JOURNAL OF COMPUTER AND SYSTEM SCIENCES 30, 395–413 (1985)

Qualitative Relativizations of Complexity Classes*

RONALD V. BOOK

Department of Mathematics,
University of California at Santa Barbara,
Santa Barbara, California 93106

TIMOTHY J. LONG

Department of Computer Science, Ohio State University,
Columbus, Ohio 43210

AND

ALAN L. SELMAN

Department of Computer Science, Iowa State University,
Ames, Iowa 50011

Received September 13, 1984; revised January 9, 1985

The principal result of this paper is a "positive relativization" of the open question "$P = ?$ $NP \cap co\text{-}NP$." That is, the nondeterministic polynomial time-bounded oracle Turing machine endowed with designated accepting states and with designated rejecting states is considered, and suitable restrictions R of this device are developed such that $P = NP \cap co\text{-}NP$ if and only if for every oracle D, $P(D) = NP_R(D)$, where $NP_R(D)$ is the class of languages $L \in NP(D)$ that are accepted by oracle machines operating with restrictions R. Positive relativizations are obtained for the $P = ?$ $\mathcal{U} \cap co - \mathcal{U}$ and $\mathcal{U} = ?$ NP questions also, where $\mathcal{U}$ is the class of languages L in NP accepted by nondeterministic machines that operate in polynomial time and that have for each input at most one accepting computation. The restrictions developed here are "qualitative" in the sense that they restrict the form and pattern of access to the oracle. In contrast, a previous paper [3] developed quantitative relativizations—the number of distinct queries to the oracle is restricted—but no quantitative positive relativization of $P = ? NP \cap co\text{-}NP$ is known.

1. INTRODUCTION

This paper is a continuation of research efforts reported in [3] and in earlier papers [2, 4, 11]. The goal of this project is to develop restrictions R of the standard deterministic and nondeterministic oracle Turing machine such that

* This research was supported in part by the National Science Foundation Grants MCS80–11979, MCS81–20263, DCR83–12472, and DCR84–02033.

0022-0000/85 $3.00

relativizations of various complexity classes by use of these new machine models preserve inclusion relationships among the complexity classes.

The principal result obtained here is a "positive relativization" of the $P = ? NP \cap co\text{-}NP$ question. That is, we describe restrictions R on the behavior of nondeterministic oracle Turing machines such that $P = NP \cap co\text{-}NP$ if and only if for every oracle D, $P(D) = NP_R(D)$, where $NP_R(D)$ is the class of languages $L \in NP(D)$ that are accepted by oracle machines operating with restriction R. Positive relativizations are obtained for the $P = ? \mathcal{U} \cap co\text{-}\mathcal{U}$ and $\mathcal{U} = ? NP$ questions also, where $\mathcal{U}$ is the class of languages L in NP accepted by nondeterministic machines that operate in polynomial time and that for each input have at most one accepting computation.

The type of restrictions developed here are "qualitative" in the sense that they impose constraints on the shape of a machine's computation trees or limit the form and pattern of access to the oracle. The following is an example of a typical qualitative restriction. Let M be a nondeterministic oracle Turing machine that is endowed with designated accepting states and with designated rejecting states. We say that M is *strong* if, relative to each oracle set, for each input string x, either there is at least one accepting computation of M on x or there is at least one rejecting computation of M on x, but not both an accepting computation and a rejecting computation of M on x. This is just one restriction required for our main result. It is an obvious restriction to impose, because strong oracle machines that run in polynomial time accept only those languages that belong to $NP(A) \cap co\text{-}NP(A)$. The other restrictions we need are highly technical, so their definitions will be given in text.

Quantitative positive relativizations of complexity classes, notably of the $P = ? NP$ and $NP = ? co\text{-}NP$ questions, are given in [3]. In [3] it is the number of distinct queries to the oracle that is limited. We suspect that no positive quantitative relativization of the $P = ? NP \cap co\text{-}NP$ question exists, and Section 5 contains a result that supports this contention.

Proof techniques to be used here are a blend of techniques used in [3 and 11]. Therefore, in order to keep this paper self-contained, a number of facts and constructs from [3; 11] will be stated here. They will be found in Sections 2 and 4, and occur as they are needed. Our main relativization result is found in Section 3, and this section especially may be read without prior study of either [3 or 11]. Section 4 compares the relativizations obtained by the various restrictions and includes a study of relationships between the classes defined here and those defined in [3]. The contents of Section 5 are indicated in the previous paragraphs. Briefly, we show in this section that our positive relativization results are the best possible short of settling open questions about the nonrelativized complexity classes. Positive relativizations for the class $\mathcal{U}$ are given in Section 6.

The research reported in these several papers imparts a deep understanding of nondeterministic oracular computations and of resultant reduction classes. Furthermore, these results clarify the so-called "Baker-Gill-Solovay phenomenon," i.e., the situation of having oracles A and B relative to which the $P = ? NP$ problem has a

positive and negative solution, respectively. With respect to this paper specifically, recall that Baker, Gill, and Solovay [1] have constructed oracles A and B such that $P(A) = NP(A) \cap co\text{-}NP(A)$ but $P(B) \neq NP(B) \cap co\text{-}NP(B)$, and that Rackoff [8] has constructed oracles C and D for which $P(C) = \mathcal{U}(C) \neq NP(C)$ and $P(D) \neq \mathcal{U}(D) = NP(D)$. Also, an oracle E such that $P(E) \neq \mathcal{U}(E) \neq NP(E)$ has been obtained by Geske [5]. Now we know what properties of the standard oracle machine model cause such pathologies to occur and, consequently, now we know what restrictions prevent such pathologies.

We conclude this introduction with a few remarks concerning notation. Namely, notation is standard and consistent with the prior papers. Unless specified otherwise, all sets are languages over the finite alphabet $\Sigma = \{0, 1\}$. The length of a string x in Σ^* is denoted $|x|$.

For a set S, $\|S\|$ denotes the cardinality of S. Let $<$ denote any standard polynomial time computable total order defined on Σ^*. For a nonempty finite set $S \subset \Sigma^*$, say $S = \{y_1, ..., y_n\}$, where $i < j$ implies $y_i < y_j$, let $c(S) = \%y_1\%...\%y_n\%$, where % is a symbol not in Σ. Let $c(\varnothing) = \%$. We consider c to be an encoding function. Notice that if $S \subset \Sigma^*$ is a finite set and $y \in \Sigma^*$, then the predicate "y is in S" can be computed in polynomial time from the inputs y and $c(S)$.

Let $\langle , \rangle$ denote any fixed polynomial time computable pairing function with polynomial time computable inverses.

An oracle Turing machine is a multitape Turing machine with a distinguished query tape and three distinguished states QUERY, YES, and NO. Oracle Turing machines to be considered in this paper are endowed with distinguished accepting states and with distinguished rejecting states. Given an oracle Turing machine and an oracle A, $L(M, A)$ will denote the set of input strings accepted by M with A as its oracle.

2. Computing Functions

The proofs of our main results employ techniques that relate efficient set acceptors with efficient computation of functions by transducers. These techniques will be described here, and they have independent interest.

Consider nondeterministic transducers with accepting states. A transducer T computes a value y on an input x if there is an accepting computation of M on x for which y is the final contents of M's output tape. Note that, in general, such transducers compute partial, multivalued functions.

Given a partial, multivalued function f, define set-f by set-$f(x) = \{y \mid y$ is a value of $f(x)\}$, for all x. If $\|\text{set-}f(x)\|$ is finite for each x, then the functon $c(\text{set-}f)$ is defined by $c(\text{set-}f)(x) = c(\text{set-}f(x))$. For each x, $c(\text{set-}f)(x)$ is a string encoding of the set of all words y such that y is a value of $f(x)$. Also, note that $c(\text{set-}f)$ is a single-valued total function.

Definition 2.1. (i) NPMV is the the set of all partial, multivalued functions computed by nondeterministic polynomial time-bounded transducers.

(ii) NPSV is the set of all $f \in$ NPMV that are single-valued.

(iii) PF is the set of all partial single-valued functions computed by deterministic polynomial time-bounded transducers.

If $\mathscr{F}$ is an arbitrary class of partial functions, let $\mathscr{F}_t$ be the set of all total functions in $\mathscr{F}$. (This notation is due to Valiant [12] and the following propositions are in the spirit of the work in [12].)

The following two propositions are proved in [3].

PROPOSITION 2.2. *$P = NP$ if and only if* NPSV $\subseteq$ PF.

PROPOSITION 2.3. *For any multivalued function f of one argument, define the single-valued function g as follows*:

$$g(x, 0^n) = c(\text{set-}f)(x), \text{ if } \|\text{set-}f(x)\| \leqslant n;$$
$$= \text{undefined, otherwise.}$$

If f is in NPMV *and $NP = co\text{-}NP$, then $g \in$* NPSV *and domain* $(g) \in NP \cap co\text{-}NP$.

It is of interest to compare the following Proposition 2.4 with Proposition 2.2. Also, whereas it is obvious that $P = NP \cap co\text{-}NP$ if and only if every characteristic function in NPSV is contained in PF, it is not so apparent that this holds for all total functions in NPSV. This result will be used several times.

PROPOSITION 2.4. *$P = NP \cap co\text{-}NP$ if and only if* $\text{NPSV}_t \subseteq$ PF.

Proof. The proof of right to left is easy. If $L \in NP \cap co\text{-}NP$, then the characteristic function of L, Ch_L, belongs to NPSV_t. Since $\text{NPSV}_t \subseteq$ PF is assumed, $\text{Ch}_L \in$ PF and, therefore, $L \in P$.

Assume now that $P = NP \cap co\text{-}NP$. Let $f \in \text{NPSV}_t$. Define graph$(f) = \{\langle x, f(x)\rangle \mid x \in \Sigma^*\}$ and pregraph$(f) = \{\langle x, y\rangle \mid y$ is a prefix of $f(x)$, $x \in \Sigma^*\}$. The set pregraph(f) is in $NP \cap co\text{-}NP$ as witnessed by a nondeterministic machine that behaves as follows:

On input $\langle x, y\rangle$, nondeterministically compute $f(x)$. From any accepting computation with $f(x)$ on the output tape, determine whether y is a prefix of $f(x)$. If y is a prefix of $f(x)$, then accept; otherwise reject.

Given an input string x, the following procedure employs pregraph(f) in a typical self-reduction technique to compute $f(x)$. Namely, pregraph(f) is queried with inputs $\langle x, 0\rangle$ and $\langle x, 1\rangle$ to determine whether the first bit b_1 of $f(x)$ is 0 or 1. Then, pregraph(f) is queried with inputs $\langle x, b_1 0\rangle$ and $\langle x, b_1 1\rangle$ to determine the second bit, etc.

```
begin
input x;
y := e {the empty string}
while ⟨x, y0⟩ ε pregraph(f) or ⟨x, y1⟩ ε pregraph(f) do
    begin
    if ⟨x, y0⟩ ε pregraph(f)
        then y := y0
        else y := y1
    end;
output y
end.
```

Assuming $P = NP \cap co\text{-}NP$, pregraph(f) is in P so the procedure is deterministic and runs in polynomial time. Clearly this procedure computes f, so that $f \in \mathrm{PF}_t$. ∎

3. $P = ?\ NP \cap co\text{-}NP$

We come now to our main result. Let M be an arbitrary oracle Turing machine. Define the partial multivalued function NEXT-CALL$_M$ on configurations of M by

> J is a value of NEXT-CALL$_M(I)$ if some computation of M beginning in configuration I reaches configuration J, M is in the QUERY state in configuration J, and in that computation no configuration prior to J is in the QUERY state.

It is clear that if M is deterministic, then NEXT-CALL$_M$ is single-valued. If M is deterministic and operates in polynomial time, then NEXT-CALL$_M$ is in *PF*. If M is nondeterministic and operates in polynomial time, then NEXT-CALL$_M$ is in NPMV.

Define an oracle machine to be *confluent* if NEXT-CALL$_M$ is single-valued. If M is confluent and NEXT-CALL$_M(I) = J$, then there can be only two types of computation paths branching out from I—those that lead to the query configuration J and those that do not lead to queries at all. It is clear that if M is nondeterministic and confluent, and M operates in polynomial time, then NEXT-CALL$_M$ is in NPSV.

This notion was introduced in [11], where it was shown that $P = NP$ if and only if for every set A, $P(A)$ is identical to the class of languages accepted in polynomial time relative to A by confluent nondeterministic oracle machines.

Let M be a confluent oracle machine and consider computations of M relative to a fixed oracle A. Given an input string x, let k be the maximum number of queries made to the oracle by some computation of M relative to A on input x. Then, there is a unique sequence

$$\{\mathrm{QUERY}_M^A(j, x)\}_{j \leq k}$$

such that each $\text{QUERY}_M^A(j, x)$, $j \leqslant k$, is a query configuration of M, and for every computation C of M relative to A on input x, if C makes l queries to the oracle A (hence, $l \leqslant k$), then the query configurations of C are exactly the configurations

$$\text{QUERY}_M^A(1, x), \ldots, \text{QUERY}_M^A(l, x),$$

and they occur in C in this order. This notation will be very useful in our proofs.

This sequence is defined inductively as follows:

$\text{QUERY}_M^A(1, x) = \text{NEXT-CALL}_M(I_0)$, where I_0 is the initial configuration of M on input x;

$\text{QUERY}_M^A(j+1, x) = \text{NEXT-CALL}_M(J)$, where J is the unique answer configuration for the query configuration $\text{QUERY}_M^A(j, x)$ and oracle A.

Given an arbitrary nondeterministic oracle machine M, let $Q(M, A, x)$ denote the set of all strings queried in the entire tree of computations of M on input x with oracle set A. The following proposition follows directly from the observations of the previous paragraph.

PROPOSITION 3.1. *For every nondeterministic confluent oracle machine M that operates in polynomial time, there exists a polynomial q such that for each oracle set A and for all input strings x,*

$$\| Q(M, A, x) \| \leqslant q(|x|).$$

Define unary relations ACCEPT_M and REJECT_M on configurations of M by

$\text{ACCEPT}_M(I) = \textit{true}$ if some computation of M starting in configuration I enters an accepting configuration without going through a query configuration;

$\text{REJECT}_M(I) = \textit{true}$ if some computation of M starting in configuration I enters a rejecting configuration without going through a query configuration.

Trivially, if I is a query configuration of M, then $\text{ACCEPT}_M(I) = \textit{false}$ and $\text{REJECT}_M(I) = \textit{false}$.

Define an oracle machine to be *mature* if for every nonquery configuration I of M, if I leads either to an accepting computation or to a rejecting computation without reaching a query configuration, then $\text{NEXT-CALL}_M(I)$ is underfined; in symbols, M is mature if and only if for every nonquery configuration I,

$$[\text{ACCEPT}_M(I) \vee \text{REJECT}_M(I) \rightarrow \text{NEXT-CALL}_M(I) \text{ is undefined}].$$

A nondeterministic oracle machine may have many different computations relative to an oracle set on any one input; M is mature if it never consults its oracle when it can solve its problems without the oracle. Deterministic machines are mature. Maturity is obviously a desirable property for nondeterministic oracle

machines to have, and it is not known whether every nondeterministic machine is equivalent to a mature one. We do prove in the next section, however, that $NP = \text{co-}NP$ implies that every polynomial time oracle machine is equivalent to a mature polynomial time oracle machine.

Recall from the Introduction that an oracle machine is *strong* if, relative to each oracle set, for each input string w, either there is an accepting computation of M on w or there is a rejecting computation of M on w, but there is not both an accepting computation and a rejecting computation of M on w.

THEOREM 3.2. *$P = NP \cap co\text{-}NP$ if and only if for every set D, $P(D)$ is identical to the class of languages accepted in polynomial time relative to D by nondeterministic oracle machines that are confluent, mature, and strong.*

The proof from right to left is immediate because $P(\varnothing) = P$, and the class of languages accepted by confluent, strong, mature oracle machines with the empty set as the oracle is $NP \cap co\text{-}NP$.

The proof from left to right is similar to the proof of [Theorem 4.5, 3]. The general idea, given a confluent, mature, and strong oracle machine M and choice of oracle set D, is to nondeterministically construct "tables" $T_Y = Q(M, D, x) \cap D$ and $T_N = Q(M, D, x) \cap \bar{D}$ upon input x and to simulate M on x with the help of these tables. By Proposition 3.1 the tables are not large, and so this is feasible. The tables T_Y and T_N must be constructed iteratively. At each iteration a new query y not already in $T_Y \cup T_N$ is found by a nondeterministic computation

$$\text{FIND}_M(x, c(T_Y), c(T_N)).$$

The function FIND_M is defined below. Confluence of M will guarantee that FIND_M is single-valued; after each iteration the oracle D is used to place the output string of $\text{FIND}_M(x, c(T_Y), c(T_N))$ correctly into one of the tables T_Y or T_N.

The function FIND_M actually does somewhat more than claimed thus far. Since M is strong, some computation of M on x relative to D will terminate in an accepting configuration or in a rejecting configuration. FIND_M will detect which of these cases occurs, thereby guaranteeing successful termination of our simulation.

The function FIND_M is defined as follows: For each input string x of M and each pair T_Y and T_N of finite sets of strings,

(1) *accept* is a value of $\text{FIND}_M(x, c(T_Y), c(T_N)$ if there is an accepting computation C of M on input x such that if w is any string queried during computation C, then

(a) $w \in T_Y \cup T_N$, and

(b) the answer used by computation C for the query about string w is YES if and only if $w \in T_Y$;

(2) *reject* is a value of $\text{FIND}_M(x, c(T_Y), c(T_N))$ if there is a rejecting computation C of M on input x such that (a) and (b) of (1) hold for C also;

(3) a string y is a value of $\text{FIND}_M(x, c(T_Y), c(T_N))$ if there is a computation C of M on input x such that

(a) string y is queried during computation C,

(b) $y \notin T_Y \cup T_N$, and

(c) if w is any string queried during computation C before y is queried, then $w \in T_Y \cup T_N$ and the answer used by C for the query about w is YES if and only if $w \in T_Y$.

Note that in this definition of FIND_M, it may be that $T_N \cap T_Y \neq \emptyset$. If $w \in T_Y \cap T_N$, then queries about w are answered as if $w \in T_Y$ only. Thus, YES answers are given for elements of T_Y and NO answers are given for elements of $T_N - T_Y$.

LEMMA 3.3. *If M is a confluent, mature, strong oracle machine that operates in polynomial time, then* $\text{FIND}_M \in \text{NPSV}_t$.

Proof. We show first that FIND_M is a total function. Suppose that $f(x, c(T_Y), c(T_N))$ is not defined for some x, $c(T_Y)$, and $c(T_N)$. Let A be any set such that $T_Y \subseteq A$ and $T_N - T_Y \subseteq \bar{A}$. Since no string y is a value of $\text{FIND}_M(x, T_Y, T_N)$, rule (3) implies that all strings queried by M on input x with oracle set A are in $T_Y \cup T_N$. Then, by rules (1) and (2), every computation of M on x with oracle set A halts in a state that is nonaccepting and nonrejecting. This contradicts the assumption that M is a strong oracle machine. Therefore, FIND_M is a total function.

The proof that FIND_M is single-valued is divided into three parts.

(1) The constant values *accept* and *reject* are not both values of $\text{FIND}_M(x, c(T_Y), c(T_N))$ because M is a strong oracle machine.

(2) If y_1 and y_2 are both values of $\text{FIND}_M(x, c(T_Y), c(T_N))$, then $y_1 = y_2$. To see this, let C_1 be a computation of M on x that causes the string y_1 to be an output value of $\text{FIND}_M(x, c(T_Y), c(T_N))$ according to rule (3). Thus, C_1 eventually queries y_1, every string w queried in C_1 before y_1 is queried belongs to $T_Y \cup T_N$, and for each such w, the answer used by C_1 is YES if and only if $w \in T_Y$. Let k be the number of times C_1 enters a query configuration prior to the first query about y_1. Letting A be any set such that $T_Y \subseteq A$ and $T_N - T_Y \subseteq \bar{A}$, it is clear that the first $k+1$ query configurations of C are exactly the configurations

$$\text{QUERY}_M^A(1, x), \ldots, \text{QUERY}_M^A(k+1, x);$$

that in each configuration $\text{QUERY}_M^A(j, x)$, $j \leqslant k$, a query is made about some string in the set $T_Y \cup T_N$; and that y_1 is queried in configuration $\text{QUERY}_M^A(k+1, x)$. Therefore, since M is confluent, if C_2 is any computation that causes a string y_2 to

be a value of $\text{FIND}_M(x, c(T_Y), c(T_N))$, it follows that C_2 must contain the query configurations $\text{QUERY}_M^A(1, x)$,..., $\text{QUERY}_M^A(k, x)$, $\text{QUERY}_M^A(k+1, x)$ as well (because in each of the configurations $\text{QUERY}(j, k)$, $j \leqslant k$, the string w that is queried is in $T_Y \cup T_N$, and the answers used by computations C_1 and C_2 about w are identical). Since the first query y_2 during computation C_2 that does not belong to $T_Y \cup T_N$ is made in configuration $\text{QUERY}_M^A(k+1, x)$, we conclude that $y_1 = y_2$.

(3) If either *accept* or *reject* is a value of $\text{FIND}_M(x, c(T_Y), c(T_N))$, then no string y is a value via rule (3). Let C be a computation that satisfies either rule (1) or rule (2), so that C terminates in either an accepting configuration or in a rejecting configuration, every word w queried during computation C is in $T_Y \cup T_N$, and the answer used by C about the query w is YES if and only if $w \in T_Y$. Let k be the number of query configurations occurring in C, and let A be any set such that $T_Y \subseteq A$ and $T_N - T_Y \subseteq \bar{A}$. There are two cases to consider. If $k = 0$, let I denote the initial configuration of M on input x. If $k > 0$, then the query configurations occurring in C are the configurations $\text{QUERY}_M^A(1, x)$,..., $\text{QUERY}_M^A(k, x)$, and in each of these configurations a word belonging to $T_Y \cup T_N$ is queried. In this case, let I denote the successor configuration in C of $\text{QUERY}_M^A(k, x)$. Now observe that in either case, $\text{ACCEPT}_M(I)$ or $\text{REJECT}_M(I)$ holds. Therefore, since M is a mature oracle machine, $\text{NEXT-CALL}_M(I)$ is undefined. Since every computation that answers YES to all queries in T_Y and answers NO to all queries in $T_N - T_Y$ must contain the sequence of query configurations $\text{QUERY}_M^A(1, x)$,..., $\text{QUERY}_M^A(k, x)$, it follows immediately that no such computation can cause a string y to be a value of $\text{FIND}_M(x, c(T_Y), c(T_N))$ via rule (3).

We have shown now that FIND_M is total and single-valued. Obviously, FIND_M is computable nondeterministically in polynomial time. Hence $\text{FIND}_M \in \text{NPSV}_t$. ∎

Proof of Theorem 3.2 *Concluded.* Assume that $P = NP \cap co\text{-}NP$ and let $L = L(M, D)$ for arbitrary oracle set D and confluent, mature, strong oracle machine M which runs in polynomial time. The lemma shows that the function $\text{FIND}_M \in \text{NPSV}_t$; therefore, by use of Proposition 2.4, $\text{FIND}_M \in \text{PF}_t$. That is, FIND_M can be computed deterministically in polynomial time. The following algorithm recognizes L deterministically in polynomial time relative to D, i.e., shows that $L \in P(D)$.

```
begin
input x;
T_Y := T_N := ∅;
while FIND_M(x, c(T_Y), c(T_N)) ∉ {accept, reject} do
    begin
    y := FIND_M(x, c(T_Y), c(T_N));
    if y ∈ oracle set
      then T_Y := T_Y ∪ {y}
      else T_N := T_N ∪ {y}
    end;
```

if $\text{FIND}_M(x, c(T_Y), c(T_N) = accept$
then accept
else reject
end.

By Proposition 3.1, for each oracle set A, $\|Q(M, A, x)\| \leqslant q(|x|)$, where q is a polynomial. Therefore, the algorithm runs in polynomial time and for each oracle set A, $T_Y \subseteq Q(M, A, x) \cap A$ and $T_N \subset Q(M, A, x) \cap \bar{A}$ throughout the execution of the algorithm. Thus, the correctness of this algorithm for L, when D is the oracle, follows immediately. ∎

Theorem 3.2 shows that the restrictions confluent, mature, and strong yield a positive relativization of the $P = ?\, NP \cap co\text{-}NP$ question. These restrictions are technical to be sure; they are nevertheless reasonably natural restrictions to impose. Russo and Zachos [9] have recently obtained a positive relativization based on a simpler simulation than ours, but the constraints they place on oracle machines are more restrictive than are ours. We contend that no substantially more general positive relativization of the $P = ?\, NP \cap \text{co-}NP$ question is possible, and this issue is taken up in detail in Section 5 below.

4. Relations Between Qualitative Classes

Now we will consider reduction classes that are defined by oracle machines possessing some combination of the attributes "confluent, mature, and strong." We will obtain separation results for classes defined by certain combinations of these attributes and we will see that other combinations of these attributes yield positive relativization results.

We introduce the following notation for representing these classes. Abbreviate the properties confluent, mature, and strong by the letters C, M, and S, respectively. Then, NPC(A), where A is a set, denotes the classes of languages $L \in NP(A)$ that are accepted in polynomial time relative to A by nondeterministic oracle machines that are confluent; NPCM(A) is the subset of $NP(A)$ defined by polynomial time-bounded oracle machines that are both confluent and mature, and so forth. As three properties are under consideration, seven reduction classes are defined: NPC, NPM, NPS, NPCM, NPCS, NPMS, and NPCMS.

The positive relativization $P = NP$ if and only if $P(A) = \text{NPC}(A)$, for every set A, is proved in [11]. The main result proved in the last section stated in this notation is:

$P = NP \cap co\text{-}NP$ if and only if for every set A, $P(A) = \text{NPCMS}(A)$.

For an oracle machine M, set A, and input string x, recall that $Q(M, A, x)$ denotes the set of all strings queried in the entire tree of computations of M on x with oracle A. In [3], the relativization $NP_B(\ )$ is studied; a language L belongs to $NP_B(A)$ if and only if L is accepted by a nondeterministic polynomial time oracle machine M such that for some polynomial p and all x, $\|Q(M, A, x)\| \leqslant (p\,|x|)$.

The following two results about $NP_B(\ )$ are proved in [3].

PROPOSITION 4.1. *$P = NP$ if and only if for every set D, $P(D) = NP_B(D)$.*

PROPOSITION 4.2. *$NP = co\text{-}NP$ if and only if for every set D, $NP(D) = co\text{-}NP_B(D)$.*

Proposition 3.1 is concisely expressed in this notation also: For every set A, $\mathrm{NPC}(A) \subseteq NP_B(A)$.

Is every nondeterministic polynomial time-bounded oracle machine equivalent to a mature machine that operates in polynomial time?

THEOREM 4.3. *If $NP = co\text{-}NP$, then for every nondeterministic polynomial time-bounded oracle machine M_1, there exists a mature nondeterministic polynomial time-bounded oracle machine M_2 such that $L(M_2, D) = L(M_1, D)$, for every set D. Furthermore, if M_1 is strong, then M_2 is strong, if M_1 is confluent, then M_2 is confluent, and if for some oracle set D there is a polynomial such that $\| Q(M_1, D, x)\| \leqslant q(|x|)$ for all x, then $\| Q(M_2, D, x)\| \leqslant q(|x|)$ for all x.*

Proof. Assume $NP = co\text{-}NP$ and let M_1 be an arbitrary nondeterministic polynomial time-bounded oracle machine. Define the set OKCON as follows: Configuration I of M_1 is an element of OKCON if and only if there is either an accepting computation or a rejecting computation of M_1 that begins at I and does not query the oracle. Clearly, $\mathrm{OKCON} \in NP$ and, since we assume $NP = co\text{-}NP$, $\mathrm{OKCON} \in NP \cap co\text{-}NP$. Let M_0 be a nondeterministic polynomial time-bounded oracle machine that recognizes OKCON and is witness to $\mathrm{OKCON} \in NP \cap co\text{-}NP$.

Without loss of generality, assume that M_1 has nondeterministic fan-out two, so that from any configuration I, there are at most two successor configurations, left(I) and right(I). Now let M_2 be a nondeterministic polynomial time-bounded oracle machine which behaves as follows. (The basic idea is to implement a depth-first search of the computation tree of M_1 on input x. When a configuration I is visited, OKCON is used to help determine which of the successors, left (I) or right (I), is to be visited next.)

On input x, begin a nondeterministic simulation of a computation of M_1 on input x. Every time this simulation enters a nonquery configuration I from which there are two possible successors, M_2 first simulates M_0 on left(I). If M_0 accepts left(I), then M_2 continues its simulation from I by moving to left (I) only. If M_0 rejects left(I), then M_2 next simulates M_0 on right(I). If M_0 accepts right(I), then M_2 continues its simulation from I by moving to right(I) only. If M_0 rejects right(I), then M_2 continues its simulation from I by nondeterministically moving to either left(I) or right(I). Finally, M_2 accepts x (rejects x) only when it finds a computation of M_1 which accepts x (rejects x, respectively).

The machine M_2 behaves like M_1 except that M_2 always checks to see if accepting or rejecting without any queries is possible before making a nondeterministic step. If so, then M_2 makes no more queries and therefore is mature. It is

clear that for all sets D, $L(M_2, D) = L(M_1, D)$. The reader may easily verify the other claims for M_2. ∎

COROLLARY 4.4. *Statement* (a) *implies statements* (b) *through* (e).

(a) $NP = co\text{-}NP$;

(b) *for every set* D, $NP(D) = \mathrm{NPM}(D)$;

(c) *for every set* D, $\mathrm{NPS}(D) = \mathrm{NPMS}(D)$;

(d) *for every set* D, $\mathrm{NPC}(D) = \mathrm{NPCM}(D)$;

(e) *for every set* D, $\mathrm{NPCS}(D) = \mathrm{NPCMS}(D)$.

Any class specified by strong oracle machines is closed under complementation and therefore does not relativize the "$NP = ?\ co\text{-}NP$" question. However, the classes NPC and NPCM which are not a priori closed under complementation can be used to relativize the "$NP = ?\ co\text{-}NP$" question. This follows as a corollary to the next theorem and to results in [3, 11].

THEOREM 4.5. *$NP = co\text{-}NP$ if and only if for every set D, $\mathrm{NPCMS}(D) = NP_B(D)$.*

Proof. If $\mathrm{NPCMS}(\varnothing) = NP_B(\varnothing)$, then, since $NP_B(\varnothing) = NP$, it follows that $\mathrm{NPCMS}(\varnothing) = NP$. Since every class specified by strong oracle machines is closed under complementation, $\mathrm{NPCMS}(\varnothing) = NP_B(\varnothing)$ implies $NP = co\text{-}NP$. Thus, the proof in one direction is complete.

By use of Proposition 3.1, for every set D, $\mathrm{NPCMS}(D) \subseteq \mathrm{NPC}(D) \subseteq NP_B(D)$. Thus, we must show $NP = co\text{-}NP$ implies for every set D, $NP_B(D) \subseteq \mathrm{NPCMS}(D)$. By Corollary 4.4, it suffices to prove $NP = co\text{-}NP$ implies $NP_B(D) \subseteq \mathrm{NPCS}(D)$. Assume that $NP = co\text{-}NP$ and let $L \in NP_B(D)$ for arbitrary D. Let nondeterministic oracle machine M_1 witness $L \in NP_B(D)$ within polynomial time-bound p, and let polynomial q bound $\|Q(M_1, D, x)\|$; i.e., for all x, $\|Q(M, D, x)\| \leqslant q(|x|)$. The technique used in [3] to prove Proposition 4.2 will be invoked now in order to obtain a nondeterministic polynomial time-bounded oracle machine M_2 such that $L = L(M_2, D)$ and such that M_2 is confluent and strong.

As in the proof of Theorem 3.2, the basic idea is to iteratively construct tables $T_Y = Q(M_1, D, x) \cap D$ and $T_N = Q(M_1, D, x) \cap \bar{D}$ upon input x and then to simulate M_1 without further use of D. The hypothesis $NP = co\text{-}NP$ will enable the simulation to recognize both L and $\bar{L}$, so that as a consequence M_2 will be strong. Also, the procedure that constructs T_Y and T_N will make queries to the oracle in a strictly sequential manner, so that as a consequence M_2 will be confluent.

We begin by defining a multivalued function f as follows: For each input string x of M, each pair T_Y and T_N of finite sets of strings, and each natural number $k > 0$, string y is a value of $f(x, c(T_Y), c(T_N), 0^k)$ if and only if there is a computation C of M_1 on x such that

(i) the kth time that C enters the QUERY state, y is the string on the query tape; and

(ii) if w is any string queried during the first $k-1$ times that C enters the QUERY state, then $w \in T_Y \cup T_N$, and the answer used by C to the query about w is YES if and only if $w \in T_Y$.

It is clear that $f \in$ NPMV. As long as $T_Y \subseteq D$ and $T_N \subseteq \bar{D}$, then $f(x, c(T_Y), c(T_N), 0^k) \subseteq Q(M_1, D, x)$ and so $\|\text{set-}f(x, c(T_Y), c(T_N), 0^k)\| \leqslant \|Q(M_1, D, x)\| \leqslant q(|x|)$. Therefore, letting g be the functon in NPSV with domain$(g) \in NP \cap co\text{-}NP$ obtained from f by Proposition 2.3, $g(\langle x, c(T_Y), c(T_N), 0^k\rangle, 0^{q(|x|)}) = c(\text{set-}f(x, c(T_Y), c(T_N), 0^k))$ when $T_Y \subseteq D$ and $T_N \subseteq \bar{D}$.

In the following oracle procedure S is a program variable (of type string, but used to encode a finite set). Recall from the definition that for any finite set A, $c(A)$ is an ordered list.

```
    begin
    input x;
    T_Y := T_N := ∅;
(1) for k := 1 to p(|x|) do
(2)     if g(⟨x,c(T_Y),c(T_N),0^k⟩,0^{q(|x|)}) is defined
        then begin
(3)         S := g(⟨x,c(T_Y),c(T_N),0^k⟩,0^{q(|x|)});
(4)         for y := first element of S to last element of S do
                if y ∈ oracle set
                then T_Y := T_Y ∪ [y]
                else T_N := T_N ∪ [y]
            end;
(5) if there exists an accepting computation of M_1 on input x of length at most
        p(|x|) such that for every word y that is queried in this computation,
        y ∈ T_Y ∪ T_N and the answer to the query is YES if and only if y ∈ T_Y
        then accept
        else reject
end.
```

Consider the implementation and running time of this procedure. Under the hypothesis $NP = co\text{-}NP$, $g \in$ NPSV, and domain$(g) \in NP \cap co\text{-}NP$. Thus, the test at line 2 and the function evaluation at line 3 can be carried out nondeterministically in polynomial time. The test at line 5 is in NP and therefore is in $NP \cap co\text{-}NP$ also. For each execution of the outer-loop at line 1, the value of S at line 3 is a sorted encoding of at most $q(|x|)$ strings. Thus, each execution of the inner for-loop takes at most polynomial time (relative to the oracle set). Since the outer for-loop iterates $p(|x|)$ times, the entire procedure can be implemented to run nondeterministically in polynomial time relative to the oracle set.

The correctness of the procedure when using oracle D depends on the fact that when execution reaches line (5), $T_Y = Q(M_1, D, x) \cap D$ and $T_N = Q(M_1, D, x) \cap \bar{D}$. Thus, the test at line 5 correctly determines membership in either L or $\bar{L}$.

Let M_2 be a nondeterministic oracle machine that implements this procedure in

polynomial time. Clearly, $L(M_1, D) = L(M_2, D)$. Since the for-loop at line 4 causes M_2 to query the members of S in order, M_2 is confluent. That M_2 is strong follows from the fact that for every oracle D, the test at line 5 is in $NP \cap co\text{-}NP$. Hence, he proof is complete. ∎

The next corollary follows from the obvious inclusions

$$\text{NPCMS}(D) \subseteq \text{NPCS}(D) \subseteq \text{NPC}(D) \subseteq NP_B(D)$$

and

$$\text{NPCMS}(D) \subseteq \text{NPCM}(D) \subseteq \text{NPC}(D) \subseteq NP_B(D),$$

for every set D.

COROLLARY 4.6. *$NP = co\text{-}NP$ if and only if for every set D,*

$$\text{NPCMS}(D) = \text{NPCS}(D) = \text{NPCM}(D) = \text{NPC}(D) = NP_B(D).$$

Thus, if there is an oracle that separates any two of the confluent classes, then $NP \neq co\text{-}NP$.

COROLLARY 4.7. *The following statements are equivalent:*

(a) $NP = co\text{-}NP$;

(b) *for every set D,* $\text{NPC}(D) = \text{co-NPC}(D)$;

(c) *for every set D,* $\text{NPCM}(D) = \text{co-NPCM}(D)$.

This corollary follows from Proposition 4.2 and the corollary just given. The equivalence of (a) and (b) is proved in [11], while (a) and (c) give a new positive relativization of the $NP = ?$ co-NP question.

COROLLARY 4.8. *$P = NP$ if and only if for every set D, $P(D) = \text{NPCMS}(D) = \text{NPCS}(D) = \text{NPCM}(D) = \text{NPC}(D) = NP_B(D)$.*

This is a consequence of Proposition 4.1, the inclusion $P(D) \subseteq \text{NPCMS}(D)$ for any set D, and the inclusions listed above. Using Proposition 4.1 again, we have the following positive relativizations of the $P = ?$ NP question, where the equivalence of (a) and (b) appears in [11] and the equivalence of (a) and (c) is new.

COROLLARY 4.9. *The following are equivalent:*

(a) $P = NP$;

(b) *for every set D,* $P(D) = \text{NPC}(D)$;

(c) *for every set D,* $P(D) = \text{NPCM}(D)$.

COROLLARY 4.10. *Statement* (a) *implies both statement* (b) *and statement* (c):

(a) $P = NP$;

(b) *for every set* D, $P(D) = \mathrm{NPCS}(D)$;

(c) *for every set* D, $P(D) = \mathrm{NPCMS}(D)$.

In Corollary 4.10, we do not know whether statement (b) or statement (c) implies $P = NP$. This is because $\mathrm{NPSC}(\varnothing) = \mathrm{NPCMS}(\varnothing) = NP \cap co\text{-}NP$, and it is an open question whether $P = NP \cap co\text{-}NP$ implies $P = NP$.

We do not have positive relativizations depending on NPS( ), NPM( ), or NPMS( ). One reason for this is that the set $Q(M, D, x)$ cannot be controlled by using only the mature and/or strong restrictions. The confluent restriction, however, allows the size of $Q(M, D, x)$ to be controlled, since Proposition 3.1 gives $\mathrm{NPC}(D) \subseteq NP_B(D)$.

Baker, Gill, and Solovay [1] constructed recursive sets C and D such that $C \in NP(D)$ and $C \notin co\text{-}NP$. It follows from that construction that

(i) $C \in \mathrm{NPM}(D)$, while

(ii) $C \notin \mathrm{NPS}(D)$ and $C \notin NP_B(D)$.

These observations imply therefore that the following relativizations differ over the recursive sets:

(i) $NP(\ )$ and $NP_B(\ )$,

(ii) $NP(\ )$ and NPS( ),

(iii) NPM( ) and NPMS( ),

(iv) NPM( ) and NPCM( ).

For (iv), note that for every set D, $\mathrm{NPCM}(D) \subseteq NP_B(D)$ and so $C \notin \mathrm{NPCM}(D)$. The inclusion relations for the classes discussed in this section relative to an arbitrary oracle are given in Fig. 1.

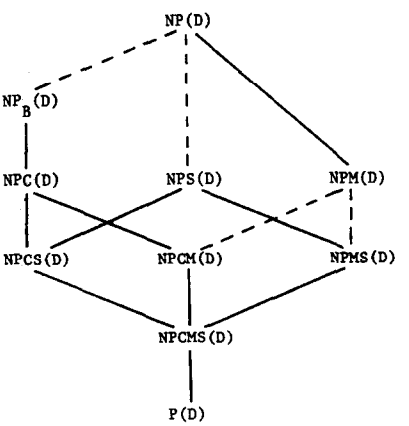

FIG. 1. $\mathscr{L}_1(D)/\mathscr{L}_2(D)$ indicates that for all D, $\mathscr{L}_1(D) \subseteq \mathscr{L}_2(D)$. $\mathscr{L}_1(D)/\mathscr{L}_2(D)$ indicates that for all D, $\mathscr{L}_1(D) \subseteq \mathscr{L}_2(D)$ and for some E, $\mathscr{L}_1(E) \neq \mathscr{L}_2(E)$.

5. $P = ?\ NP \cap co\text{-}NP$ Revisited

In light of the classes defined and studied in the last section, it is natural to inquire again whether some weakening of the requirements "confluent, mature, and strong" can yield a positive relativization of the $P = ?\ NP \cap co\text{-}NP$ question. We will show that this issue is intimately related to an open question raised by Valiant [12] and to unknown properties of the mathematical structure of *NP*.

Let us agree for the moment to keep the attribute "strong," for surely we want our relativized classes to be closed under complements. Define the operator $\mathrm{NPS}_B(\ )$ so that $\mathrm{NPS}_B(D)$ is the subset of $NP_B(D)$ consisting of all languages L in $NP_B(D)$ whose membership in $NP_B(D)$ is witnessed by a strong oracle machine. The main results of [3], Propositions 4.1 and 4.2, suggest that if a quantitative relativization of "$P = ?\ NP \cap co\text{-}NP$" exists, then $\mathrm{NPS}_B(\ )$ is the likely candidate.

Statement (a) of the following theorem is an open question raised by Valiant [12]. If this assertion is correct, then $\mathrm{NPS}_B(\ )$ relativizes $P = ?\ NP \cap co\text{-}NP$, and this is our strongest result concerning a possible quantitative relativization.

THEOREM 5.1. *Statement* (a) *implies statements* (b) *and* (c):

(a) *If* $P = NP \cap co\text{-}NP$, *then for every function* $f \in \mathrm{NPMV}_t$ *there exists a function* $g \in \mathrm{PF}_t$ *such that for all strings* x, $g(x)$ *is a value of* $f(x)$ (*i.e.*, g *is a single-valued restriction of* f).

(b) $P = NP \cap co\text{-}NP$ *if and only if for every set* D, $P(D) = \mathrm{NPS}_B(D)$.

(c) $P = NP \cap co\text{-}NP$ *if and only if for every set* D, $P(D) = \mathrm{NPSC}(D)$.

Proof. We need prove only that statement (a) implies the implications from left to right in statements (b) and (c). Therefore, let us assume that (a) is correct and that $P = NP \cap co\text{-}NP$. Suppose that $L \in \mathrm{NPS}_B(D)$ is witnessed by oracle machine M. Consider the function FIND_M defined in the proof of Theorem 3.2 and observe from the proof of Lemma 3.3 that FIND_M is total when M is a strong nondeterministic oracle machine. Moreover, if for any input string x and finite sets T_Y and T_N, *accept* and *reject* are both not values of $\mathrm{FIND}_M(x, c(T_Y), c(T_N))$, then there must exist at least one string y that is a value. Since $\mathrm{FIND}_M \in \mathrm{NPMV}_t$ there exists a restriction in PF_t, that we henceforth will call FIND_M as well, for we will make no further reference to the multivalued function with which we began.

Now suppose there is a polynomial q such that $\| Q(M, D, x) \| \leqslant q(|x|)$ for all x and consider the procedure in the proof of Theorem 3.2. The argument there shows that the procedure runs in polynomial time, and correctness follows from the observation that $\mathrm{FIND}_M(x, c(T_Y), c(T_N)) \notin \{accept, reject\}$ implies existence of a string y such that $y = \mathrm{FIND}_M(x, c(T_Y), c(T_N))$. Therefore, the claim that (a) implies (b) is proved. It follows immediately that (a) implies (c) because $P(D) \subseteq \mathrm{NPSC}(D) \subseteq \mathrm{NPS}_B(D)$, for all D. ∎

We consider one more issue in this section. Recall once again that an oracle machine M is strong if relative to *every* oracle set, for each input string x, either

there is an accepting computation of M on x or there is a rejecting computation of M on x, and there is not both an accepting computation of M on x and a rejecting computation of M on x. As a refinement of this notion, define the relativization $NP\overline{OS}(\ )$ so that a language $L \in NP\overline{OS}(D)$ if and only if there exists a nondeterministic oracle machine that accepts L relative to D such that for every input string x, relative to the *fixed* oracle D, either there is an accepting computation of M on x relative to D or there is a rejecting computation of M on x relative to D, and there is not both an accepting computation of M on x relative to D and a rejecting computation of M on x relative to D. For every set D, $NP\overline{OS}(D)$ is the reduction class of the strong reducibility $\leqslant_T^{SN}$ relative to D (see [7] for a discussion of $\leqslant_T^{SN}$). It should be clear that $NP\overline{OS}(D)$ is closed under complements and, in addition, that $\mathrm{NPS}(D) \subseteq NP\overline{OS}(D)$ for every D. A diagonalization argument shows that there exist sets L and D such that $L \in NP(D) \cap \text{co-}NP(D)$ but $L \notin \mathrm{NPS}(D)$, and therefore for this set D, $\mathrm{NPS}(D) \neq NP\overline{OS}(D)$. The question we raise is whether $\mathrm{NPCM}\overline{OS}(\ )$ suffices for a positive relativization of $P = ?\ NP \cap \mathrm{co} - NP$. That is, is "$P = NP \cap co\text{-}NP$" equivalent to the assertion "for every set D, $P(D) = \mathrm{NPCM}\overline{OS}(D)$"? We will show, dependent on a plausible conjecture about the mathematical structure of the class NP, that $\mathrm{NPCM}\overline{OS}(\ )$ yields a positive relativization of the $P = ?\ NP$ question and so not of $P = ?\ NP \cap co\text{-}NP$.

In order to motivate the conjecture to be raised, we digress to recall the following two well-known properties about disjoint pairs of recursively enumerable sets:

(1) For every r.e., nonrecursive set A there exist disjoint sets B_1 and B_2 such that $A = B_1 \cup B_2$ and B_1 and B_2 are both r.e. and nonrecursive.

(2) If B_1 and B_2 are disjoint r.e. sets, then $\mathbf{d}(B_1 \cup B_2) = \mathbf{d}(B_1) \vee \mathbf{d}(B_2)$. (For any set B, $\mathbf{d}(B)$ denotes the Turing degree of B.) In particular, $B_1 \leqslant_T B_1 \cup B_2$ and $B_2 \leqslant_T B_1 \cup B_2$.

Now the analogue of (1) to polynomial-time complexity holds. Namely, Ladner [6] has shown that for every set A in $NP - P$, there exist disjoint sets B_1 and B_2 in $NP - P$ such that $A = B_1 \cup B_2$.

Consider the proof of (2). To decide B_1 with oracle $B_1 \cup B_2$, on input x, if $x \notin B_1 \cup B_2$, then reject, else simultaneously enumerate B_1 and B_2 until x is output. Accept if the enumeration of B_1 outputs x and reject if the enumeration of B_2 outputs x.

The proof just given certainly suggests that the analogue of (2) is false. It seems reasonable to conjecture (assuming $P \neq NP$) that there exist disjoint sets B_1 and B_2 in $NP - P$ such that $B_1 \cup B_2 \in NP - P$ and either $B_1 \not\leqslant_T^P B_1 \cup B_2$ or $B_2 \not\leqslant_T^P B_1 \cup B_2$. Neither Ladner's proof [6] of the analogue of 1 nor Schöning's elegant treatment [10] settles this result, for their techniques give $B_1 \leqslant_m^P B_1 \cup B_2$ and $B_2 \leqslant_m^P B_1 \cup B_2$.

Our result can now be stated.

THEOREM 5.2. *Statement* (a) *implies statement* (b):

(a) *$P = NP$ or there exist disjoint sets B_1 and B_2 in $NP - P$ such that $B_1 \cup B_2 \in NP - P$ and either $B_1 \not\leqslant_T^P B_1 \cup B_2$ or $B_2 \not\leqslant_T^P B_1 \cup B_2$.*

(b) *$P = NP$ if and only if for every set D, $P(D) = \mathrm{NPCM}\overline{OS}(D)$.*

Proof. For every set D, $P(D) \subseteq \mathrm{NPCM}\overline{OS}(D) \subseteq \mathrm{NPC}(D)$. Therefore statement (b) from left to right follows immediately from Corollary 4.9.

We now assume $P \neq NP$ and with the help of statement (a) we demonstrate the existence of a set A such that $\mathrm{NPCM}\overline{OS}(A) \not\subseteq P(A)$. Let B_1 and B_2 be disjoint sets in $NP - P$ whose existence is given by (a), and let $A = B_1 \cup B_2$. Without loss of generality assume that $B_1 \not\leq_T^P A$. Let M be an oracle machine that operates in polynomial time and implements the following procedure.

```
input x;
if x ∉ oracle
  then reject
  else if x ∈ B1 {a nondeterministic computation}
       then accept
       else {the path taken did not accept x}
         if x ∈ B2 {nondeterministic}
           then reject
           else halt in a nonaccepting and nonrejecting state.
```

Since the procedure queries exactly one string, M is confluent and mature. Also, M with oracle set A exhibits the strongness properties and $L(M, A) = B_1$. Therefore, $B_1 \in \mathrm{NPCM}\overline{OS}(A)$, but by assumption, $B_1 \notin P(A)$, and so our proof is complete. ∎

6. Relativizing $\mathcal{U}$

In this section we develop positive relativizations of the problems "$\mathcal{U} = ?\ NP$" and "$P = ?\ \mathcal{U} \cap co\text{-}\mathcal{U}$". This being the last section of this paper, our intent is to show that the techniques developed here are applicable in a variety of situations.

The class $\mathcal{U}$ is the collection of all languages L for which there is a nondeterministic Turing machine M that witnesses $L \in NP$ such that for every input word x to M, M has at most one accepting computation [12]. Whether $\mathcal{U} = NP$ is a well-known open problem. It is clear that $P \subseteq \mathcal{U} \subseteq NP$. It is known that there exist sets A, B, and C such that $P(A) \neq \mathcal{U}(A) = NP(A)$ and $P(B) = \mathcal{U}(B) \neq NP(B)$ [8], and such that $P(C) \neq \mathcal{U}(C) \neq NP(C)$ [5].

Extend the definition scheme of Section 4 so that for every set D, $\mathcal{U}\mathrm{CMS}(D)$ denotes the class of languages L such that $L \in \mathcal{U}(D)$ is witnessed by an oracle machine that is confluent, mature, and strong.

Define the class of functions $\mathcal{U}\mathrm{SV}$ to be the set of all $f \in \mathrm{NPSV}$ computed by some polynomial time-bounded transducer T such that for every x and y, if $f(x) = y$, then there is a unique accepting computation of T on x that outputs y.

Proposition 6.1 *$NP = \mathcal{U}$ if and only if* $\mathrm{NPSV} = \mathcal{U}\mathrm{SV}$.

Proof. The proof from right to left is obvious. For the proof from left to right, let $f \in \mathrm{NPSV}$ and note that $\mathrm{graph}(f) = \{\langle x, y\rangle \mid y = f(x)\}$ belongs to NP. Assuming $NP = \mathcal{U}$, $\mathrm{graph}(f) \in \mathcal{U}$. A unique nondeterministic computation of $f(x)$ from $\mathrm{graph}(f)$ is trivially obtained. We conclude that $f \in \mathcal{U}\mathrm{SV}$. ∎

PROPOSITION 6.2. *$P = \mathcal{U} \cap co\text{-}\mathcal{U}$ if and only if $\mathcal{U}\mathrm{SV}_t \subseteq P_t$.*

The proof of this proposition is nearly identical to the proof of Proposition 2.4.

THEOREM 6.3. *$\mathcal{U} = NP$ if and only if for every set D, $\mathcal{U}(D) = \mathrm{NPCM}(D)$.*

Proof. We sketch the proof in the nontrivial direction. The properties confluent and mature guarantee that the function FIND_M belongs to NPSV (cf. Theorem 3.2 and Lemma 3.3). Therefore, by Proposition 6.1, $\mathrm{FIND}_M \in \mathcal{U}\mathrm{SV}$. The procedure given in the proof of Theorem 3.2 now applies in order to complete the argument. ∎

THEOREM 6.4. *$P = \mathcal{U} \cap co\text{-}\mathcal{U}$ if and only if for every set D, $P(D) = \mathcal{U}\mathrm{CMS}(D)$.*

Proof. This time FIND_M belongs to $\mathcal{U}\mathrm{SV}_t$ and so, by Proposition 6.2, $\mathrm{FIND}_M \in \mathrm{PF}_t$. Therefore, the proof of Theorem 3.2 applies directly. ∎

It is possible to give a positive relativization of the "$P = \mathcal{U}$" question also, but the restrictions needed are less natural. Its development is left, therefore, to the interested reader.

REFERENCES

1. T. BAKER, J. GILL, AND R. SOLOVAY, Relativizations of the $P = ?$ NP question, *SIAM J. Comput.* **4** (1975), 431–442.
2. R. BOOK, Bounded query machines: On NP and PSPACE, *Theoret. Comput. Sci.* **15** (1981), 27–39.
3. R. BOOK, T. LONG, AND A. SELMAN, Quantitative relativizations of complexity classes, *SIAM J. Comput.* **13** (1984), 461–487.
4. R. BOOK AND C. WRATHALL, Bounded query machines: On NP( ) and NPQUERY( ), *Theoret. Comput. Sci.* **15** (1981), 41–50.
5. J. GESKE AND J. GROLLMANN, Relativizations of unambiguous and random polynomial time classes, *SIAM J. Comput.*, to appear.
6. R. LADNER, On the structure of polynomial time complexity, *J. Assoc. Comput. Mach.* **22** (1975), 155–171.
7. T. LONG, Strong nondeterministic polynomial-time reducibilities, *Theoret. Comput. Sci.* **21** (1982), 1–25.
8. C. RACKOFF, Relativized questions involving probabilistic algorithms, *J. Assoc. Comput. Mach.* **29** (1982), 261–268.
9. D. RUSSO AND S. ZACHOS, Positive relativizations of probabilistic polynomial-time complexity classes, manuscript, 1984.
10. U. SCHÖNING, A uniform approach to obtain diagonal sets in complexity classes, *Theoret. Comput. Sci.* **18** (1982), 95–103.
11. A. SELMAN, XU MEI-RUI, AND R. BOOK, Positive relativizations of complexity classes, *SIAM J. Comput.* **12** (1983), 565–579.
12. L. VALIANT, Relative complexity of checking and evaluating, *Inform. Process. Lett.* **5** (1976), 20–23.