by a language in K^A, i.e., some infinite subset of L^A is in K^A. Oracles for which this does not happen were shown to exist in [6] and constructed recursively in [18], [31]. We borrow their terminology. (In the definitions below: L is a language; $\bar{L}$ is the complement of L, i.e., $\Sigma^* - L$; K, K_1, and K_2 are complexity classes.)

DEFINITION. L is *K-immune*, if L is infinite and has no infinite subset in K.

DEFINITION. L is *K-bi-immune* if both L and $\bar{L}$ are K-immune.

DEFINITION. $K_1 \subsetneqq K_2$ *with immunity* if $K_1 \subsetneqq K_2$, and there exists a language $L \in K_2$ that is K_1-immune.

DEFINITION. $K_1 \perp K_2$ *with immunity* (read as "K_1 is incomparable to K_2 with immunity") if there exist languages $L_1 \in K_1$, $L_2 \in K_2$, such that L_1 is K_2-immune and L_2 is K_1-immune.

DEFINITION. $K_1 \subsetneqq K_2$ *with bi-immunity* if there exists a language $L \in K_2$ that is K_1-bi-immune and $K_1 \subsetneqq K_2$.

DEFINITION. $K_1 \perp K_2$ *with bi-immunity* (read as "K_1 is incomparable to K_2 with bi-immunity") if there exist languages $L_1 \in K_1$, $L_2 \in K_2$, such that L_1 is K_2-bi-immune and L_2 is K_1-bi-immune.

We assume throughout that we have enumerations of:

(1) all oracle polynomial time bounded Turing Machines, $\{P_1^{()}, P_2^{()}, \cdots\}$, where machine $P_i^{()}$ takes time at most $p_i(x) = i + |x|^i$ on input x;

(2) all oracle nondeterministic polynomial time bounded Turing Machines $\{NP_1^{()}, NP_2^{()}, \cdots\}$, where machine $NP_i^{()}$ takes at most time $p_i(x) = i + |x|^i$ on input x along any path;

(3) (for a fixed k) all oracle $DTIME(2^{O(n^k)})$ time bounded Turing Machines, $\{E_1^{()}, E_2^{()}, \cdots\}$, where machine $E_i^{()}$ takes at most time $h_i(x) = 2^{i|x|^k}$ on input x.

We use the symbol "$L(M)$" to denote the language recognized by that machine M. Throughout this paper Σ is the fixed alphabet $\{0, 1\}$. The symbol "$\langle , \rangle$" denotes an easily computed mapping from $\Sigma^* \times \Sigma^*$ to N or $\Sigma^* \times \Sigma^*$ to Σ^*; usage will be clear from context, e.g., "$\langle i, j \rangle < s$" means that "$\langle i, j \rangle$" is a number which is less than s; and "$\langle i, x \rangle w$" means that $\langle i, x \rangle$ is a string and represents the concatenation of the string $\langle i, x \rangle$ with the string w. If a number appears as an argument to $\langle , \rangle$ then we mean its binary representation. If w_1 and w_2 are strings, then $|\langle w_1, w_2 \rangle| > \text{Max}(|w_1|, |w_2|)$. If n_1 and n_2 are numbers then $\langle n_1, n_2 \rangle > \text{Max}(n_1, n_2)$. The symbol $\langle , , \rangle$ denotes an easily computed mapping from $\Sigma^* \times \Sigma^* \times \Sigma^*$ to Σ^*.

2. *NP* versus EXP_k. One difference between NP and $EXP_k = DTIME(2^{O(n^k)})$ ($k \geqq 1$ arbitrary but fixed) is that NP is closed under "polynomial padding" while EXP_k is not. Book [7] exploited this difference to prove $NP \neq EXP_k$. His proof relativizes, i.e., it can be modified easily to show that for any oracle A, $NP^A \neq EXP^A$. This has also been observed by Dekhtyar [10].

The proof of $NP \neq EXP_k$ does not indicate in what way NP and EXP_k are related. $NP \subsetneqq EXP_k$, $EXP_k \subsetneqq NP$, and $NP \perp EXP_k$ are possibilities. Dekhtyar [10] claims (without proof) that oracles A, B, and C exist such that $NP^A \subsetneqq EXP_1^A$, $EXP_1^B \subsetneqq NP^B$, and $NP^C \perp EXP_1^C$. This shows that any technique that relativizes will not suffice to resolve how NP and EXP_1 relate.

It is possible that every s
constructing oracles that induc
and EXP_k, we show that any
of these questions.

2.1. $P^A \subsetneqq NP^A \subsetneqq EXP_k^A$. (
complexity theory was given
existence of an oracle A such t
EXP_k^A by the relativized versi
an oracle A such that $P^A \subsetneqq NP^A$
with immunity. Clearly this im

The following proof is an
argument proof of the same th

THEOREM 1. *There exists*
both proper inclusions hold wit

Proof. Let

$$L_1^A = \{x | \exists_y |y| =$$

$$L_2^A = \{0^n | 1^{2^n} \in$$

Note that all strings in L
ensure that L_1^A, L_2^A, and the co
strings.

Clearly we have $L_1^A \in NP$
infinite and contains no infinit
ments:

$$RP_i\text{: } L(P_i^A)$$

$$RNP_i\text{: } L(N$$

The infinitude of $L_1^A(L_2^A)$
as a formal requirement.

We need a function to hel
an injective function com
(x_1, x_2, x_3), $c(x_1, x_2, x_3) > x_i$.

We code $NP^A \subseteq EXP_1^A$ vi

For all i, for all x

$$NP_i^A(x)$$

(Note that $1x$ is interpret

We construct A in stages.
the set of strings placed into
at no later stage may an eleme
guarantees that requirements
and that $A = \bigcup_{s=0}^{\infty} A_s$ is recurs

CONSTRUCTION

Stage 0: $A_0 \leftarrow \varnothing$, $r(0) \leftarrow 0$.

Stage $s+1$:

It is possible that every set in NP has an infinite EXP_k subset or vice versa. By constructing oracles that induce all possible relationships, with immunity, between NP and EXP_k, we show that any technique that relativizes will not suffice to resolve any of these questions.

2.1. $P^A \subsetneq NP^A \subsetneq EXP_k^A$. One of the first and most basic oracle constructions in complexity theory was given in Baker, Gill and Solovay [2], where they showed the existence of an oracle A such that $P^A = NP^A$. For such an oracle we have $NP^A = P^A \subsetneq EXP_k^A$ by the relativized version of the Time Hierarchy Theorem [20]. We construct an oracle A such that $P^A \subsetneq NP^A \subsetneq EXP_1^A$, and, moreover, we obtain these inequalities with immunity. Clearly this implies the same result for EXP_k^A, $k > 1$, instead of EXP_1^A.

The following proof is an arithmetic forcing argument similar to [24]. A priority argument proof of the same theorem is in [14].

THEOREM 1. *There exists a recursive oracle A such that $P^A \subsetneq NP^A \subsetneq EXP_1^A$, and both proper inclusions hold with immunity.*

Proof. Let

$$L_1^A = \{x \mid \exists_y |y| = |x|,\ xy \in A,\ |xy| \text{ is not a power of two}\},$$

$$L_2^A = \{0^n \mid 1^{2^n} \in A\}.$$

Note that all strings in L_1^A are of a length that is not a power of two. This will ensure that L_1^A, L_2^A, and the coding of NP^A into EXP_1^A are affected by disjoint sets of strings.

Clearly we have $L_1^A \in NP^A$ and $L_2^A \in EXP_1^A$. We need to ensure that $L_1^A (L_2^A)$ is infinite and contains no infinite subset in $P^A (NP^A)$. We state this in terms of requirements:

$$RP_i\colon L(P_i^A) \cap L_1^A \text{ infinite} \to L(P_i^A) \cap \overline{L_1^A} \neq \varnothing,$$

$$RNP_i\colon L(NP_i^A) \cap L_2^A \text{ infinite} \to L(NP_i^A) \cap \overline{L_2^A} \neq \varnothing.$$

The infinitude of $L_1^A (L_2^A)$ follows from the construction and need not be stated as a formal requirement.

We need a function to help code NP^A into EXP_1^A. Let $c\colon N \times N \times N \to 2N+1$ be an injective function computable in polynomial time such that for all (x_1, x_2, x_3), $c(x_1, x_2, x_3) > x_i$.

We code $NP^A \subseteq EXP_1^A$ via:

For all i, for all x

$$NP_i^A(x) \text{ accepts in } t \text{ steps iff } 0^{c(i,1x,t)} \in A.$$

(Note that $1x$ is interpreted as a binary number.)

We construct A in stages. At the end of each stage we define A_s and $r(s)$: A_s is the set of strings placed into A through stage s, $r(s)$ is a "restraint function" in that at no later stage may an element w, $|w| < r(s)$, be placed into A. The restraint function guarantees that requirements which are declared satisfied at stage s remain satisfied; and that $A = \bigcup_{s=0}^{\infty} A_s$ is recursive.

CONSTRUCTION

Stage 0: $A_0 \leftarrow \varnothing$, $r(0) \leftarrow 0$.

Stage $s+1$:

There are six substages. To avoid notational problems the phrase "$A_s \leftarrow A_s \cup \{w\}$" means add w to A_s. If "A_s" is referenced later we mean the updated version. The phrase "decide a string" means to decide whether it is in A or not. "Restrain a string" means to decide that the string will never enter A.

We pick a length n so that later (in substage c) we can run $P_i^{A_s}$ $(0 \leqq i \leqq s/2)$ on some string of length $n/2$ and diagonalize (to satisfy some RP_i). Since the diagonalization will involve strings of length n, we pick n large (so the number of strings of length n is large) and we are careful to not put too many strings of length n into A_s in substages a and b.

Let $n_s = n$ be the least even number, not a power of 2, such that $n > r(s)$ and $p_s^2(n) < 2^{n/2}$.

Substage a: (Decide all strings of length k, $r(s) < k < n$). For each k, $r(s) < k < n$, if $k = c(i, 1x, t)$ for some (i, x, t), then run $NP_i^{A_s}(x)$ for t steps. If it accepts then set $A_s \leftarrow A_s \cup \{0^{c(i,1x,t)}\}$. Since $t < c(i, 1x, t)$, placing $0^{c(i,1x,t)}$ into A does not affect the computation that this string codes.

Substage b: (Decide all strings of length k, $n < k < p_s(n)$). We want to do a similar construction as in substage a, but there is one glitch: strings of length $n < k$ may enter A at a later time and invalidate a computation which we coded. To avoid this we decide certain strings of length n to preserve computations. The key point is that when we finish coding there will still be enough strings of length n undecided to help satisfy requirements RP_i, $0 \leqq i \leqq s/2$.

For each k, $n < k \leqq p_s(n)$, such that $k = c(i, 1x, t)$ for some (i, x, t) run $NP_i^{A_s}(x)$. If it accepts then $A_s \leftarrow A_s \cup \{0^{c(i,1x,t)}\}$ and we restrain from A all strings of length n that were queried.

If it rejects then we search for a set of unrestrained strings B (of length n), such that $NP_i^{A_s \cup B}(x)$ accepts. If we find such a B then, for a particular accepting path, restrain all elements queried of length n that are not in $A_s \cup B$, and set

$$A_s \leftarrow A_s \cup \{x \mid x \in B \text{ and } x \text{ is queried on that path}\} \cup \{0^{c(i,1x,t)}\}.$$

Note that the number of elements of length n which are decided is less than $t < p_s(n)$.

If none accept then we do not put $0^{c(i,1x,t)}$ into A because (key point!) whichever elements of length n enter A, NP_i^A will not accept, since we tried all possibilities.

Substage c: (Decide remaining strings of length n, and satisfy RP_i). Note that the number of strings of length n not yet decided is less than (number of machines run during substage b) $\times$ (max number of strings queried in such a run) $< p_s(n) \times p_s(n) = p_s^2(n) < 2^{n/2}$.

For each x, $|x| = n/2$, let $D_x = \{xy : |y| = n/2\}$. Since there are less than $2^{n/2}$ decided strings, one of the D_x must contain only undecided strings. Let D_{x_0} be the least such (for definiteness). Note that $x_0 \in L_1^A$ iff $D_{x_0} \cap A \neq \emptyset$. Run $P_i^{A_s}(x_0)$ for all i such that $0 \leqq i \leqq s/2$ and RP_i not satisfied. If any accept then declare all such RP_i satisfied, since $x_0 \notin L_1^{A_s}$ and the restraint will be set to insure $x_0 \notin L_1^A$. If all reject then take the least (for definiteness) $w \in D_{x_0}$ not queried in any of the $P_i^{A_s}(x_0)$ computations and set $A_s \leftarrow A_s \cup \{w\}$. This does not satisfy any RP_i but it does help to make L_1^A infinite.

Substage d: (Requirements RNP_i, $1 \leqq i \leqq s/2$). Run $NP_i^{A_s}(0^n)$ for all i such that $1 \leqq i \leqq s/2$, and RNP_i not satisfied. Since all strings of length $\leqq p_s(n)$ are now decided, all the $NP_i^{A_s}(0^n)$ computations are valid. If $NP_j^{A_s}(0^n)$ accepts then declare all such RNP_j satisfied, since $1^{2^n} \notin A_s$, and $r(s)$ will be set to ensure $1^{2^n} \notin A$. If all reject then set $A_s \leftarrow A_s \cup \{1^{2^n}\}$. This does not satisfy any RNP_i but it helps make L_2^A infinite.

Substage e: (Decide strings of length k, $p_s(n) < k \leqq 2^n$). Similar to substage a.

Substage f: Set $A_{s+1} \leftarrow A_s$ as it stands at the end of substage e.

Set $r(s+1) \leftarrow 2^n + 1$.

END OF CONSTRUCTION

$L_1^A(L_2^A)$ is infinite becaus
with indices less than $s/2$;
$|L_1^{A_s}| \geqq s - (s/2) = s/2$ ($|L_2^{A_s}| \geqq$

LEMMA 2. $NP^A \subseteq EXP_1^A$

Proof. We show that for a
accepts in $p_i(|x|)$ steps which
$1x$ as a binary number, $1x$
EXP_1^A machine has time to v

Note that $1x$ was the larg
is not strictly necessary but n
for almost all x, as (finitely oft
It is necessary to be able to c

All elements $x \in L_1^A$ are
stage $s+1$ where $|x| = n$; no e
the x_0 used in substage c of s

LEMMA 3. *Each requirer*

Proof. If a requirement
and will never receive attenti
all $s \geqq s_0$, for all $j < i$, RP_j do

We show that if $P_i^A \cap L_1^A$ i
Let s_2 be the least number g
exist as $P_i^A \cap L_1^A$ is infinite a
not been satisfied then at sta
thus satisfied. □

LEMMA 4. *Each requiren*

Proof. Similar to Lemma

2.2. $EXP_k^B \subsetneq NP^B$. In th
previous section is possible as
1 in that we exhibit an oracle
no infinite subset of L^B or $\overline{L}$
Meyer, who suggested the p
which is proved rigorously in

PROPOSITION. *Let B be*
EXP_k^B*-bi-immune. Then any*
than $2^{O(n^k)}$ *on all but a finite*

Lynch [25] proved that i
that every infinite subset of
way: any deterministic algori
all but a finite number of val
reason that L is hard ("hard"
the set L is its own hard core
for a proper subset of L that n
on hard cores see [11], and fo

THEOREM 5. *There exists*
inequality is witnessed by a la

Proof. For clarity we pre
k is similar.

Set $r(s+1) \leftarrow 2^n + 1$.
END OF CONSTRUCTION

$L_1^A(L_2^A)$ is infinite because during stages $1, 2, \cdots, s+1$ we consider only machines with indices less than $s/2$; hence in at most $s/2$ stages nothing enters $L_1^A(L_2^A)$ so $|L_1^{A_s}| \geqq s-(s/2) = s/2$ ($|L_2^{A_s}| \geqq s - s/2 = s/2$).

LEMMA 2. $NP^A \subseteq EXP_1^A$.

Proof. We show that for any i, $L(NP_i^A) \in EXP_1^A$. The string $x \in L(NP_i^A)$ iff $NP_i^A(x)$ accepts in $p_i(|x|)$ steps which happens exactly when $0^{c(i,1x,p_i(x))} \in A$. Since, looking at $1x$ as a binary number, $1x < 2^{|1x|}$, $c(i, 1x, p_i(x)) < c(i, 2^{|1x|}, p_i(x)) = 2^{O(|x|)}$; hence an EXP_1^A machine has time to write down the query. □

Note that $1x$ was the largest argument to the function $c(\cdots)$. The "t" in $c(i, 1x, t)$ is not strictly necessary but makes the proofs easier. Without it the code would work for almost all x, as (finitely often) the code strings would interfere with the computation. It is necessary to be able to code only one string per length for the sake of substage c.

All elements $x \in L_1^A$ are there because we put them there during substage c of stage $s+1$ where $|x| = n$; no element enters L_1^A accidentally. Formally, let $x_0(s)$ denote the x_0 used in substage c of stage s. Note that $L_1^A \subseteq \{x_0(t) \mid t \in N\}$.

LEMMA 3. *Each requirement RP_i is satisfied.*

Proof. If a requirement ever receives attention, then it is permanently satisfied and will never receive attention again; hence, there exists a least stage s_0 such that for all $s \geqq s_0$, for all $j < i$, RP_j does not receive attention at stage s.

We show that if $P_i^A \cap L_1^A$ is infinite then RP_i receives attention. Let $s_1 = \max(s_0, 2i)$. Let s_2 be the least number greater than s_1 such that $x_0(s_2) \in P_i^A$ (such a string must exist as $P_i^A \cap L_1^A$ is infinite and all elements of L_1^A are $x_0(s)$ for some s). If RP_i has not been satisfied then at stage s_2, during substage c, RP_i receives attention and is thus satisfied. □

LEMMA 4. *Each requirement RNP_i is satisfied.*

Proof. Similar to Lemma 3. □

2.2. $EXP_k^B \subsetneqq NP^B$. In this section we show that the opposite inclusion of the previous section is possible as well. We obtain a stronger type of result than Theorem 1 in that we exhibit an oracle B such that there is an infinite language $L^B \in NP^B$ with no infinite subset of L^B or $\overline{L^B}$ in EXP_k^B. We say that L^B is EXP_k^B-bi-immune. Albert Meyer, who suggested the problem to us, pointed out the following consequence, which is proved rigorously in [14].

PROPOSITION. *Let B be an oracle such that there is an infinite $L \in NP^B$ which is EXP_k^B-bi-immune. Then any deterministic algorithm for L must operate in time greater than $2^{O(n^k)}$ on all but a finite set of points.*

Lynch [25] proved that if $L \notin DTIME(f(n))$ then there exists a set $L' \subseteq L$ such that every infinite subset of L' is not in $DTIME(f(n))$ in the following very strong way: any deterministic algorithm that decides L' must take more than time $f(n)$ on all but a finite number of values. L' is called the "hard core" of L, because it is the reason that L is hard ("hard" meaning $L \notin DTIME(f(n))$). In the above proposition, the set L is its own hard core with respect to EXP_k^B, that is, there is no need to look for a proper subset of L that makes L hard; the set L is hard in and of itself. For more on hard cores see [11], and for more on bi-immunity see [4].

THEOREM 5. *There exists a recursive oracle B such that $EXP_k^B \subsetneqq NP^B$, and this inequality is witnessed by a language $L^B \in NP^B$ such that L^B is EXP_k^B-bi-immune.*

Proof. For clarity we present the proof for $k = 1$. The proof for larger values of k is similar.

We construct the set B in stages. During the construction we code EXP_1^B into NP^B by the following method:

For all i and x, E_i^B accepts x iff there exists w such that $\langle i, x\rangle w \in B$ and $|w| = |\langle i, x\rangle|^8 + 1 - |\langle i, x\rangle|$. Clearly this implies $EXP_1^B \subseteq NP^B$. Note that $|\langle i, x\rangle w| = |\langle i, x\rangle|^8 + 1$ is not a fourth power. This will ensure that no code string is ever placed into B by accident.

We let

$$L^B = \{x \mid \exists_y y \in B, |w| = |x|^4\}.$$

Note that $L^B \in NP^B$ for all B.

To take care of infinite subsets of L^B and $\overline{L^B}$ we have the following requirements:

$$R_{\langle 1,i\rangle}: L(E_i^B) \text{ infinite} \rightarrow L(E_i^B) \cap \overline{L^B} \neq \emptyset.$$

$$R_{\langle 2,i\rangle}: L(E_i^B) \text{ infinite} \rightarrow L(E_i^B) \cap L^B \neq \emptyset.$$

The requirements inherit a priority ordering from the ordering given by the pairing function $\langle \text{-}, \text{-}\rangle$.

Recall that h_i bounds the running time of machine E_i. We let B_s denote the strings put into B through the first s stages.

CONSTRUCTION

Stage 0: $B_0 \leftarrow \emptyset$.

Stage $s+1$:

(1) For each $i \leqq s$, if $h_i(s) < 2^{s^2}$, the run $E_i^{B_s}$ on all strings of length s and preserve each of these computations by restraining from B all the strings queried in the computation which were not in B_s. Note that the total number of strings restrained at this stage is at most $s \times 2^s \times 2^{s^2} < 2^{s^3}$.

(2) Find the least $e = \langle j, i\rangle < s$ such that

(a) R_e is not satisfied.

(b) There is an $x \in \Sigma^*$ such that $|x| = s$ and $E_i^{B_s}$ accepts x.

(c) $h_i(s) < 2^{s^2}$.

If $j = 1$, then to ensure $E_i^B \cap \overline{L^B} \neq \emptyset$ we restrain all strings of length s^4 from B.

If $j = 2$, then to ensure $E_i^B \cap L^B \neq \emptyset$ we place into B the least string of length s^4 that is not restrained from B. (Note that such a string will not be a code string.) There must be such a string since the total number of strings restrained from B up to this point in the construction is less than $\sum_{i=1}^{s} 2^{i^3} < 2^{s^4}$. (Note that the $j = 1$ case of the previous stages restrains only strings of length less than s^4.)

R_e is said to have acted and is now declared satisfied.

(3) For each $E_i^{B_s}(x)$ which has just been run and which accepted x, find some w such that $|w| = |\langle i, x\rangle|^8 - |\langle i, x\rangle| + 1$, and $\langle i, x\rangle w$ is not restrained from B. Put $\langle i, x\rangle w$ into B. Such a w must exist since the number of possible w is $2^{|\langle i,x\rangle|^8 - |\langle i,x\rangle| + 1}$. Note that $|\langle i, x\rangle| \geqq |x| = s$. The function $f(n) = n^8 - n + 1$ in domain $n \geqq s$ has its only minimum at $n = s$. Hence $2^{|\langle i,x\rangle|^8 - |\langle i,x\rangle| + 1} \geqq 2^{s^8 - s + 1}$, which exceeds 2^{s^4}, the number of strings restrained. Hence such a w must exist.

END OF CONSTRUCTION

We prove that each R_e is eventually satisfied. Note that each R_e is acted upon at most once. Assume that $e = \langle j, i\rangle$ and E_i^B accepts an infinite set. Let s_0 be a stage beyond which no R_k, $k < e$, will act, and such that for all $s > s_0$, we have $h_i(s) < 2^{s^2}$. Past s_0, all computations of E_i^B are preserved. Since E_i^B is infinite, there is a stage $s_1 \geqq s_0$ where $E_i^{B_{s_1}}$ accepts some string of length s_1. At this stage R_e acts and becomes satisfied.

2.3. NP^C incomparable

in the other. We show there

THEOREM 6. *There is a*

with immunity.

Proof. Let

$$L_1^C = \{$$

$$L_2^C = \{$$

(The condition that n be odd

of each other.)

Clearly $L_1^C \in NP^C$ and

and contains no infinite subs

$$RE_i: L($$

$$RNP_i:$$

We construct C in stag

the set of strings placed into

at no later stage may an elem

guarantees that a requiremen

recursive.

CONSTRUCTION

Stage 0: $C_0 \leftarrow \emptyset$, $r(0) \leftarrow$ (

Stage $s+1$: There are tw

Case 1: s is even (look a

such that $n \geqq r(s)$. Run $E_i^{C_s}$(

RE_i not satisfied. If any acce

since $0^n \notin L_1^{C_s}$ and the restrain

take the least x (for definiten

of the above computations, a

at most $(s/4) \times (1/s) \times 2^{n^{k+1}} <$

increases the cardinality of L

Set $r(s+1) \leftarrow 2^{n^{k+1}} + 1$. T

in the above computations so

have $0^n \notin L_1^{C_{s+1}}$, and $0^n \notin L_1^C$.

then it remains satisfied at al

Case 2: s is odd (look a

all i such that $0 \leqq i \leqq s/4$, $p_i($

then let $C_{s+1} \leftarrow C_s$ and declar

will be set to insure $0^n \notin L_2^C$.

action puts 0^n into L_2^C, whic

infinite.

Set $r(s+1) \leftarrow 2^{n^k} + 1$. If

remains satisfied at all later st

END OF CONSTRUCTION

$L_1^C (L_2^C)$ are infinite by th

the requirements are satisfied

in L_1^C and L_2^C are there becaus

LEMMA 7. *Each requiren*

2.3. NP^C incomparable to EXP_k^C. It may be that neither NP nor EXP is contained in the other. We show there is an oracle for which this is the case.

THEOREM 6. *There is a recursive oracle C such that NP^C is incomparable to EXP_k^C with immunity.*

Proof. Let

$$L_1^C = \{0^n \mid \exists_x x \in C |x| = n^{k+1} \text{ and } n \text{ is odd}\},$$
$$L_2^C = \{0^n \mid 1^{2^n} \in C\}.$$

(The condition that n be odd in L_1^C will make actions taken for L_1^C and L_2^C independent of each other.)

Clearly $L_1^C \in NP^C$ and $L_2^C \in EXP_k^C$. We need to ensure that $L_1^C(L_2^C)$ is infinite and contains no infinite subset in $EXP_k^C(NP^C)$. We state this in terms of requirements:

$$RE_i\colon L(E_i^C) \text{ infinite} \to L(E_i^C) \cap \overline{L_1^C} \neq \varnothing,$$
$$RNP_i\colon L(NP_i^C) \text{ infinite} \to L(NP_i^C) \cap \overline{L_2^C} \neq \varnothing.$$

We construct C in stages. At the end of each stage we define C_s and $r(s)$: C_s is the set of strings placed into C through stage s, $r(s)$ is a "restraint function" in that at no later stage may an element w, $|w| < r(s)$, be placed into C. The restraint function guarantees that a requirement satisfied at stage s remains so, and that $C = \bigcup_{s=0}^{\infty} C_s$ is recursive.

CONSTRUCTION

Stage 0: $C_0 \leftarrow \varnothing$, $r(0) \leftarrow 0$.

Stage $s+1$: There are two cases:

Case 1: s is even (look at RE_i requirements). Let $n_s = n$ be the least odd number such that $n \geqq r(s)$. Run $E_i^{C_s}(0^n)$ for all i such that $0 \leqq i \leqq s/4$, $h_i(n) < (1/s) \times 2^{n^{k+1}}$, and RE_i not satisfied. If any accepts then set $C_{s+1} \leftarrow C_s$ and declare all such RE_i satisfied, since $0^n \notin L_1^{C_s}$ and the restraint will be set to insure $0^n \notin L_1^C$. If no $E_i^{C_s}(0^n)$ accepts then take the least x (for definiteness), such that $|x| = n^{k+1}$, and x was not queried in any of the above computations, and set $C_{s+1} \leftarrow C_s \cup \{x\}$. Such an x exists because there are at most $(s/4) \times (1/s) \times 2^{n^{k+1}} < 2^{n^{k+1}}$ strings queried. This action puts 0^n into L_1^C, which increases the cardinality of L_1^C, and thus helps it to be infinite.

Set $r(s+1) \leftarrow 2^{n^{k+1}} + 1$. The value $r(s+1)$ exceeds the length of any string queried in the above computations so $E_i^{C_s}(0^n) = E_i^C(0^n)$. Since $r(s+1) > \text{Max}\,(n^{k+1}, r(s))$, we have $0^n \notin L_1^{C_{s+1}}$, and $0^n \notin L_1^C$. Therefore if an RE_i requirement is satisfied at stage $s+1$ then it remains satisfied at all later stages.

Case 2: s is odd (look at RNP_i requirements). Let $n = r(s)$. Run $NP_i^{C_s}(0^n)$ for all i such that $0 \leqq i \leqq s/4$, $p_i(n) < 2^{n^k}$, and RNP_i not satisfied. If any $NP_i^{C_s}(0^n)$ accepts then let $C_{s+1} \leftarrow C_s$ and declare all such RNP_i satisfied, since $0^n \notin L_2^{C_s}$ and the restraint will be set to insure $0^n \notin L_2^C$. If no $NP_i^{C_s}(0^n)$ accepts then set $C_{s+1} \leftarrow C_s \cup \{1^{2^{n^k}}\}$. This action puts 0^n into L_2^C, which increases the cardinality of L_2^C, and thus helps it to be infinite.

Set $r(s+1) \leftarrow 2^{n^k} + 1$. If an RNP_i requirement is satisfied at stage $s+1$, then it remains satisfied at all later stages by the same reasoning used for the R_{e_i} requirements.

END OF CONSTRUCTION

$L_1^C(L_2^C)$ are infinite by the same argument used in Theorem 1. It remains to show the requirements are satisfied. While reading the proof bear in mind that the elements in L_1^C and L_2^C are there because we purposely put them in; no element is in by accident.

LEMMA 7. *Each requirement RE_i is satisfied.*

Proof. Assume not and let i_0 be the least i with RE_{i_0} not satisfied. Then $L(E_{i_0}^C)$ is an infinite subset of L_1^C, and i_0 is never declared satisfied. There must exist an s_0 such that

(a) $i_0 \leqq s_0/4$.

(b) For all $n > n_{s_0}$, $h_i(n) < (1/s) \times 2^{n^{k+1}}$.

(c) For all $i < i_0$, if RE_i will be declared satisfied (as opposed to being satisfied by having E_i^C accept a finite set) then it has been declared satisfied at some time earlier than s_0.

For any even $s > s_0$, at stage s $E_{i_0}^{C_s}(0^{n_s})$ will be run. If it ever accepts then RE_{i_0} will be satisfied, contrary to hypothesis. Hence it rejects 0^{n_s} for all $s > s_0$. Since $L(E_{i_0}^C) \subseteq L_1^C \subseteq \{0^{n_s} \mid s = 0, 1, 2, \cdots, s_0\}$, $L(E_{i_0}^C)$ is finite contrary to hypothesis. Thus RE_{i_0} is satisfied. □

LEMMA 8. *Each requirement RNP_i is satisfied.*

Proof. Similar to Lemma 7. □

3. A general theorem. The proofs of Theorems 1, 5, and 6 only use certain relations between polynomial and exponential functions; hence, they are a natural target for generalization. From our general theorem we obtain oracles relating

(a) $DTIME(n^c)$ and $NTIME(n^d)(c > d > 0)$,
(b) $DTIME(c^n)$ and $NTIME(d^n)(c > d > 0)$,
(c) $DTIME(n^{O(\log n)})$ and $NTIME(n^k)$ $(k > 1)$,
(d) P and $NTIME(n^k)$ $(k > 1)$,
(e) $DTIME(n \log n)$ and $NTIME(n)$,
(f) $DTIME(n \log^*(n))$ and $NTIME(n)$,
(g) $D \# P$ and EXP_k.

Items (a) and (b) answer questions raised by Janos Makowsky [26]. These comparisons are interesting because the *DTIME* machines have more time to operate, but the *NTIME* machines are nondeterministic; therefore, there is no obvious relation between the two. $NTIME(f)$ can be replaced by Σ_j-$TIME(f)$ in the list above. $NTIME(f)$ can also be replaced by $ATIME(f)$, which allows an unbounded number of alternations, though any path of the machine must halt in time f.

THEOREM 9. *Let $D_1, D_2, \cdots$ and $N_1, N_2, \cdots$ be sequences of computable functions such that*

(1) *For all n and i, $N_{i+1}(n) \geqq N_i(n) \geqq n$, $D_{i+1}(n) \geqq D_i(n) \geqq n$.*

(2) *For all i*

$$\lim_{n\to\infty} D_i(n) = \infty, \qquad \lim_{n\to\infty} N_i(n) = \infty, \qquad \lim_{n\to\infty} \frac{N_i(n)}{D_1(n)} = 0, \qquad \lim_{n\to\infty} \frac{D_i(n)}{2^{N_1(n)}} = 0.$$

(3) *D_1 and N_1 are time constructible, so one can compute $D_1(n)(N_1(n))$ in $O(D_1(n))(O(N_1(n)))$ steps. By the linear speed-up theorem* [20] *we may assume the order constant is one.*

(4) *For all i the function that maps x to $N_i(|x|)$ is computable in $D_1(|x|)$ steps.*

(5) *$N_i(n)$ is odd infinitely often.*

Let

$$D^{()} = \bigcup_{i=0}^{\infty} DTIME^{()}(D_i(n)), \qquad N^{()} = \bigcup_{i=0}^{\infty} NTIME^{()}(N_i(n)).$$

Then:

(a) *There exists an oracle A such that $N^A \subsetneqq D^A$ with immunity.*

(b) *There exists an oraci*
(c) *There exists an oracl*
Proof. All three proofs
we shall be brief. We use the
bounds; usage will be clear f
(a) $N^A \subsetneqq D^A$ with immu
is constructed such that the f

RN_i: L

We code N^A into D^A vi

$N_i^A(x$

CONSTRUCTION
Stage 0: $A_0 \leftarrow \emptyset$, $r(0) \leftarrow 0$.
Stage $s+1$ (satisfy RN_s)
Let $n_s = n$ be the least n
k, $n \leqq k \leqq D_1(n)$, and for each
any computation path then
placing it into A does not int
Run $N_s^{A_s}(0^n)$; if it reject
will not place any string of le
requirement is satisfied.
Set $r(s+1) \leftarrow D_1(n)$.
END OF CONSTRUCTION

We see that $N^A \subseteq D^A$: F

An easy calculation shows th
Each RN_i is satisfied by
(b) $D^B \subsetneqq N^B$ with bi-im
$L^B \in N^B$ for all B. B is const

$R_{\langle 1,i\rangle}$:

$R_{\langle 2,i\rangle}$:

We code D^B into N^B by

$D_i^B(x)$ accepts iff th

(We modify $\langle , \rangle$ so that it alwa
Note that the coding depe
on strings of length $\equiv 0 \bmod 3$
We construct B and $\bar{B}$ si
"place into $\bar{B}$."

CONSTRUCTION:
Stage 0: $B_0 \leftarrow \emptyset$.
Stage $s+1$:
(1) For each $i \leqq s$, if $D_i($
preserve each of these compu

(b) *There exists an oracle B such that $D^B \subsetneq N^B$ with bi-immunity.*

(c) *There exists an oracle C such that N^C and D^C are incomparable with immunity.*

Proof. All three proofs are similar to those in Theorems 1, 5, and 6, and hence we shall be brief. We use the same symbols N_i and D_i for machines as well as time bounds; usage will be clear from context.

(a) $N^A \subsetneq D^A$ with immunity. Let $L^A = \{0^n \mid 1^{D_1(n)} \in A\}$. Clearly $L^A \in D^A$. Oracle A is constructed such that the following requirements are satisfied:

$$RN_i\colon L(N_i^A) \cap L^A \text{ infinite} \to N_i^A \cap \overline{L^A} \neq \varnothing.$$

We code N^A into D^A via

$$N_i^A(x) \text{ halts in } t \text{ steps iff } \langle i, x, 0^t\rangle \in A.$$

CONSTRUCTION

Stage 0: $A_0 \leftarrow \varnothing$, $r(0) \leftarrow 0$.

Stage $s+1$ (satisfy RN_s):

Let $n_s = n$ be the least number such that $n > r(s)$ and $D_1(n) > N_s(n)$. For each k, $n \leqq k \leqq D_1(n)$, and for each w, $|w| = k$, $w = \langle i, x, 0^t\rangle$, if $N_i^{A_s}(x)$ accepts in t steps along any computation path then place w into A. Note that the string w is so long that placing it into A does not interfere with the computation it codes.

Run $N_s^{A_s}(0^n)$; if it rejects then place $1^{D_1(n)}$ into A. Since $N_s(n) < D_1(n)$, and we will not place any string of length less than $D_1(n)$ into A, $N_s^{A_s}(0^n) = N_s^A(0^n)$ and the requirement is satisfied.

Set $r(s+1) \leftarrow D_1(n)$.

END OF CONSTRUCTION

We see that $N^A \subseteq D^A$: For a fixed i

$$x \in N_i^A \quad \text{iff } \langle i, x, 0^{N_i(|x|)}\rangle \in A.$$

An easy calculation shows the query length is less than $D_1(|x|)$ for $|x|$ large.

Each RN_i is satisfied by the same reasoning used in Lemma 2.

(b) $D^B \subsetneq N^B$ with bi-immunity. Let $L^B = \{x \mid \exists_y y \in B, |y| = 3N_1(|x|)\}$. Note that $L^B \in N^B$ for all B. B is constructed to satisfy the following requirements:

$$R_{\langle 1,i\rangle}\colon L(D_i^B) \text{ infinite} \to L(D_i^B) \cap \overline{L^B} \neq \varnothing,$$

$$R_{\langle 2,i\rangle}\colon L(D_i^B) \text{ infinite} \to L(D_i^B) \cap L^B \neq \varnothing.$$

We code D^B into N^B by:

$$D_i^B(x) \text{ accepts iff there exists } w,\ |w| = 3N_1(|x|)+1 \text{ and } \langle i, x\rangle w \in B.$$

(We modify $\langle , \rangle$ so that it always produces a string of length divisible by three.)

Note that the coding depends on strings of length $\equiv 1 \bmod 3$ in B while L^B depends on strings of length $\equiv 0 \bmod 3$ in B.

We construct B and $\bar{B}$ simultaneously. The expression "restrain from B" means "place into $\bar{B}$."

CONSTRUCTION:

Stage 0: $B_0 \leftarrow \varnothing$.

Stage $s+1$:

(1) For each $i \leqq s$, if $D_i(s) < 2^{N_1(s)}$, then run $D_i^{B_s}$ on all strings of length s and preserve each of these computations by restraining from B all the strings queried in

the computation that were not in B_s. Note that the total number of strings restrained at this stage is at most $s \times 2^s \times 2^{N_1(s)} < 2^{2N_1(s)}$.

(2) Find the least $e = \langle j, i \rangle < s$ such that

(a) R_e is not satisfied.

(b) There is an $x \in \Sigma^*$ such that $|x| = s$ and $D_i^{B_s}$ accepts x.

If $j = 1$, then to ensure $D_i^B \cap \overline{L^B} \neq \emptyset$ we restrain all strings of length $3N_1(|x|)$.

If $j = 2$, then to ensure $D_i^B \cap L^B \neq \emptyset$ we place into B_{s+1} the least string of length $3N_1(|x|)$ that is not restrained from B. There must be such a string since the total number of strings restrained from B up to this point in the construction is less than $\sum_{i=1}^{s} 2^{2N_1(i)} < 2^{3N_1(s)}$.

Note that the $j = 1$ case of the previous stages restrains only strings of length $3N_1(|x|)$. These strings will not be relevant when we try to code D_i^B into NP^B.

R_i is now said to be satisfied.

(3) For each $D_i^{B_s}(x)$ that has just been run and has accepted x, find some w such that $|w| = 3N_1(|x|) + 1$ and $\langle i, x \rangle w$ is not restrained from B. Put $\langle i, x \rangle w$ into B_{s+1}. Such a w will exist, since the number of possible w's is $2^{3N_1(|x|)+1}$ which exceeds $2^{3N_1(s)}$, the number of strings restrained (not including those possibly restrained in the $j = 1$ case above, since they are all of length $3N_1(|x|)$).

END OF CONSTRUCTION

The requirements are satisfied for reasons similar to those stated at the end of Theorem 5.

(c) N^C incomparable to D^C with immunity. Let

$$L_1^C = \{0^n \mid \exists x \in C |x| = 2N_1(n) + 1, N_1(n) \text{ odd}\},$$

$$L_2^C = \{0^n \mid 1^{2D_1(n)} \in C, D_1(n) \text{ even}\}.$$

The parity of n is included so that actions taken to place or restrain elements from $L_1^C(L_2^C)$ do not affect $L_2^C(L_1^C)$. Clearly we have $L_1^C \in N^C$ and $L_2^C \in D^C$. We need to ensure that $L_1^C(L_2^C)$ is infinite and contains no infinite subset in $D^C(N^C)$. We state this in terms of requirements:

$$RD_e\text{: } L(E_e^C) \cap L_1^C \text{ infinite} \rightarrow L(E_e^C) \cap \overline{L_1^C} \neq \emptyset,$$

$$RN_e\text{: } L(N_e^C) \cap L_2^C \text{ infinite} \rightarrow L(N_e^C) \cap \overline{L_2^C} \neq \emptyset.$$

We construct C in stages. The same conventions apply here as in Theorem 6.

CONSTRUCTION

Stage 0: $C_0 \leftarrow \emptyset$, $r(0) \leftarrow 0$.

Stage $s+1$: There are two cases:

Case 1: s is even (look at RD_e requirements). Let $n_s = n$ be the least odd number such that $n \geq r(s)$. Run $D_i^{C_s}(0^n)$ for all i such that $0 \leq i \leq s/4$, $D_i(n) < (1/s) \times 2^{2N_1(n)+1}$, and RE_i not satisfied. We denote n_s by n throughout. If any accepts then set $C_{s+1} \leftarrow C_s$ and declare all such RE_i satisfied, since $0^n \notin L_1^{C_s}$ and the restraint will be set to insure $0^n \notin L_1^C$. If no $D_i^{C_s}(0^n)$ accepts then take the least x (for definiteness), such that $|x| = 2N_1(n) + 1$, x was not queried in any of the above computations, and set $C_{s+1} \leftarrow C_s \cup \{x\}$. Such an x exists because there are at most $(s/4) \times (1/s) \times 2^{2N_1(n)+1} < 2^{2N_1(n)+1}$ strings queried. This action puts 0^n into L_1^C, which helps it to be infinite.

Case 2: s is odd (look at RN_e requirements). Let n be the least number larger than $r(s)$. Run $N_i^{C_s}(0^n)$ for all i such that $0 \leq i \leq s/4$, $N_i(n) < 2D_1(n)$, and RNP_i not satisfied. If any $N_i^{C_s}(0^n)$ accepts then set $C_{s+1} \leftarrow C_s$ and declare all such RN_i satisfied, since $0^n \notin L_2^{C_s}$ and the restraint will be set to insure $0^n \notin L_2^C$. If no $N_i^{C_s}(0^n)$ accepts then set $C_{s+1} \leftarrow C_s \cup \{1^{2D_1(n)}\}$. This action puts 0^n into L_2^C which helps it to be infinite.

In either case set $r(s+$
requirement is satisfied at s
END OF CONSTRUCTIO

The same reasoning us
the requirements satisfied.

In all the constructions i
a language out of a nondeter
has to ask too long. The san
any class where no computa
particular, we can diagonal
definition of alternating mac

4. Corollaries. We list s
of the general theorem each
of oracles to force relations
are not logically independer
cases we can, using padding
cannot be equal. No proofs
in [7] and [8]. By the com
NTIME can be replaced by

Corollaries that mentic
proved using Theorem 9 v
mention $DTIME(O(f))$ an
$if(n)$. Similar remarks apply
In most of these cases, proo

COROLLARY 10. *Given*
B, and C such that

(1) $NTIME^A(n^d) \subsetneq DT$

(2) $DTIME^B(n^c) \subsetneq NT$

(3) $NTIME^C(n^d)$ *incor*

with all the inclusions holding

COROLLARY 11. *Given*
B, and C such that

(1) $NTIME^A(d^n) \subsetneq DT$

(2) $DTIME^B(c^n) \subsetneq NT$

(3) $NTIME^C(d^n)$ *incor*

with all the inclusions holding

COROLLARY 12. *Given*

(1) $NTIME^A(n^k) \subsetneq DT$

(2) $DTIME^B(n^{O(\log n)})$

(3) $NTIME^C(n^k)$ *incor*

(4) *For all oracles X, N*

with all the inclusions holding

COROLLARY 13. *Given*

(1) $NTIME^A(n^k) \subsetneq P^A$,

(2) $P^B \subsetneq NTIME^B(n^k)$,

(3) $NTIME^C(n^k)$ *incor*

(4) *For all oracles X, N*

with all the inclusions holding

COROLLARY 14. *There e*

In either case set $r(s+1) \leftarrow 2^{n^k}+1$. This preserves all computations; hence if any requirement is satisfied at stage s then it is permanently satisfied.

END OF CONSTRUCTION

The same reasoning used in Theorem 6 applies to prove L_1^C, L_2^C infinite and all the requirements satisfied.

In all the constructions in this paper (and in the above proof), where we diagonalize a language out of a nondeterministic class we do so by making the questions a machine has to ask too long. The same technique can be used to diagonalize a language out of any class where no computation path can write a long question on the query tape. In particular, we can diagonalize out of Σ_i^p and other alternating classes (see [19] for definition of alternating machines). We will use this fact in some of the corollaries below.

4. Corollaries. We list some corollaries of the general theorem. Due to the nature of the general theorem each corollary consists of three statements about the existence of oracles to force relationships between complexity classes. Some of the statements are not logically independent, e.g., Corollary 10-(1) implies Corollary 13-(1). In some cases we can, using padding techniques, prove that for all oracles the classes in question cannot be equal. No proofs are given for these, as the reader may find similar proofs in [7] and [8]. By the comment following Theorem 9, in all the corollaries below *NTIME* can be replaced by *ATIME* or Σ_i-*TIME*.

Corollaries that mention $DTIME(f)$ and/or $NTIME(f)$, (Σ_j-$TIME(f)$) are proved using Theorem 9 with $D_i(n)=f(n)$ and/or $N_i(n)=f(n)$. Corollaries that mention $DTIME(O(f))$ and/or $NTIME(O(f))$ use $D_i(n)=if(n)$ and/or $N_i(n)=if(n)$. Similar remarks apply to $DTIME(n^{O(f)})$, $NTIME(n^{O(f)})$, and Σ_n-$TIME(n^{O(f)})$. In most of these cases, proofs are omitted.

COROLLARY 10. *Given c, d real numbers such that $1<d<c$, there exist oracles A, B, and C such that*

(1) $NTIME^A(n^d) \subsetneq DTIME^A(n^c)$,

(2) $DTIME^B(n^c) \subsetneq NTIME^B(n^d)$,

(3) $NTIME^C(n^d)$ *incomparable to* $DTIME^C(n^c)$,

with all the inclusions holding with immunity, and (2) *with bi-immunity.*

COROLLARY 11. *Given c, d real numbers such that $0<d<c$ there exist oracles A, B, and C such that*

(1) $NTIME^A(d^n) \subsetneq DTIME^A(c^n)$,

(2) $DTIME^B(c^n) \subsetneq NTIME^B(d^n)$,

(3) $NTIME^C(d^n)$ *incomparable to* $DTIME^C(c^n)$,

with all the inclusions holding with immunity, and (2) *with bi-immunity.*

COROLLARY 12. *Given a $k \geq 1$, there exist oracles A, B, and C such that*

(1) $NTIME^A(n^k) \subsetneq DTIME^A(n^{O(\log n)})$,

(2) $DTIME^B(n^{O(\log n)}) \subsetneq NTIME^B(n^k)$,

(3) $NTIME^C(n^k)$ *incomparable to* $DTIME^C(n^{O(\log n)})$,

(4) *For all oracles X,* $NTIME^X(n^k) \neq DTIME^X(n^{O(\log n)})$,

with all the inclusions holding with immunity, and (2) *with bi-immunity.*

COROLLARY 13. *Given a $k>1$, there exist oracles A, B, and C such that*

(1) $NTIME^A(n^k) \subsetneq P^A$,

(2) $P^B \subsetneq NTIME^B(n^k)$,

(3) $NTIME^C(n^k)$ *incomparable to* P^C,

(4) *For all oracles X,* $NTIME^X(n^k) \neq P^X$,

with all the inclusions holding with immunity, and (2) *with bi-immunity.*

COROLLARY 14. *There exist oracles A, B, and C such that*

(1) $NTIME^A(n) \subsetneq DTIME^A(n \log n)$,
(2) $DTIME^B(n \log n) \subsetneq NTIME^B(n)$,
(3) $NTIME^C(n)$ *incomparable to* $DTIME^C(n \log n)$,
with all the inclusions holding with immunity, and (2) *with bi-immunity.*

COROLLARY 15. *There exist oracles A, B, and C such that*
(1) $NTIME^A(n) \subsetneq DTIME^A(n \log^*(n))$,
(2) $DTIME^B(n \log^*(n)) \subsetneq NTIME^B(n)$,
(3) $NTIME^C(n)$ *incomparable to* $DTIME^C(n \log^*(n))$,
with all the inclusions holding with immunity, and (2) *with bi-immunity.*

By the note following Theorem 9, we may replace *NTIME* with any alternating class. This yields Corollaries 16, 17, and 18.

COROLLARY 16. *Given a positive real k and an integer j there exist oracles A, B, and C such that*
(1) $P^A \subsetneq \Sigma_j^{p,A} \subsetneq EXP_k^A$,
(2) $EXP_k^B \subsetneq \Sigma_j^{p,B}$,
(3) $\Sigma_j^{p,C}$ *incomparable to* EXP_k^C,
(4) *For all oracles X,* $\Sigma_j^{p,X} \neq EXP_k^X$,
with all the inclusions holding with immunity, and (2) *with bi-immunity.*

Proof. For (1) a simple modification of the construction in Theorem 1 will suffice. For (2) and (3) let $D_i(n) = 2^{in^k}$ and $N_i(n) = n^i$ and use the note following Theorem 9. □

It is known that $DTIME(n \log^*(n)) \subseteq \Sigma_2\text{-}TIME(n)$ [28], (which [28] showed implies $DTIME(n) \neq NTIME(n)$). This was proven by techniques that do not appear to relativize. Corollary 17 verifies that, indeed, the proof does not relativize, nor can it be made to relativize with modification. The containment is known to be false for almost all oracles [12].

COROLLARY 17. *Given* $k \geq 1$ *there exist oracles A, B, and C such that*
(1) $\Sigma_2\text{-}TIME^A(n) \subsetneq DTIME^A(n \log^*(n))$,
(2) $DTIME^B(n \log^*(n)) \subsetneq \Sigma_2\text{-}TIME^B(n)$,
(3) $\Sigma_2\text{-}TIME^C(n)$ *incomparable to* $DTIME^C(n \log^*(n))$,
with all the inclusions holding with immunity, and (2) *with bi-immunity.*

COROLLARY 18. *Given a* $k \geq 1$ *there exist oracles A, B, and C such that*
(1) $ATIME^A(n^{O(1)}) \subsetneq EXP_k^A$,
(2) $EXP_k^B \subsetneq ATIME^B(n^{O(1)})$,
(3) $ATIME^C(n^{O(1)})$ *incomparable to* EXP_k^C,
(4) *For all oracles X,* $ATIME^X(n^{O(1)}) \neq EXP_k^X$,
with all the inclusions holding with immunity, and (2) *with bi-immunity.*

Since $ATIME(n^{O(1)}) = PSPACE$ [19] one may think that we can replace $ATIME^A(n^{O(1)})$ with $PSPACE^A$ in the above corollary; however, this depends on the definition of relativized space bounded machines. If the space bound applies to the oracle tape then $ATIME(n^{O(1)}) = PSPACE$ relativizes, but there exists A such that $A \notin DSPACE^A(\log n)$ (any nonrecursive A will suffice). There are other definitions of relativized space [9], [30] that allow the machine to ask long questions. Under these definitions, $PSPACE^A$ is very powerful and the proof that $ATIME(n^{O(1)}) = PSPACE$ does not relativize [27] (for the definition in [25] Savitch's Theorem and the simulation of space by time also do not relativize, though in the definition in [30] they do). Oracle constructions using the [25] definition of relativized space appear (not surprisingly) in [25]. Oracle constructions using the [30] definition of relativized space appear in [13]. Recently J. Buss [9] proposed a definition of relativized space such that A is in $DSPACE^A(\log n)$, Savitch's Theorem relativizes, the simulation of space by time relativizes, and $ATIME(n^{O(1)}) = PSPACE$ relativizes.

COROLLARY 19. (*Using*
PSPACE relativizes.) *Given*
(1) $PSPACE^A \subsetneq EXP_k^A$,
(2) $EXP_k^B \subsetneq PSPACE^B$,
(3) $PSPACE^C$ *incompar*
(4) *For all oracles X PS*
with all the inclusions holding

Proof. Since *PSPACE* =
$X\ PSPACE^X = ATIME^X(n^C$

Valiant [34] defined the
that there exists a polynomia

$$f(x) = \text{the}$$

He proves that computing th
complete [33], [34]. $D \# P$
bounded Turing Machines w
from $P^{SAT} \subseteq D \# P$ little els
complexity class. $D^A \# P$ den
Turing Machines that have t
$\# P$-function calls. $D \# P^A$
oracle Turing Machines that
an oracle A such that $D \# P$

Some languages in $D \# P$
however, such a language ca
used to recognize it is poly
powerfulness of the question
time it has to work with.

COROLLARY 20. *There e*
(1) $D^A \# P \subsetneq EXP_k^A$,
(2) $EXP_k^B \subsetneq D^B \# P$,
(3) $D^C \# P$ *incomparable*
(4) *For all oracles X,* D^X
with all the inclusions holding

Proof. Since $NP^X \subseteq D^X$
exponentially long strings, c
construct A, B and C. □

5. Conclusions. Nondete
in one way; deterministic mac
The oracles constructed in thi
kinds of power relate to each o

The languages L_1^C and L_2^C
that may separate the two kin
for a D^C machine to recogniz
of what C is. This intuition is
defined in Theorem 9, for alm
also holds for all the pairs of
to believe that these classes ar
$DTIME(n \log^*(n)) \subseteq \Sigma_2\text{-}TIM$
yields another counterexamp
Hypothesis, [22] for the first r

COROLLARY 19. (*Using a definition of relativized space such that* $ATIME(n^{O(1)}) = PSPACE$ *relativizes.*) *Given a* $k \geqq 1$ *there exist oracles A, B, and C such that,*

(1) $PSPACE^A \subsetneqq EXP_k^A$,

(2) $EXP_k^B \subsetneqq PSPACE^B$,

(3) $PSPACE^C$ *incomparable to* EXP_k^C,

(4) *For all oracles* X $PSPACE^X \neq EXP_k^X$,

with all the inclusions holding with immunity, and (2) *with bi-immunity.*

Proof. Since $PSPACE = ATIME(n^{O(1)})$, relativizes we know that for all oracles X $PSPACE^X = ATIME^X(n^{O(1)})$. Using this, Corollary 18 yields the desired result. □

Valiant [34] defined the class of functions $\#P$ to be the set of all functions f such that there exists a polynomial bounded $NDTM$ M such that

$$f(x) = \text{the number of accepting paths of } M \text{ on } x.$$

He proves that computing the permanent of a matrix, and other problems, are $\#P$-complete [33], [34]. $D\#P$ denotes the set of languages computed by polynomial bounded Turing Machines with an oracle to compute some functions in $\#P$. Aside from $P^{SAT} \subseteq D\#P$ little else is known about how $D\#P$ compares with the other complexity class. $D^A\#P$ denotes languages recognized by polynomial bounded oracle Turing Machines that have two oracle tapes, one to query A, and the other to make $\#P$-function calls. $D\#P^A$ denotes languages recognized by polynomial bounded oracle Turing Machines that can query any $\#P^A$ function. Angluin [1] has exhibited an oracle A such that $D\#P^A - (\Sigma_2^{P,A} \cup \Pi_2^{P,A}) \neq \varnothing$.

Some languages in $D\#P$ are hard, as the ability to call a $\#P$ function is powerful; however, such a language can not take too much time to recognize, as the machine used to recognize it is polynomially bounded. Our next corollary shows how the powerfulness of the questions that a $D\#P$ machine can ask contrasts with the limited time it has to work with.

COROLLARY 20. *There exist oracles A, B, and C such that,*

(1) $D^A\#P \subsetneqq EXP_k^A$,

(2) $EXP_k^B \subsetneqq D^B\#P$,

(3) $D^C\#P$ *incomparable to* EXP_k^C,

(4) *For all oracles* X, $D^X\#P \neq EXP_k^X$,

with all the inclusions holding with immunity, and (2) *with bi-immunity.*

Proof. Since $NP^X \subseteq D^X\#P$ for all X, but a $D^X\#P$ machine cannot query exponentially long strings, constructions like the three in Theorem 9 will work to construct A, B and C. □

5. Conclusions. Nondeterministic, alternating, and $D\#P$ machines are powerful in one way; deterministic machines that can use a lot of time are powerful in another. The oracles constructed in this paper indicate that it will be hard to say how the two kinds of power relate to each other, in that recursion theoretic techniques will not suffice.

The languages L_1^C and L_2^C in Theorem 9 provide examples of the kinds of languages that may separate the two kinds of classes. Indeed, it looks like it would be very hard for a D^C machine to recognize L_1^C or for an N^C machine to recognize L_2^C *regardless* of what C is. This intuition is justified by the result that for the classes $D^{()}$ and $N^{()}$ defined in Theorem 9, for almost all oracles X, D^X is incomparable to N^X [12]. This also holds for all the pairs of classes mentioned in the corollaries. This may tempt one to believe that these classes are incomparable in the unrelativized case, but the result $DTIME(n \log^*(n)) \subseteq \Sigma_2\text{-}TIME(n)$ proves this false. This result combined with [12] yields another counterexample to the Random Oracle Hypothesis (see [6] for the Hypothesis, [22] for the first refutation). It may be the case that if the classes involved

are above a certain time limit then the deterministic and nondeterministic classes are incomparable. On the other hand, all *natural NP*-problems are actually in EXP_1, so it may be the case that for low time bounds the deterministic classes are contained in the nondeterministic ones, and for higher ones, the reverse is true. If this is the case, it would be interesting to know where the turning point is, and if for some classes in between we have incomparability.

It is an open question whether our immunity results can be extended to bi-immunity. It seems hard to construct an oracle A such that a language L^A is bi-immune with respect to a nondeterministic time class. It is also open if an oracle can be constructed to force equality in Corollaries 10, 11, 14, and 15.

Acknowledgments. I wish to thank Harry Lewis and Albert Meyer for useful conversations on the topics discussed in this paper; Amihood Amir and Clyde Kruskal for proofreading; and an anonymous referee for pointing out some subtle mistakes in an older version of the proof of Theorem 1.

I would also like to thank both the University of Maryland and Harvard University for providing computer time.

REFERENCES

[1] D. ANGLUIN, *On counting problems and the polynomial-time hierarchy*, Theoret. Comput. Sci., 12 (1980), pp. 161-173.

[2] T. BAKER, J. GILL AND R. SOLOVAY, *Relativizations of the P = ?NP question*, this Journal, 1 (1975), pp. 305-322.

[3] T. BAKER AND A. SELMAN, *A second step towards the polynomial hierarchy*, Theoret. Comput. Sci., 8 (1979), pp. 177-187.

[4] J. L. BALCAZAR, *Simplicity for relativized complexity classes*, manuscript.

[5] J. L. BALCAZAR AND U. SCHONING, *Bi-immune sets for complexity classes*, manuscript.

[6] C. G. BENNET AND J. GILL, *Relative to a random oracle A, $P^A \neq NP^A \neq co\text{-}NP^A$ with probability*, this Journal, 10 (1981), pp. 96-113.

[7] R. BOOK, *Translational lemmas, polynomial time, and* $(\log n)^n$*-space*, Theoret. Comput. Sci., 8 (1976), pp. 177-187.

[8] ———, *On languages accepted in polynomial time*, this Journal, 1 (1972), pp. 281-287.

[9] J. BUSS, *Relativized alternation*, in Lecture Notes in Computer Science 223, Springer, Berlin-New York, 1986, pp. 66-76. (Structure in Complexity Theory—Proceedings of 1st Structure Conference.)

[10] M. DEKHTYAR, *On the relation of deterministic and nondeterministic complexity classes*, in Lecture Notes in Comput. Sci., 45, Springer, Berlin-New York, 1977, pp. 255-259.

[11] D. DU AND R. BOOK, *The existence and density of generalized complexity cores*, J. Assoc. Comput. Mach., to appear.

[12] W. I. GASARCH, *More on the random oracles hypothesis: What's true almost always is not necessarily so*, Tech. Rep. 1596, Univ. of Maryland, Baltimore, MD.

[13] ———, *Relativized space with immunity*, in Math. Systems Theory, Springer, New York-Berlin, to appear.

[14] W. I. GASARCH AND S. HOMER, *Relativizations comparing NP and exponential time*, Inform. and Control, 58 (1983), pp. 88-100.

[15] J. GRESKE AND J. GROLLMAN, *Relativizations of unambiguous and random polynomial time classes*, this Journal, 15 (1986), pp. 511-519.

[16] J. HASTAD, *Almost optimal lower bounds for small depth circuits*, Proc. 18th Annual ACM Symposium on the Theory of Computing, May 1986, pp. 6-20.

[17] H. HELLER, *On relativized exponential and probabilistic complexity classes*, manuscript.

[18] S. HOMER AND W. MAASS, *Oracle dependent properties of the lattice of NP sets*, Theoret. Comput. Sci., 24 (1983), pp. 279-289.

[19] J. HOPCROFT, A. K. CHANDRA, D. C. KOZEN AND L. J. STOCKMEYER, *Alternation*, J. Assoc. Comput. Mach., 28 (1981), pp. 114-133.

[20] J. HOPCROFT AND J. ULLMAN, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.

[21] S. KURTZ, *Another oracle with s*

[22] ———, *On the random oracle h*

[23] ———, *A Relativized Failure o*
Science, Univ. of Chicago, C

[24] ———, *On Sparse sets in NP –*

[25] N. LYNCH, *Log space machines*

[26] J. MAKOWSKY, personal commu

[27] P. ORPONEN, *Complexity classe*
Science 154, Springer, Berlin
Spain, July 1983.

[28] W. PAUL, N. PIPPENGER, E. S.
and related problems, Proceed
1983, pp. 429-437.

[29] C. RACKOFF, *Relativized quest*
(1982), pp. 261-268.

[30] W. RUZZO, J. SIMON AND M.
Comput. System Sci., 28 (19

[31] U. SCHONING AND R. BOOK,
pp. 329-337.

[32] M. SIPSER, *On relativizations an*
Springer, Berlin-New York,

[33] L. G. VALIANT, *The complexity of*

[34] ———, *The complexity of enume*

[35] C. WILSON, *Relativized circuit co*

[36] A. C. C. YAO, *Separating the poly*
Foundation of Computer Sci

[21] S. KURTZ, *Another oracle with strange properties*, manuscript, 1983.
[22] ———, *On the random oracle hypothesis*, Inform. and Control, 57 (1983), pp. 40–47.
[23] ———, *A Relativized Failure of the Berman–Hartmanis Conjecture*, Tech. Rep. Dept. of Computer Science, Univ. of Chicago, Chicago, IL.
[24] ———, *On Sparse sets in NP – P: Relativizations*, this Journal, 14 (1985), pp. 113–119.
[25] N. LYNCH, *Log space machines with multiple oracle tapes*, Theoret. Comput. Sci., 6 (1978), pp. 25–39.
[26] J. MAKOWSKY, personal communication.
[27] P. ORPONEN, *Complexity classes of alternating machines with oracles*, in Lecture Notes in Computer Science 154, Springer, Berlin–New York, 1983. Proceedings 10th ICALP Colloquium, Barcelona, Spain, July 1983.
[28] W. PAUL, N. PIPPENGER, E. SZEMEREDI AND W. TROTTER, *On determinism and nondeterminism and related problems*, Proceedings 24th IEEE Symposium of the Foundation of Computer Science, 1983, pp. 429–437.
[29] C. RACKOFF, *Relativized questions involving probabilistic algorithms*, J. Assoc. Comput. Mach., 29 (1982), pp. 261–268.
[30] W. RUZZO, J. SIMON AND M. TOMPA, *Space-bounded hierarchies and probabilistic computations*, J. Comput. System Sci., 28 (1982), pp. 216–230.
[31] U. SCHONING AND R. BOOK, *Immunity, relativization, and nondeterminism*, this Journal, 13 (1984), pp. 329–337.
[32] M. SIPSER, *On relativizations and existence of complete sets*, in Lecture Notes in Computer Sci., 140, Springer, Berlin–New York, 1982. Proc. 9th ICALP Conference, Aarhus, Denmark.
[33] L. G. VALIANT, *The complexity of computing the permanent*, Theoret. Comput. Sci., 8 (1979), pp. 189–201.
[34] ———, *The complexity of enumeration and reliability problems*, this Journal, 3 (1979), pp. 410–421.
[35] C. WILSON, *Relativized circuit complexity*, J. Comput. System Sci., 31 (1985), pp. 169–181.
[36] A. C. C. YAO, *Separating the polynomial time hierarchy by oracles*, Proc. 26th IEEE Symposium on the Foundation of Computer Science, 1985, pp. 1–10.