SIAM J. COMPUT.
Vol. 4, No. 4, December 1975

RELATIVIZATIONS OF THE $\mathscr{P} =? \mathscr{NP}$ QUESTION*

THEODORE BAKER†, JOHN GILL‡ AND ROBERT SOLOVAY¶

Abstract. We investigate relativized versions of the open question of whether every language accepted nondeterministically in polynomial time can be recognized deterministically in polynomial time. For any set X, let $\mathscr{P}^X$ (resp. $\mathscr{NP}^X$) be the class of languages accepted in polynomial time by deterministic (resp. nondeterministic) query machines with oracle X. We construct a recursive set A such that $\mathscr{P}^A = \mathscr{NP}^A$. On the other hand, we construct a recursive set B such that $\mathscr{P}^B \neq \mathscr{NP}^B$. Oracles X are constructed to realize all consistent set inclusion relations between the relativized classes $\mathscr{P}^X$, $\mathscr{NP}^X$, and co $\mathscr{NP}^X$, the family of complements of languages in $\mathscr{NP}^X$. Several related open problems are described.

Key words. computational complexity, nondeterministic computation, query machines, polynomial-bounded computation

1. Introduction. An important problem in the theory of computation is to characterize the power of nondeterministic computation. A fundamental open question is whether $\mathscr{NP}$ properly contains $\mathscr{P}$. Here $\mathscr{P}$ is the class of languages recognized in polynomial time by deterministic Turing machines, and $\mathscr{NP}$ is the class of languages accepted in polynomial time by nondeterministic Turing machines. One reason for the importance of the $\mathscr{P} =? \mathscr{NP}$ question is that $\mathscr{P}$ and $\mathscr{NP}$ are very natural classes of languages, invariant under reasonable changes of machine model. $\mathscr{P}$ or $\mathscr{NP}$ is the same class whether defined by computations by one-tape Turing machines, multitape Turing machines, or random-access machines. The $\mathscr{P} =? \mathscr{NP}$ question thus deals with the basic nature of computation and not merely with minor aspects of our models of computers.

We can formulate a question similar to $\mathscr{P} =? \mathscr{NP}$ for other models of mathematical computers. In particular, we can relativize the $\mathscr{P} =? \mathscr{NP}$ question to the case of machines which compute with the aid of an oracle. When the oracle answers membership questions about sets of binary strings, the resulting machine class is formally quite similar to the class of Turing machines without oracle. The main result of this paper is that the relativized $\mathscr{P} =? \mathscr{NP}$ question has an affirmative answer for some oracles but a negative answer for other oracles.

We feel that this is further evidence of the difficulty of the $\mathscr{P} =? \mathscr{NP}$ question. By slightly altering the machine model, we can obtain differing answers to the relativized question. This suggests that resolving the original question requires careful analysis of the computational power of machines. It seems unlikely that ordinary diagonalization methods are adequate for producing an example of a language in $\mathscr{NP}$ but not in $\mathscr{P}$; such diagonalizations, we would expect, would apply equally well to the relativized classes, implying a negative answer to all relativized $\mathscr{P} =? \mathscr{NP}$ questions, a contradiction. On the other hand, we do not

* Received by the editors July 16, 1973, and in final revised form October 21, 1974. This work was supported by JSEP Contract N-00014-67-A-0112-0044 and NSF Grant GK-43121 at Stanford University, and NSF Grant GP-38753X at the University of California, Berkeley.

† Department of Mathematics, Florida State University, Tallahassee, Florida 32306.

‡ Department of Electrical Engineering, Stanford University, Stanford, California 94305.

¶ Department of Mathematics, University of California, Berkeley, California 94720.

feel that one can give a general method for simulating nondeterministic machines by deterministic machines in polynomial time, since such a method should apply as well to relativized machines and therefore imply affirmative answers to all relativized $\mathscr{P} =? \mathscr{NP}$ questions, also a contradiction. Our results suggest that the study of natural, specific decision problems offers a greater chance of success in showing $\mathscr{P} \neq \mathscr{NP}$ than constructions of a more general nature.

Our model for computation with the aid of an oracle is the query machine, which is an extension of the multitape Turing machine as described in Hopcroft and Ullman [2]. A *multitape Turing machine* consists of a finite-state control unit, a read-only input tape, a finite number of worktapes, and optionally a write-only output tape. A *query machine*, described by Cook [1], is a multitape Turing machine with a distinguished worktape, called the *query tape*, and three distinguished states, called the *query* state, the *yes* state, and the *no* state.

The action of a query machine is similar to that of a Turing machine with the following extension. When a query machine enters its query state, the next operation of the machine is determined by an oracle. An oracle for a set X will place the query machine into its *yes* state if the binary string written on the query tape is an element of X; otherwise the oracle places the machine into the *no* state. Since there is no chance for confusion, we will identify an oracle for a set X with the set X itself.

A query machine is *deterministic* if its finite control specifies at most one possible operation for each configuration of the machine; otherwise the machine is *nondeterministic*. Certain states of a query machine's finite control are designated as *accepting* states. A language is *recognized* by a deterministic query machine with oracle X if the machine halts on all inputs and halts in an accepting state just when the input string belongs to the language. The language *accepted* by a nondeterministic query machine with oracle X is the set of input strings for which some possible computation of the machine halts in an accepting state.

A query machine is *polynomial-bounded* if there is a polynomial $p(n)$ such that every computation of the machine on every input of length n halts within $p(n)$ steps, whatever oracle X is used. For any oracle X, we denote by $\mathscr{P}^X$ the class of languages recognized by polynomial-bounded deterministic query machines with oracle X, and we denote by $\mathscr{NP}^X$ the class of languages accepted by polynomial-bounded nondeterministic query machines with oracle X. The class of languages whose complements are in $\mathscr{NP}^X$ is denoted by co $\mathscr{NP}^X$.

Every query machine can be converted to a polynomial-bounded machine by attaching a clock which terminates every computation of the machine that exceeds some predetermined polynomial time bound. We can thereby produce a list of polynomial-bounded query machines which perform all possible polynomial-bounded computations. We will denote by P_i the ith deterministic polynomial-bounded machine in this list and by NP_i the ith nondeterministic polynomial-bounded query machine. Without loss of generality, we can assume that $p_i(n)$ is a strict upper bound on the length of any computation by P_i or NP_i with oracle X on an input of length n, where $p_i(n) = i + n^i$. We indicate by use of a superscript when an oracle has been specified for a query machine; thus NP_i^X and P_i^X denote query machines using oracle X.

We encode each finite sequence of binary strings $x_1, x_2, \cdots, x_m$ into the binary string $\langle x_1, x_2, \cdots, x_m\rangle$ that is obtained from the string $x_1 * x_2 * \cdots * x_m$ (over the alphabet $\{0, 1, *\}$) by replacing each occurrence of 0, 1, and $*$ by 00, 01, and 11, respectively. Both the encoding and decoding can be performed in time bounded above by a linear function of $|x_1| + |x_2| + \cdots + |x_m|$. Note that $|x_i| \leqq |\langle x_1, x_2, \cdots, x_m\rangle|$ for every $i \leqq m$.

Consider the canonical enumeration of binary strings: Λ, 0, 1, 00, 01, 10, 11, 000, $\cdots$. When convenient, we will identify the natural number i with the ith string in this enumeration. As usual, if x is a binary string, then x^n denotes x concatenated with itself n times.

In § 2 we shall construct an oracle A such that $\mathcal{P}^A = \mathcal{NP}^A$. By contrast, in § 3 we shall construct an oracle B such that $\mathcal{P}^B \neq \mathcal{NP}^B$. We shall also show that many other relations between the relativized $\mathcal{P}$, $\mathcal{NP}$, and co $\mathcal{NP}$ classes can be shown to hold for certain oracles. In § 4, we describe open problems concerned with the relativization of the Meyer–Stockmeyer $\mathcal{P}$-hierarchy [6].

Some of the results of this paper have been discovered independently by M. J. Fischer, R. Ladner, A. R. Meyer, and H. B. Hunt III. We have acknowledged these independent contributions in the body of the paper.

2. $\mathcal{P} = \mathcal{NP}$, relativized.

In this section we construct a recursive oracle A such that $\mathcal{P}^A = \mathcal{NP}^A$. We also prove that if A is any polynomial-space complete language, then $\mathcal{P}^A = \mathcal{NP}^A$. We conclude the section by showing that whenever $\mathcal{P}^A = \mathcal{NP}^A$, there is a deterministic procedure using oracle A which finds an accepting computation of NP_i^A on input x, provided NP_i^A accepts x; the running time of this procedure is bounded above by a polynomial of the maximum computation time of NP_i^A on input x.

Our first observation is that for every oracle X, the class $\mathcal{NP}^X$ contains polynomial-complete sets.

DEFINITION. Let $\mathcal{S}$ be a class of languages. A set K in $\mathcal{S}$ is *Cook-polynomial-complete* (or simply Cook-complete) in $\mathcal{S}$ if every language in $\mathcal{S}$ can be recognized by a polynomial-bounded deterministic query machine using oracle K. A set K in $\mathcal{S}$ is *Karp-polynomial-complete* (or Karp-complete) if for every set S in $\mathcal{S}$ there is a function $f(x)$ computable in polynomial time such that $x \in S \Leftrightarrow f(x) \in K$.

Every Karp-complete set is Cook-complete. Ladner, Lynch, and Selmon [4] compare the polynomial-bounded reducibilities of Cook and Karp. Note that if K is Cook-complete in $\mathcal{S}$, then $\mathcal{S} \subseteq \mathcal{P}^K$.

LEMMA 1. *For any oracle X, define the language $K(X)$ to be $\{\langle i, x, 0^n\rangle$: some computation of NP_i^X accepts x in fewer than n steps$\}$. Then $K(X)$ is Karp-complete in $\mathcal{NP}^X$. In particular, $\mathcal{P}^X = \mathcal{NP}^X$ if and only if $K(X) \in \mathcal{P}^X$.*

Proof. Clearly $K(X) \in \mathcal{NP}^X$. Now suppose $S \in \mathcal{NP}^X$, say S is accepted by NP_i^X. If we let $f(x) = \langle i, x, 0^{p_i(|x|)}\rangle$, then $f(x)$ is computable in polynomial time. Now $x \in S \Leftrightarrow NP_i^X$ accepts $x \Leftrightarrow NP_i^X$ accepts x in $< p_i(|x|)$ steps $\Leftrightarrow \langle i, x, 0^{p_i(|x|)}\rangle \in K(X)$. Since S was arbitrary, we conclude that $K(X)$ is Karp-complete in $\mathcal{NP}^X$.

Clearly, if $\mathcal{NP}^X = \mathcal{P}^X$, then $K(X) \in \mathcal{P}^X$. Conversely, suppose $K(X) \in \mathcal{P}^X$. Then $\mathcal{P}^{K(X)} \subseteq \mathcal{P}^X$, and $\mathcal{NP}^X \subseteq \mathcal{P}^{K(X)}$ because $K(X)$ is Cook-complete in $\mathcal{NP}^X$. Therefore $\mathcal{NP}^X \subseteq \mathcal{P}^X$ Q.E.D.

Remark. If $\tilde{K}(X)$ is defined by $\{\langle i, x\rangle : NP_i^X$ accepts $x\}$, then $\tilde{K}(x)$ does not belong to $\mathcal{NP}^X$, since there is no uniform polynomial bound on running time of machines NP_i. Thus, although every language in $\mathcal{NP}^X$ can be reduced to $\tilde{K}(X)$ in polynomial time, $\tilde{K}(X)$ is not polynomial-complete.

The following result was also discovered, independently, by Albert Meyer with Michael Fischer and by H. B. Hunt III.

THEOREM 1. *There is an oracle A such that $\mathcal{P}^A = \mathcal{NP}^A$.*

Proof. We construct an oracle A such that $A = K(A)$. Let $A = \{\langle i, x, 0^n\rangle : NP_i^A$ accepts x in $< n$ steps$\}$. This is a valid inductive definition of a set. In a computation of length $< n$, no string of length $\geqq n$ can be queried. To simulate NP_i^A on input x for $< n$ steps, we need know only which elements of length $< n \leqq |\langle i, x, 0^n\rangle|$ belong to A. Therefore A is well-defined, and by definition $A = K(A)$. Since $K(A) = A \in \mathcal{P}^A$, we conclude $\mathcal{P}^A = \mathcal{NP}^A$ by Lemma 1. Q.E.D.

Remarks. Kleene's recursion theorem can also be used to produce a recursive oracle A such that $A = K(A)$. The oracle A constructed in Theorem 1 can be recognized deterministically in exponential time.

We shall next show that there are naturally occurring languages A such that $\mathcal{P}^A = \mathcal{NP}^A$.

A language is said to be *recognizable in polynomial space* if there is a polynomial $p(n)$ and a deterministic Turing machine which recognizes the language and uses no more than $p(n)$ worktape squares on any input of length n. We denote by $\mathcal{PS}$ the family of languages recognizable in polynomial space. We could define $\mathcal{NPS}$ to be the family of languages accepted in polynomial space by nondeterministic Turing machines. But this definition is unnecessary because Savitch [9] has shown that $\mathcal{NPS} = \mathcal{PS}$; every language accepted nondeterministically in space $p(n)$ can be recognized deterministically in space $p(n)^2$.

A language S is *log-space reducible* to A if there is a function $f(x)$ computable in space $\log(|x|)$ such that $x \in S \Leftrightarrow f(x) \in A$. The language A is *polynomial-space complete* if every language in $\mathcal{PS}$ is log-space reducible to A. Since every function computable in log space is computable in polynomial time, every polynomial-space complete language is Karp-complete in $\mathcal{PS}$. If A is polynomial-space complete, then $\mathcal{PS} = \mathcal{P}^A$.

One example of a polynomial-space complete language is $1EQ$, the set of valid sentences in the first-order theory of equality. Other natural examples of word problems which represent polynomial-space complete languages are given by Stockmeyer and Meyer [10]. An artificial example of a polynomial-space complete language is $A = \{\langle i, x, 0^n\rangle$: deterministic Turing machine P_i recognizes x in space $< n\}$.

THEOREM 2. *If A is polynomial-space complete, then $\mathcal{P}^A = \mathcal{NP}^A$.*

Proof. Suppose A is polynomial-space complete. Then $A \in \mathcal{PS}$ and $\mathcal{PS} \subseteq \mathcal{P}^A$. Also $\mathcal{NP}^A \subseteq \mathcal{NPS}$, since every query made of the oracle for A can be answered in polynomial space without recourse to the oracle for A. But $\mathcal{NPS} = \mathcal{PS}$. Therefore $\mathcal{NP}^A \subseteq \mathcal{NPS} = \mathcal{PS} \subseteq \mathcal{P}^A$. Q.E.D.

When $\mathcal{P}^A = \mathcal{NP}^A$, then every language S in $\mathcal{NP}^A$ can be recognized in polynomial time by some deterministic query machine with oracle A. In fact, this deterministic machine P_j^A can be constructed so that it "simulates" a nondeterministic machine NP_i^A that accepts S; whenever NP_i^A on input x reaches a nondeterministic branch point in its computation, the simulating machine P_j^A correctly decides which branch to follow.

LEMMA 2. *Suppose A is an oracle such that $\mathscr{P}^A = \mathscr{NP}^A$. Then for each nondeterministic query machine NP_i there is a deterministic query machine P_j such that P_j^A on input x produces as output an accepting computation of NP_i^A on input x, whenever NP_i^A accepts x.*

Proof. A computation of a query machine NP_i^X is a sequence of instantaneous descriptions of the machine. An instantaneous description is an encoding of the total configuration of the machine, including the state of the finite control, contents of the tapes, and locations of tape heads. A sequence $I_0, I_1, \cdots, I_m$ of instantaneous descriptions of NP_i^X represents a computation of NP_i^X if I_k encodes a configuration of NP_i^X which can be reached in a single step from the configuration encoded by I_{k-1}.

We define COMP (X) to be the set of accepting computations of query machines with oracle X. Specifically, $\mathrm{COMP}(X) = \{\langle i, x, 0^{p_i(|x|)}, I_0, I_1, \cdots, I_m\rangle$: $I_0, I_1, \cdots, I_m$ is an accepting computation of NP_i^X on input $x\}$. Note that $\mathrm{COMP}(X) \in \mathscr{P}^X$, and $m < p_i(|x|)$.

Let INIT (X) be the set of partial computations which are the initial parts of accepting computations by machines with oracle X; that is,

$$\mathrm{INIT}(X) = \{\langle i, x, 0^{p_i(|x|)}, I_0, I_1, \cdots, I_k\rangle : \text{there exist } I_{k-1}, \cdots, I_m \text{ such that } m < p_i(|x|) \text{ and } \langle i, x, 0^{p_i(|x|)}, I_0, I_1, \cdots, I_m\rangle \in \mathrm{COMP}(X)\}.$$

Then $\mathrm{INIT}(X) \in \mathscr{NP}^X$.

If $\mathscr{P}^A = \mathscr{NP}^A$, then INIT$(A)$ can be recognized by some deterministic polynomial-bounded query machine with oracle A. Now suppose NP_i^A accepts x. Then $\langle i, x, 0^{p_i(|x|)}, I_0\rangle \in \mathrm{INIT}(A)$, where I_0 is the instantaneous description of the initial configuration of NP_i^A with input x. We can find an accepting computation of NP_i^A on x as follows:

We wish to determine a computation $I_0, I_1, \cdots, I_m$ such that $m < p_i(|x|)$ and $\langle i, x, 0^{p_i(|x|)}, I_0, I_1, \cdots, I_k\rangle \in \mathrm{INIT(A)}$ for each $k \leqq m$. To find I_k, suppose we have already found $I_0, I_1, \cdots, I_{k-1}$ with $\langle i, x, 0^{p_i(|x|)}, I_0, I_1, \cdots, I_{k-1}\rangle \in \mathrm{INIT}(A)$.

Determine in polynomial time an instantaneous description I_k such that $\langle i, x, 0^{p_i(|x|)}, I_0, \cdots, I_{k-1}, I_k\rangle \in \mathrm{INIT}(A)$. (There are only finitely many possibilities for I_k, since I_k must be the instantaneous description of a configuration of NP_i^A reachable in one step from the configuration described by I_{k-1}.) Since NP_i^A accepts x, we have $\langle i, x, 0^{p_i(|x|)}, I_0, I_1, \cdots, I_m\rangle \in \mathrm{COMP}(A)$ for some $m < p_i(|x|)$. Therefore we will find an accepting computation of NP_i^A on input x in a number of steps at most a polynomial of $p_i(|x|)$. Q.E.D.

We shall use the method of the proof of Lemma 2 in the proof of Theorem 6.

3. $\mathscr{P} \neq \mathscr{NP}$, relativized.

In this section we show that there exist recursive oracles X such that $\mathscr{P}^X \subsetneq \mathscr{NP}^X$. ($\subsetneq$ denotes proper containment.) We shall construct recursive sets B, C, D, E, and F such that

(i) $\mathscr{P}^B \neq \mathscr{NP}^B$;

(ii) $\mathscr{NP}^C$ is not closed under complementation;

(iii) $\mathscr{P}^D \neq \mathscr{N}\mathscr{P}^D$ but $\mathscr{N}\mathscr{P}^D$ is closed under complementation;

(iv) $\mathscr{P}^E \neq \mathscr{N}\mathscr{P}^E$ and $\mathscr{P}^E = \mathscr{N}\mathscr{P}^E \cap \text{co}\,\mathscr{N}\mathscr{P}^E$;

(v) $\mathscr{P}^F \subsetneq \mathscr{N}\mathscr{P}^F \cap \text{co}\,\mathscr{N}\mathscr{P}^F$ and $\mathscr{N}\mathscr{P}^F \neq \text{co}\,\mathscr{N}\mathscr{P}^F$.

(Recall that co $\mathscr{N}\mathscr{P}^X$ is the class of languages whose complements belong to $\mathscr{N}\mathscr{P}^X$.) By suitable choices of oracles, all consistent set inclusion relations among the relativized classes $\mathscr{P}$, $\mathscr{N}\mathscr{P}$, and co $\mathscr{N}\mathscr{P}$ can be realized.

For any oracle X, define the language $L(X) = \{x\colon \text{there is } y \in X \text{ such that } |y| = |x|\}$. Clearly $L(X) \in \mathscr{N}\mathscr{P}^X$. In fact, $L(X)$ can be accepted in linear time by a query machine that writes on its query tape a nondeterministically chosen string y of the same length as input x, then accepts x if and only if y belongs to X.

The following result was obtained independently by Richard Ladner. Although the next theorem follows immediately from Theorems 4–7, we include a direct proof in order to illustrate the basic techniques of this section.

THEOREM 3. *There is an oracle B such that $\mathscr{P}^B \neq \mathscr{N}\mathscr{P}^B$.*

Proof. We construct a set B such that $L(B)$ does not belong to $\mathscr{P}^B$. The construction of B proceeds in stages. We denote by $B(i)$ the finite set of strings placed into B prior to stage i. Recall that $p_i(n)$ is an upper bound on the lengths of computations by P_i^X and NP_i^X for all oracles X and all inputs of length n. Let $n_0 = 0$.

Stage i. Choose $n > n_i$ so large that $p_i(n) < 2^n$. Run query machine P_i with oracle $B(i)$ on input $x_i = 0^n$. If $P_i^{B(i)}$ accepts 0^n, then place no strings into B at this stage. Otherwise, if $P_i^{B(i)}$ rejects 0^n, then add to B the least string (that is, the earliest occurring string in the canonical enumeration of binary strings) of length n not queried during the computation of $P_i^{B(i)}$ on input 0^n. (Such a string exists because not every string of length n can be queried by $P_i^{B(i)}$ on 0^n; in $p_i(n)$ steps, $P_i^{B(i)}$ can ask at most $p_i(n)$ questions, and we have chosen n so that $p_i(n) < 2^n =$ the number of strings of length n.) Finally, let $n_{i+1} = 2^n$. (This will ensure that no string of length $\leqq 2^n$ is added to B at a later stage.) Go to the next stage.

The computation of P_i on input x_i is the same whether B or $B(i)$ is used as oracle, because no string queried by $P_i^{B(i)}$ on input x_i is later added to or deleted from B. At stage i, we ensure that P_i^B does not recognize $L(B)$; by construction, $P_i^{B(i)}$ and hence P_i^B rejects x_i iff some string of length $|x_i|$ belongs to B, that is, iff $x_i \in L(B)$. Therefore $L(B)$ does not belong to $\mathscr{P}^B$. Q.E.D.

The set B constructed in the proof of Theorem 3 is sparse; for every n, there is at most one string x in B such that $n \leqq |x| < 2^n$. An obvious modification of the proof of Theorem 3 yields the following result: there are arbitrarily sparse recursive sets such that $\mathscr{P}^B \neq \mathscr{N}\mathscr{P}^B$.

Richard Ladner has shown that there are oracles B recognizable deterministically in exponential time such that $\mathscr{P}^B \neq \mathscr{N}\mathscr{P}^B$.

The proof of Theorem 3 makes use of the fact that we can query an oracle about any number of length n in approximately n steps. We can therefore conclude that $L(B) \in \mathscr{N}\mathscr{P}^B$. Alternate models for query machines involve the notion of *oracle tapes.* In one model, a query machine is supplied with a tape on which are written the strings in the oracle set, separated by asterisks and listed in increasing order. A variation of this model is obtained by supplying the query machine with an oracle tape on which is written the characteristic function of the oracle set. With these models of query machines, our proofs of Theorems 1–3 are no longer valid. Paul Morris [7] has pointed out that with the oracle-tape models for query machines, if $\mathscr{P} = \mathscr{NP}$, then $\mathscr{P}^X = \mathscr{NP}^X$ for every oracle X.

From Theorems 1 and 3 we see that the answer to the question $\mathscr{P}^X =? \mathscr{NP}^X$ depends on the oracle X. Kurt Mehlhorn [5] has shown that the family of oracles X for which $\mathscr{P}^X = \mathscr{NP}^X$ is a "meagre" set in a space of recursive oracles. In this sense, "most" oracles satisfy $\mathscr{P}^X \neq \mathscr{NP}^X$.

The next result follows from Theorem 6, but again the direct proof is much simpler.

THEOREM 4. *There is an oracle C such that $\mathscr{NP}^C$ is not closed under complementation.*

Proof. By a construction very similar to that of Theorem 3, we generate an oracle C such that $\mathscr{NP}^C$ does not contain $\bar{L}(c)$, the complement of $L(C)$. Let $C(i)$ be the set of string placed into C before stage i, and let $n_0 = 0$.

Stage i. Choose $n > n_i$ so that $p_i(n) < 2^n$. (Thus n is greater than the length of every string queried earlier in this construction.) Run nondeterministic query machine NP_i with oracle $C(i)$ on input $x_i = 0^n$. If $NP_i^{C(i)}$ accepts 0^n, then choose any accepting computation and place into C some string of length n not queried during this computation. Otherwise, place no string into C at this stage. Let $n_{i+1} = 2^n$, and go to next stage.

By construction, NP_i^C accepts x_i iff some string of length $|x_i|$ belongs to C, that is, iff $x_i \in L(C)$. Consequently NP_i^C does not accept $\bar{L}(C)$. Therefore $\bar{L}(C)$ does not belong to $\mathscr{NP}^C$. Q.E.D.

The next result is due to Albert Meyer with Michael Fischer and, independently, to Richard Ladner.

THEOREM 5. *There is an oracle D such that $\mathscr{P}^D \neq \mathscr{NP}^D$ but $\mathscr{NP}^D$ is closed under complementation.*

Proof. It is easily seen that $\mathscr{NP}^X$ is closed under complementation if and only if $\bar{K}(X) \in \mathscr{NP}^X$, where $\bar{K}(X)$ is the complement of the Karp-complete language $K(X)$. We shall construct an oracle D such that (i) $L(D) \in \mathscr{NP}^D - \mathscr{P}^D$ and (ii) $u \in \bar{K}^D$ iff u is a prefix of some string v in D such that $|v| = 2|u|$. Then $\mathscr{P}^D \neq \mathscr{NP}^D$ from (i); and $\bar{K}(D) \in \mathscr{NP}^D$ from (ii) and so $\mathscr{NP}^D$ is closed under complementation.

At stage n in the construction, we decide the membership in D of all strings of length n. In the course of the construction, some strings will be *reserved* for $\bar{D}$, that is, designated as nonmembers of D. An index i will be *cancelled* at some stage when we ensure that P_i^D does not recognize $L(D)$. As usual, $D(n)$ denotes those strings placed into D prior to stage n.

Stage $n = 2m$. For every string z of length $n = 2m$ not reserved for $\bar{D}$ at an earlier stage, determine the prefix u of z of length m. If u encodes a triple $\langle i, x, 0^l \rangle$, then place z into D iff $NP_i^{D(n)}$ does not accept x in fewer than l steps.

Stage $n = 2m + 1$. Let i be the least uncancelled index. If any string of length $\geqq n$ has been reserved for $\bar{D}$, or if $p_i(n) \geqq 2^m$, then add no elements to D at this stage. Otherwise, run P_i with oracle $D(n)$ on input 0^n and reserve for $\bar{D}$ all strings of length $\geqq n$ queried during this computation. If $P_i^{D(n)}$ accepts 0^n, then add no elements to D. But if $P_i^{D(n)}$ rejects 0^n, then add to D the least string of length n not queried (and so not reserved for $\bar{D}$). Finally cancel index i.

Every index is eventually cancelled, and when index i is cancelled at some stage, we have guaranteed that P_i^D does not recognize $L(D)$. Therefore $L(D) \in \mathscr{NP}^D - \mathscr{P}^D$.

At any odd stage $2m+1$, at most $p_i(n) < 2^m$ strings are reserved for $\bar{D}$, and so fewer than $2^0 + 2^1 + \cdots + 2^{m-1} < 2^m$ strings of length $2m$ can be reserved for $\bar{D}$ at odd stages before stage $2m$. Therefore every string u of length m is the prefix of at least one string v of length $2m$ which is never reserved for $\bar{D}$. By construction, $u \in \bar{K}(D)$ iff u prefixes a number $v \in D$ of length $2|u|$, and so $\bar{K}(D) \in \mathscr{NP}^D$. Q.E.D.

The next theorem gives an answer to a question posed by Albert Meyer.

THEOREM 6. *There is an oracle E such that* $\mathscr{P}^E \neq \mathscr{NP}^E$ *and* $\mathscr{P}^E = \mathscr{NP}^E \cap \mathrm{co}\,\mathscr{NP}^E$.

Proof. The construction of E is considerably more complicated than that of the preceding oracles. As usual, we guarantee that $L(E)$ is not in $\mathscr{P}^E$ and so $\mathscr{P}^E \neq \mathscr{NP}^E$. But we must also ensure that whenever both S and its complement $\bar{S}$ belong to $\mathscr{NP}^E$, then S in fact is in $\mathscr{P}^E$.

The rough idea of the proof is the following. By adding infinitely many new elements to an oracle A such that $\mathscr{P}^A = \mathscr{NP}^A$, we obtain an oracle E such that $\mathscr{P}^E \neq \mathscr{NP}^E$. Now if we could quickly recognize the set $E - A$, then $\mathscr{P}^E = \mathscr{NP}^E$. Although we cannot quickly recognize the entire set $E - A$, we can arrange the construction so that whenever both S and its complement $\bar{S}$ are accepted in polynomial time by nondeterministic query machines with oracle E, then we can quickly recognize relevant portions of the set $E - A$, so that we can then combine the machines accepting S and $\bar{S}$ into a deterministic machine to recognize S.

Define $e(n)$ inductively by $e(0) = 0$ and $e(n+1) = 2^{2^{e(n)}}$. Choose any oracle A such that $\mathscr{P}^A = \mathscr{NP}^A$. Without any loss of generality, we can assume that A contains no elements of length $e(n)$ for any $n \geqq 0$. The oracle E is obtained by adding to A at stage n at most one string of length $e(n)$. Let $E(0) = A$ and let $E(n)$ denote the set of numbers placed into E before stage n. There are two types of requirements to be satisfied in the construction of E. An unsatisfied requirement $\langle i, i\rangle$ is *vulnerable* at stage n if $p_i(e(n)) < 2^{e(n)}$. An unsatisfied requirement $\langle j, k\rangle$ with $j \neq k$ is *vulnerable* at stage n if there is a string x such that $e(n-1) < \log_2(|x|) \leqq e(n) \leqq \max\{p_j(|x|), p_k(|x|)\} < e(n+1)$ and neither NP_j nor NP_k with oracle $E(n)$ accepts x. We agree that requirement r_1 has higher priority than requirement r_2 just when $r_1 < r_2$.

Stage n. We satisfy the requirement of highest priority that is vulnerable at stage n. To satisfy requirement $\langle j, k\rangle$ with $j \neq k$, we simply add no string to E at this stage. To satisfy requirement $\langle i, i\rangle$, we run P_i with oracle $E(n)$ on input $0^{e(n)}$. If $P_i^{E(n)}$ rejects $0^{e(n)}$, then we add to E the least string of length $e(n)$ not queried by $P_i^{E(n)}$ on input $0^{e(n)}$; otherwise we add no new element to E.

Every requirement $\langle i, i\rangle$ is eventually satisfied, since there are only finitely many requirements of higher priority. Suppose requirement $\langle i, i\rangle$ is satisfied at stage n. Then P_i^E rejects $0^{e(n)}$ iff there is a string in E of length $e(n)$, and so P_i^E does not recognize $L(E)$. Therefore $L(E) \in \mathscr{NP}^E - \mathscr{P}^E$.

Now suppose S and $\bar{S}$ belong to $\mathscr{NP}^E$. We must show that $S \in \mathscr{P}^E$. Assume NP_j^E accepts S and NP_k^E accepts $\bar{S}$. Note that requirement $\langle j, k\rangle$ is never satisfied, else there would be a string x accepted by neither NP_j^E nor NP_k^E. Let m be so large that

(i) for every x such that $|x| \geqq e(m)$ there is at most one n such that $\log_2 |x| \leqq e(n) \leqq \max \{p_j(|x|), p_k(|x|)\}$;

(ii) if a requirement of higher priority than requirement $\langle j, k\rangle$ is ever satisfied, then it is satisfied before stage m.

We can decide S deterministically in polynomial time using oracle E by the following procedure:

With input x, if $|x| < e(m)$, then use a finite table to decide if $x \in S$. Otherwise $|x| \geqq e(m)$. Calculate the least n such that $e(n) \geqq \log_2 |x|$. Determine which elements were added to E before stage n by querying the oracle E about *all* strings of lengths $e(0), e(1), \cdots, e(n-1)$. Only $O(|x|)$ strings need be queried here, since $e(n-1) \leqq \log_2 |x|$.

Now there are two cases. If $e(n) > \max \{p_j(|x|), p_k(|x|)\}$, then no computation of either NP_j^E or NP_k^E can query oracle E about any string of length $\geqq e(n)$. Therefore the computation of NP_j on input x is the same with oracle $E(n)$ as with oracle E. Since we have already calculated $E(n) - A$, we can produce a query machine $NP_{i(n)}^{E(n)}$ which simulates $NP_j^{E(n)}$ but makes no queries about strings of length $e(m)$ for any $m < n$; these queries are answered without recourse to the oracle $E(n)$ by using the finite table of elements of $E(n) - A$. Clearly $NP_{i(n)}$ on input x gives the same result with oracle A as with oracle $E(n)$. Therefore NP_j^E accepts x if and only if $NP_{i(n)}^A$ accepts x, that is, iff $\langle i(n), x, 0^{p_j(|x|)}\rangle \in K(A)$. (We can easily make sure that $NP_{i(n)}$ has the same running time bound as NP_j, and that the length of the index $i(n)$ is at most a polynomial of $|x|$.) Since $K(A)$ belongs to $\mathscr{P}^A$, we can now determine in polynomial time if NP_j^E accepts x.

In the other case, $e(n) \leqq \max \{p_j(|x|), p_k(|x|)\}$. If neither NP_j nor NP_k with oracle $E(n)$ accepted x, then requirement $\langle j, k\rangle$ would be satisfied at stage n. But requirement $\langle j, k\rangle$ can never be satisfied, for then both NP_j^E and NP_k^E would reject x. Therefore at least one of $NP_j^{E(n)}$ and $NP_k^{E(n)}$ accepts x. As in the first case, since we know $E(n) - A$, we can discover in polynomial time which machine accepts x.

Suppose, to be definite, $NP_j^{E(n)}$ accepts x. (In case $NP_k^{E(n)}$ accepts x, the argument is similar.) We must now determine if NP_j^E accepts x, where E might contain a string of length $e(n)$ not in $E(n)$.

Since $\mathscr{P}^A = \mathscr{NP}^A$ and since we have already calculated $E(n) - A$, we can use the method of Lemma 2 to find an accepting computation of $NP_j^{E(n)}$ on input x. Now we examine this computation to see if it represents a valid computation when oracle E is used instead of $E(n)$. Whenever a string y of length $e(n)$ is queried by $NP_j^{E(n)}$ on x, we consult oracle E about y. There are two subcases.

If no such y belongs to E, then the computation of $NP_j^{E(n)}$ on x is the same as the computation of NP_j^E on x. Now $NP_j^{E(n)}$ accepts x, and so NP_j^E also accepts x. We conclude that $x \in S$.

In the other subcase, $NP_j^{E(n)}$ on input x queries some string y of length $e(n)$ which belongs to E. By construction, there is at most one number in E of length $e(n)$. Thus we have correctly calculated $E(n+1) - E(n)$. Earlier we found $E(n) - A$, so we now know $E(n+1) - A$. Using the method of the first case above, we can finally determine which of machines NP_j and NP_k with oracle $E(n+1)$ accepts x. Since $e(n+1) > \max \{p_j(|x|), p_k(|x|)\}$, no number of length $\geqq e(n+1)$ can be queried by NP_j^E or NP_k^E on x, so NP_j^E accepts x iff $NP_j^{E(n+1)}$ accepts x. Q.E.D.

The next result accounts for the only remaining relation between relativized classes $\mathscr{P}$ and $\mathscr{NP}$.

THEOREM 7. *There is an oracle F such that* $\mathscr{P}^F \subsetneqq \mathscr{NP}^F \cap \text{co}\,\mathscr{NP}^F \subsetneqq \mathscr{NP}$.

Proof. We outline the construction. Let $L_1(F) = \{x : |x|$ is even and there is $y \in F$ with $|y| = |x|\}$, and let $L_2(F) = \{x : |x|$ is odd and there is a string $0y \in F$ with $|0y| = |x|\}$. Modify the construction of Theorem 4 so that $L_1(F) \in \mathscr{NP}^F - \text{co}\,\mathscr{NP}^F$ and $L_2(F) \in \mathscr{NP}^F \cap \text{co}\,\mathscr{NP}^F - \mathscr{P}^F$. To force $L_2(F) \in \mathscr{NP}^F \cap \text{co}\,\mathscr{NP}^F$, we require that for every odd n there is a string $0y$ of length n in F iff there is no string $1y$ of length n in F. We omit the details. Q.E.D.

In this section we have constructed oracles X such that $\mathscr{P}^X \neq \mathscr{NP}^X$. The principal method for showing that $\mathscr{NP}^X$ properly contains $\mathscr{P}^X$ is to ensure that $L(X) \in \mathscr{NP}^X - \mathscr{P}^X$. It is easy to modify the proof of Theorem 3 to obtain an oracle X for which $L(X) \in \mathscr{NP}^X$, but every deterministic query machine with oracle X that recognizes $L(X)$ requires exponential time for all but finitely many inputs.

4. Open problems.

We shall describe several open problems suggested by Albert Meyer. First we recall another characterization of $\mathscr{NP}$ [6].

LEMMA 3. *A language L belongs to* $\mathscr{NP}$ *iff there is a polynomial* $p(n)$ *and a predicate* $R(x, y)$ *in* $\mathscr{P}$ *such that* $x \in L \Leftrightarrow (\exists y)[|y| \leqq p(|x|)\ \&\ R(x, y)]$.

Proof. ($\Leftarrow$) Suppose $x \in L \Leftrightarrow (\exists y)[|y| \leqq p(|x|)\ \&\ R(x, y)]$. Then we can accept L in polynomial time by nondeterministically selecting a string y such that $|y| \leqq p(|x|)$ and accepting x if $R(x, y)$ is true.

($\Rightarrow$) If $L \in \mathscr{NP}$, then L is accepted by some query machine NP_i. Define predicate $R(x, y)$ to hold iff y encodes an accepting computation of NP_i on input x. Clearly $R(x, y)$ belongs to $\mathscr{P}$. We then choose a polynomial $p(n)$ large enough that $|y| < p(|x|)$ whenever y encodes a computation of length $\leqq p_i(|x|)$. Q.E.D.

One can draw analogies between the class $\mathscr{P}$ of languages recognizable in polynomial time and the class $\mathscr{R}$ of recursive (decidable) languages. We may consider a language "practically" decidable if it can be decided by some deterministic polynomial-bounded procedure. From Lemma 3, $\mathscr{NP}$ contains exactly those languages definable by polynomial-bounded existential quantification over predicates in $\mathscr{P}$. Similarly, co $\mathscr{NP}$ contains those languages definable by polynomial-bounded universal quantification over predicates in $\mathscr{P}$. Thus $\mathscr{NP}$ corresponds to Σ_1 in Kleene's arithmetic hierarchy [8], while co $\mathscr{NP}$ corresponds to Π_1.

Meyer and Stockmeyer [6] have defined a polynomial-bounded analogue of the arithmetic hierarchy, the $\mathscr{P}$-hierarchy. They define $\Sigma_0^{\mathscr{P}} = \Pi_0^{\mathscr{P}} = \Delta_0^{\mathscr{P}}$ to be the class $\mathscr{P}$. Then $\Sigma_{i+1}^{\mathscr{P}}$ is the class of languages definable by polynomial-bounded existential quantification over predicates in $\Pi_i^{\mathscr{P}}$; that is, $L \in \Sigma_{i+1}^{\mathscr{P}}$ iff there is a polynomial $p(n)$ and a predicate $R(x, y)$ in $\Pi_i^{\mathscr{P}}$ such that $x \in L \Leftrightarrow (\exists y)[|y| \leqq p(|x|)$ $\&\ R(x, y)]$. Similarly, $\Pi_{i+1}^{\mathscr{P}}$ contains exactly those languages definable by polynomial-bounded universal quantification over predicates in $\Sigma_i^{\mathscr{P}}$. (Equivalently, $L \in \Sigma_i^{\mathscr{P}}$ iff there is a polynomial $p(n)$ and a predicate $R(x, y_1, \cdots, y_i)$ in $\mathscr{P}$ such that $x \in L \Leftrightarrow (\exists y_1)(\forall y_2) \cdots (Qy_i)[|y_1|, |y_2|, \cdots, |y_i| \leqq p(|x|)\ \&\ R(x, y_1, \cdots, y_i)]$, where there are i alternations of polynomial-bounded quantifiers.) $\Delta_{i+1}^{\mathscr{P}}$ is defined to be the class of languages recognizable in polynomial time with the aid of an oracle for some language in $\Sigma_i^{\mathscr{P}}$; that is, $L \in \Delta_{i+1}^{\mathscr{P}}$ iff $L \in \mathscr{P}^S$ for some S in $\Sigma_i^{\mathscr{P}}$. The $\mathscr{P}$-

hierarchy is $\{\Sigma_i^{\mathscr{P}}, \Pi_i^{\mathscr{P}}, \Delta_i^{\mathscr{P}} : i \geqq 0\}$.

The $\mathscr{P}$-hierarchy shares several of the properties of the arithmetic hierarchy:

(i) $L \in \Sigma_i^{\mathscr{P}} \Leftrightarrow \bar{L} \in \Pi_i^{\mathscr{P}}$;

(ii) $\Sigma_i^{\mathscr{P}} \cup \Pi_i^{\mathscr{P}} \subseteq \Delta_{i+1}^{\mathscr{P}} \subseteq \Sigma_{i+1}^{\mathscr{P}} \cap \Pi_{i+1}^{\mathscr{P}}$.

However, it is not known if any of the inclusions in (ii) are proper; the $\mathscr{P}$-hierarchy may consist of only a single class, namely, $\Sigma_0^{\mathscr{P}} = \Pi_0^{\mathscr{P}} = \Delta_0^{\mathscr{P}} = \mathscr{P}$. One of the results of [6] implies the following.

LEMMA 4 (Meyer–Stockmeyer). *If* $\Sigma_i^{\mathscr{P}} = \Pi_i^{\mathscr{P}}$ *for any* $i \geqq 1$, *then* $\Sigma_j^{\mathscr{P}} = \Pi_j^{\mathscr{P}} = \Sigma_i^{\mathscr{P}}$ *for every* $j \geqq i$. *In particular, if* $\mathscr{P} = \mathscr{NP}$, *then* $\Sigma_i^{\mathscr{P}} = \Pi_i^{\mathscr{P}} = \Delta_i^{\mathscr{P}} = \mathscr{P}$ *for every* i.

From Lemma 4 we see that proving $\Sigma_i^{\mathscr{P}} \neq \Pi_i^{\mathscr{P}}$ for any $i \geqq 1$ is a difficult problem since $\mathscr{P} \neq \mathscr{NP}$ is an immediate consequence.

It is easy to relativize the $\mathscr{P}$-hierarchy. For any oracle X, let $\Sigma_0^{\mathscr{P},X} = \Pi_0^{\mathscr{P},X} = \Delta_0^{\mathscr{P},X} = \mathscr{P}^X$. Then $\Sigma_{i+1}^{\mathscr{P},X}$ is the class of languages definable by polynomial-bounded existential quantification over predicates in $\Pi_i^{\mathscr{P},X}$; also $\Pi_{i+1}^{\mathscr{P},X}$ is the class of languages definable by polynomial-bounded universal quantification over predicates in $\Sigma_i^{\mathscr{P},X}$; and $\Delta_{i+1}^{\mathscr{P},X}$ contains languages L such that $L \in \mathscr{P}^S$ for some $S \in \Sigma_i^{\mathscr{P},X}$. The $\mathscr{P}$, X-hierarchy is $\{\Sigma_i^{\mathscr{P},X}, \Pi_i^{\mathscr{P},X}, \Delta_i^{\mathscr{P},X} : i \geqq 0\}$.

Properties (i) and (ii) hold for relativized $\mathscr{P}$-hierarchies, as does Lemma 4. For the oracle A of Theorem 1, $\mathscr{NP}^A = \mathscr{P}^A$, hence the $\mathscr{P}$, A-hierarchy collapses entirely. If D is the oracle of Theorem 5, then $\Sigma_0^{\mathscr{P},D} \subsetneq \Sigma_1^{\mathscr{P},D} = \Pi_1^{\mathscr{P},D}$; the $\mathscr{P}$, D-hierarchy consists of only two levels. If E is the oracle of Theorem 6, then $\Sigma_0^{\mathscr{P},E} \subsetneq \Sigma_1^{\mathscr{P},E} \subsetneq \Sigma_2^{\mathscr{P},E}$ (since $\Pi_1^{\mathscr{P},E} \neq \Sigma_1^{\mathscr{P},E}$) and $\Sigma_0^{\mathscr{P},E} = \Sigma_1^{\mathscr{P},E} \cap \Pi_1^{\mathscr{P},E}$.

Several questions can be asked about relativized $\mathscr{P}$-hierarchies.

(i) Does there exist an oracle X such that $\Sigma_i^{\mathscr{P},X} \subsetneq \Sigma_{i+1}^{\mathscr{P},X}$ for all i, that is, such that the $\mathscr{P}$, X-hierarchy contains infinitely many distinct classes?

(ii) Does there exist an oracle X such that $\Sigma_{i-1}^{\mathscr{P},X} \subsetneq \Sigma_i^{\mathscr{P},X} = \Sigma_{i+1}^{\mathscr{P},X}$; that is, the $\mathscr{P}$, X-hierarchy extends exactly i levels?

(iii) Does there exist an oracle X such that the $\mathscr{P}$, X-hierarchy is not trivial but $\Sigma_i^{\mathscr{P},X} \subsetneq \Delta_{i+1}^{\mathscr{P},X} = \Sigma_{i+1}^{\mathscr{P},X} \cap \Pi_{i+1}^{\mathscr{P},X}$ for all i?

(Only question (i) has an affirmative answer for the arithmetic hierarchy.) An interesting open question, less general than (i)–(iii) is

(iv) Does there exist an oracle X such that $\Sigma_2^{\mathscr{P},X} \neq \Pi_2^{\mathscr{P},X}$?

We were unable to settle (iv) by the methods of this paper.

Acknowledgments. We wish to thank Richard Ladner, Albert Meyer, and Michael Fischer for allowing us to report their results in this paper. We are especially grateful to Albert Meyer for suggesting Theorem 6 and the open problems of § 4. We thank the referees for their corrections and suggestions.

REFERENCES

[1] S. A. COOK, *The complexity of theorem-proving procedures*, Proc. Third Annual ACM Symposium on the Theory of Computing, Shaker Heights, Ohio, 1971, pp. 151–158.

[2] J. HOPCROFT AND J. ULLMAN, *Formal Languages and their Relation to Automata*, Addison-Wesley, Reading, Mass., 1969.

[3] R. M. KARP, *Reducibility among combinatorial problems*, Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–104.

[4] R. LADNER, N. LYNCH AND A. SELMAN, *Comparison of polynomial-time reducibilities*, Proc. Sixth Annual ACM Symposium on the Theory of Computing, Seattle, Wash., 1974, pp. 110–121.

[5] K. Mehlhorn, *On the size of computable functions*, Proc. 14th IEEE Symposium on Switching and Automata Theory, Iowa City, Iowa, 1973, pp. 190–196.

[6] A. R. Meyer and L. J. Stockmeyer, *The equivalence problem for regular expressions with squaring requires exponential space*, Proc. 13th IEEE Symposium on Switching and Automata Theory, 1972, pp. 125–129.

[7] P. H. Morris, personal communication.

[8] H. Rogers, Jr., *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, New York, 1967.

[9] W. J. Savitch, *Relationships between nondeterministic and deterministic tape complexities*, J. Comput. System Sci., 4 (1970), pp. 177–192.

[10] L. J. Stockmeyer and A. R. Meyer, *Word problems requiring exponential time: preliminary report*, Proc. Fifth Annual ACM Symposium on the Theory of Computing, Austin, Texas, 1973, pp. 1–9.