SIAM J. COMPUT.
Vol. 9, No. 1, February 1980

0097-5397/80/0901-0003 $01.00/0

REFINING NONDETERMINISM IN RELATIVIZED POLYNOMIAL-TIME BOUNDED COMPUTATIONS*

CHANDRA M. R. KINTALA† AND PATRICK C. FISCHER‡

Abstract. Let $\mathcal{P}_{g(n)}$ denote the class of languages acceptable by polynomial-time bounded Turing machines making at most $g(n)$ nondeterministic moves on inputs of length n. For any constructible $g(n)$, $\mathcal{P} \subseteq \mathcal{P}_{g(n)} \subseteq \mathcal{NP}$. The classes $\mathcal{P}_{g(n)}$, for various $g(n)$ of the form $(\log n)^k$, $k \geqq 1$, are relativized and the relationships among those relativized classes are studied. In particular, oracle sets are constructed which (1) make all the relativized classes equal (this follows from Baker, Gill and Solovay (1975)); (2) make all the classes associated with powers of log n different; (3) for any k, make all the classes below $(\log n)^k$ different while the kth power class is equal to relativized $\mathcal{NP}$. Results regarding closure of the $(\log n)^k$ classes under complementation are also given.

1. Introduction. The fundamental open question posed by Cook [2] and Karp [4] asks whether $\mathcal{P}$ equals $\mathcal{NP}$. Here $\mathcal{P}$ is the class of languages recognized in polynomial time by deterministic Turing machines, and $\mathcal{NP}$ is the class of languages accepted in polynomial time by nondeterministic Turing machines. A language L is said to be $\mathcal{NP}$-complete if L is in $\mathcal{NP}$ and every language in $\mathcal{NP}$ is polynomial-time reducible to L. A wide variety of problems (currently estimated at 2,000) have been shown to be $\mathcal{NP}$-complete. Inherent in such studies is the general conviction that any $\mathcal{NP}$-complete language has the difficulty of the whole class $\mathcal{NP}$ embedded in it and hence it acts as a "representative" of the class $\mathcal{NP}$.

For many $\mathcal{NP}$-complete languages, if n is the length of the input, a nondeterministic algorithm exists requiring a total number of moves which is a polynomial in n, but the number of nondeterministic moves is at most linear in n. (By a nondeterministic move, we mean a "strict" nondeterministic move where there are at least two choices for the next step of the machine.) As a matter of fact, one can easily show that the $v/3$-clique problem (given a graph G with v vertices and e edges, does G have a clique of size $v/3$?) is $\mathcal{NP}$-complete and that there is a nondeterministic polynomial algorithm for this problem making at most $O(\sqrt{n})$ nondeterministic moves on inputs of length n, given any "standard" representation of the graphs as strings. On the other hand, the nondeterministic polynomial-time algorithm for accepting primes described by Pratt [7] needs $\Omega(n^2)$ nondeterministic moves even though the recognition of primes has not been shown to be $\mathcal{NP}$-complete.

These observations on the nondeterministic-step requirements of the candidates for the languages in $\mathcal{NP}$-$\mathcal{P}$ suggest that we study the class $\mathcal{NP}$ from the viewpoint of restricted nondeterminism.

DEFINITION 1.1. For any function $g(n) \geqq 0$, let

$\mathcal{P}_{g(n)} = \{L \mid L \subseteq \{0, 1\}^*$ and there is a constant c such that L is accepted by a polynomial-time bounded Turing machine making at most $g(n)$ c-ary nondeterministic moves$\}$.

Clearly, $\mathcal{NP} = \bigcup_{k=0}^{\infty} \mathcal{P}_{n^k}$. Also,

$$\mathcal{P}_0 = \mathcal{P}_{\log n} \subseteq \mathcal{P}_{(\log n)^2} \subseteq \cdots \subseteq \mathcal{P}_{(\log n)^k} \subseteq \cdots \subseteq \mathcal{P}_n \subseteq \mathcal{NP}.$$

(The first equality comes from the fact that log n nondeterministic moves of a Turing machine can be simulated deterministically by following at most $c \cdot \log n$ branches, i.e. a

* Received by the editors June 28, 1978, and in revised form January 30, 1979.

† Computer Science Department, University of Southern California, Los Angeles, California 90007.

‡ Department of Computer Science, Pennsylvania State University, University Park, Pennsylvania 16802.

polynomial number of branches.)[1] Refining the original question $\mathcal{P} = ? \mathcal{NP}$, we can ask whether $\mathcal{P}_{(\log n)k} = ? \mathcal{P}_{(\log n)k+1}$ for any $k \geqq 1$. We have not been able to answer the latter question for any k and suspect that these questions will be difficult to solve. To support this we can relativize the question analogously to the work of Baker, Gill, and Solovay [1], and show that for every $k \geqq 1$, the corresponding relativized questions have affirmative answers for some oracles but negative answers for other oracles. We also construct oracles for each k such that the relativized version of the above hierarchy is distinct up to the kth level but collapses from the kth level onwards. The behavior of these relativized classes under the operation of complementation is also investigated. The results we obtain indicate that the power of even "small" amounts of nondeterminism in polynomial-bounded machines is not amenable to the traditional methods of analysis. On the other hand, in Kintala [5] related questions in some other familiar classes of computations have turned out to be tractable.

Note. Some readers may prefer to allow $O(g(n))$ binary nondeterministic moves instead of $g(n)$ c-ary nondeterministic moves in Definition 1.1. All our results in this paper will carry over with this definition also. We have, however, chosen the above definition because (i) it appears unnatural to us to restrict the machines to just two choices at any step and (ii) the proofs would not be simplified if the latter definition were used since the "fan-out" constant c would be replaced by a constant c' to be applied to $g(n)$.

Some of the results in this paper are contained in Chapter 3 of [5]. Related results were first presented in [6]. Many of the proof methods used here are analogous to those of Baker, Gill and Solovay [1].

2. The model and the definitions. The model for computations in this paper is the *query machine* which is a nondeterministic multitape Turing machine with an additional work tape 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. We will identify an oracle for a set X with the set X itself and shall deal only with recursive oracles. The *language* accepted by a query machine with oracle X is the set of input strings for which some possible computation of the machine halts in one of the designated *accepting* states. Henceforth, we fix $\{0, 1\}$ as the alphabet in which all the languages are encoded. A query machine is *polynomial-time 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.

A c-ary nondeterministic move of a machine is a move in which the number of choices for the next step of the machine is c. Such a nondeterministic move is sometimes said to have a *fan-out* of c. Any query machine M can be so designed that all the nondeterministic moves made by M have the same fan-out c for some constant c, which depends on M. The relativized version of Definition 1.1 can now be provided.

DEFINITION 2.1. For a given function $g(n)$ and any oracle X, let $\mathcal{P}^X_{g(n)} = \{L | L \subseteq \{0. 1\}^*$ and there is a constant c such that L is accepted by a polynomial-time bounded query machine with oracle X making at most $g(n)$ c-ary nondeterministic moves$\}$. We also define $\mathcal{NP}^X = \bigcup_{k=0}^{\infty} \mathcal{P}^X_{n^k}$.

[1] $\log n$ = the largest integer m such that $2^m \leqq n$.

A recursive enumeration of all nondeterministic polynomial-time bounded query machines can be obtained by listing all pairs $\langle Q, p\rangle$ where Q is a query machine and p is a polynomial and attaching a $p(n)$-clock to Q. The new machine Q' will halt whenever its computation exceeds $p(n)$ steps. We will use such an enumeration as our "standard enumeration" and let M_i^X denote the ith machine in this enumeration.

For each i, we can effectively determine a c_i and an a_i such that c_i is the fan-out of the nondeterministic moves made by M_i^X and $p_i(n) = a_i + n^{a_i}$ is a strict upper bound on the length of any computation by M_i^X, for any oracle X.

Given a time-constructible function $g(n)$ and an oracle X, we let $M_{i,g(n)}^X$ denote the query machine which results by attaching a $g(n)$-time clock to M_i^X.[2] This clock stops M_i^X if $g(n)$ nondeterministic moves are exceeded. We observe that c_i bounds the fan-out of $M_{i,g(n)}^X$ and that $p_i(n)$ still bounds its run time since the $g(n)$-time counter runs in parallel with the main computation of M_i^X and makes no nondeterministic moves of its own.

For the readability of our statements, we introduce the following additional terminology:

DEFINITION 2.2. Let

(1) $ML_{i,k}^X \triangleq M_{i,(\log n)^k}^X$;

(2) $\mathcal{PL}_k^X \triangleq \mathcal{P}_{(\log n)^k}^X$;

(3) $\mathcal{PL}^X \triangleq \bigcup_{k=1}^{\infty} \mathcal{PL}_k^X$.

For any oracle X,

$$\mathcal{P}^X = \mathcal{PL}_1^X \subseteq \mathcal{PL}_2^X \subseteq \cdots \subseteq \mathcal{PL}_k^X \subseteq \cdots \mathcal{PL}^X \subseteq \mathcal{P}_n^X \subseteq \mathcal{NP}^X.$$

We call the preceding hierarchy as the "relativized $\mathcal{PL}$-hierarchy."

A language A is said to be *polynomial-time reducible* to B if there is a function f computable by a deterministic Turing machine in time bounded by some polynomial, such that $x \in A$ if and only if $f(x) \in B$. (This definition is that of Karp [4].) For a given class $\mathcal{C}$ of languages, a language B is said to be *$\mathcal{C}$-complete if* $B \in \mathcal{C}$ and A is a polynomial-time reducible to B for every $A \in \mathcal{C}$. As mentioned in the introduction, many $\mathcal{NP}$-complete languages are in $\mathcal{P}_n$. We will exhibit $\mathcal{PL}_k$-complete languages for all $k \geqq 1$.

3. $\mathcal{PL} = ?\mathcal{PL}_{k+1}$ relativized. Baker, Gill, and Solovay [1] have constructed an oracle A such that $\mathcal{P}^A = \mathcal{NP}^A$. It is obvious that for any such A, $\mathcal{PL}_k^A = \mathcal{PL}_{k+1}^A$ for every $k \geqq 1$. We will construct here oracles for which these classes differ.

DEFINITION 3.1. For every $k \geqq 1$ and all oracles X, let

$$L_k(X) = \{w \mid (\exists y)[|y| = (\log(|w|))^k \text{ and } wy \in X]\}.$$

Clearly, $L_k(X) \in \mathcal{PL}_k^X$, since one can nondeterministically guess y, then check whether $wy \in X$.

THEOREM 3.2. *For every $k \geqq 2$, there is an oracle B_k such that $\mathcal{PL}_{k-1}^{B_k} \subsetneq \mathcal{PL}_k^{B_k}$.*

Proof. We shall define the required oracle B_k in such a way that $L_k(B_k) \notin \mathcal{PL}_{k-1}^{B_k}$. Since $L_k(B_k) \in \mathcal{PL}_k^{B_k}$, the theorem will follow immediately. B_k will be constructed in $wy \in X$.

[2] A function $g(n)$ is said to be *time-constructible* if there is a *real-time*-bounded deterministic Turing machine which, given inputs of length n, will produce outputs of length exactly $g(n)$. Functions of the form $(\log n)^k$, $\log^k(n)$, n, n^k, etc., are all time-constructible. See [5], [6] for related discussions.

stages. We denote by $B_k(i)$ the finite set of strings placed into B_k prior to stage i. Let $B_k(0) = \varnothing$, $n_{-1} = 0$ and start at stage 0.

Stage i. Choose a sufficiently large n_i so large that

(i) $n_i > n_{i-1}$,

(ii) $n_i + (\log n_i)^k > p_{i-1}(n_{i-1})$,

(iii) $c_i^{(\log n_i)^{k-1}} p_i(n_i) < 2^{(\log n_i)^k}$

where p_i and c_i are as described in Definition 2.2. The first two requirements can obviously be satisfied, and one can note that $2^{(\log n_i)^k} = (2^{\log n_i})^{(\log n_i)^{k-1}}$. Then a suitable n_i can always be found.

Simulate the machine $M = ML_{i,k-1}^{B_k(i)}$ on input $x = 0^{n_i}$. If M accepts x in any of its possible computations, then place no string into B_k at this stage. If M rejects x, then add some string of the form xy such that $|y| = (\log |x|)^k$ and xy is not queried during any possible computation of M on x. Such a string exists because there are at most $c_i^{(\log n_i)^{k-1}}$ possible computations, each of length at most $p_i(n_i)$. Therefore, the number of queries the machine could have made in all its possible computations is $c_i^{(\log n_i)^{k-1}} \cdot p_i(n_i) < 2^{(\log n_i)^k}$ by condition (iii). So some such xy is available. Set $B_k(i+1) = B_k(i) \cup \{xy\}$ and go to stage $i+1$.

The computation of $ML_{i,k-1}^{B_k}$ on input $x = 0^{n_i}$ is the same whether B_k or $B_k(i)$ is used as an oracle because conditions (i) and (ii) guarantee that no string queried by $ML_{i,k-1}^{B_k}$ in any computation of x is later added to B_k (strings are obviously never deleted from B_k). This is true because no string queried is longer than the run-time bound $p_i(n_i)$, while strings added, if any, after stage i are of length at least $n_{i+1} + (\log n_{i+1})^k > p_i(n_i)$. At stage i, we ensure that $ML_{i,k-1}^{B_k}$ does not recognize $L_k(B_k)$, since, by construction, $ML_{i,k-1}^{B_k}$ in any computation of x is later added to B_k (strings are obviously never deleted $|y| = (\log (|x|))^k$, that is, if and only if $x \in L_k(B_k)$. Hence $L_k(B_k) \notin \mathcal{PL}_{k-1}^{B_k}$. □

We can modify the above construction to obtain a uniform oracle B for all k by using the familiar dovetailing methods. We omit details.

THEOREM 3.3. *There is an oracle B such that $\mathcal{PL}_{k-1}^B \subsetneq \mathcal{PL}_k^B$ for every $k \geq 2$, i.e.,*

$$\mathcal{P}^B = \mathcal{PL}_1^B \subsetneq \mathcal{PL}_2^B \subsetneq \cdots \subsetneq \mathcal{PL}_k^B \subsetneq \cdots \subsetneq \mathcal{PL}^B.$$

The above results can be interpreted in the following manner. The question $\mathcal{P} = ? \mathcal{NP}$ asks whether providing "full" nondeterminism to a class of polynomial-time bounded machines increases the class of languages accepted by them. Baker, Gill, and Solovay [1] have shown that the relativized $\mathcal{P} = ? \mathcal{NP}$ question has an affirmative answer for some oracles but a negative answer for other oracles, which is taken to be further evidence of the difficulty of the $\mathcal{P} = ? \mathcal{NP}$ question. By the same token, our observations indicate that the refined questions as defined in the Introduction are also difficult in the sense that neither familiar diagonalization methods nor simulation methods are applicable to resolve the original or the refined questions of $\mathcal{P} = ? \mathcal{NP}$. The preceding arguments do not however discount the possible use of information counting methods, such as those initiated by Hartmanis and Stearns [3]. We have noted in Kintala [5] and Kintala and Fischer [6] that even those methods are probably not applicable.

An interesting property of the relativized $\mathcal{PL}$-hierarchy is exhibited by the following theorem which shows that problems concerning this hierarchy do not automatically reduce to equivalent problems in $\mathcal{P} = ? \mathcal{NP}$.

THEOREM 3.4. *For every $k \geq 2$, there is an oracle D_k such that*

$$\mathcal{P}^{D_k} = \mathcal{PL}_1^{D_k} \subsetneq \mathcal{PL}_2^{D_k} \subsetneq \cdots \subsetneq \mathcal{PL}_k^{D_k} = \cdots = \mathcal{PL}^{D_k} = \mathcal{NP}^{D_k}.$$

Proof. For any given m, and any oracle X, define

$$\hat{L}_m(X) = \{w \mid (\exists y)[|y| = (\log(|w|))^m;\ |wy| \text{ is even};\ wy \in X]\}.$$

Note that this definition adds only the restriction "$|wy|$ is even" to the definition for $L_k(X)$. Thus, it is immediate that $\hat{L}_m(X) \in \mathcal{P}\mathcal{L}_m^X$. We shall construct here the required oracle D_k in such a way that:

(1) For all d and for any given x such that $|x| \geqq d$ and $p_d(|x|) < |x|^{(\log|x|)^{1/k}}$, $M_d^{D_k}$ accepts x is and only if for $y = 0^d 1x10^t$, such that $|y| = p_d(|x|) + 1$, $(\exists w)[|w| = (\log|x|)^k + \varepsilon$, $\varepsilon = 0$ or 1, $|yw|$ is odd, and $yw \in D_k]$.

(2) $\hat{L}_m(D_k) \notin \mathcal{P}\mathcal{L}_{m-1}^{D_k}$ for every m such that $2 \leqq m \leqq k$.

Requirement (1) will guarantee $\mathcal{P}\mathcal{L}_k^{D_k} = \mathcal{N}\mathcal{P}^{D_k}$ as follows: Given any language $L \in \mathcal{N}\mathcal{P}^{D_k}$, there is an index d such that L is accepted by $M_d^{D_k}$. We can then construct a machine $M = ML_{j,k}^{D_k}$ for some j which, given x, constructs $y = 0^d 1x10^t$ if $|x| \geqq d$ and $2|x| + 2 < p_d(|x|) < |x|^{(\log|x|)^{1/k}}$ so that $|y| = p_d(|x|) + 1$. This is possible except perhaps for a finite number of x. M then guesses a string w such that $|w| = (\log|x|)^k + \varepsilon$, where $\varepsilon = 0$ or 1, so that $|yw|$ is odd. M then queries D_k and accepts x if and only if $yw \in D_k$. If w is such that the above conditions on $|x|$ are not satisfied, then M accepts x if and only if it is in a finite table stored in M. Thus, information about the acceptance of strings in $\mathcal{N}\mathcal{P}^{D_k}$ is encoded into strings in D_k of odd lengths.

Requirement (2) is satisfied by "attacking" the classes $\mathcal{P}\mathcal{L}_1^{D_k}, \cdots, \mathcal{P}\mathcal{L}_{k-1}^{D_k}$ in a cyclic manner through some strings in D_k of even length. This will imply $\mathcal{P}\mathcal{L}_{m-1}^{D_k} \subsetneq \mathcal{P}\mathcal{L}_m^{D_k}$ for $2 \leqq m \leqq k$.

We will say that a string $y = 0^d 1x10^t$ is *admissible* if $|x| \geqq d$ and $|y| \leqq x|^{(\log|x|)1/k}$. As before, D_k will be constructed in stages. Let $D_k(i)$ denote the set of strings added to D_k prior to stage i. During the course of construction, some strings will be *reserved* for $\bar{D}_k$. An index $e = \langle j, m\rangle$ (for some canonical enumeration of the pairs of the form $\langle j, m\rangle$ such that $j \geqq 0$ and $2 \leqq m \leqq k$) will be *canceled* at some stage n_e when we ensure that $ML_{j,m-1}^{D_k}$ does not accept $\hat{L}_m(D_k)$. Set $D_k(0) = \varnothing$, $n_{-1} = 0$, and start at stage 0.

Stage i: Execute the following two routines.

Routine A (Towards requirement (1)). For every string y of length i, if $y = 0^d 1x10^t$ is admissible, simulate all the computations of $M_d^{D_k(i)}$ on input x for up to $i-1$ steps. In any such computation only strings of length less than i are queried.

For each such y and the associated d and x, if any computation of $M_d^{D_k(1)}$ accepts x, then place some string of the form yw into D_k where

(i) $w \in \{0, 1\}^*$; $|w| = (\log|x|)^k + \varepsilon$ where $\varepsilon = 0$ or 1 so that $|yw|$ is odd and

(ii) yw has not been reserved for $\bar{D}_k$ in an earlier stage.

(We will ensure in condition (iv) of the following routine, which is the only routine reserving strings for $\bar{D}_k$, that such w is available, if needed. It is of course possible that no strings will be added to D_k at this stage by Routine A.)

Routine B (Towards requirement (2)). Let $e = \langle j, m\rangle$ be the least uncanceled index. If i is such that $i + (\log i)^m$ is odd, or if any of the following four conditions is not satisfied, then skip this routine and go to stage $i+1$. Otherwise, cancel e at this stage and choose $n_e = i$ as follows: suppose $e - 1 = \langle g, h\rangle$ for some $g \geqq 0$ and $2 \leqq h \leqq k$.

(i) $i > p_g(n_{e-1})$;

(ii) no string of length $\geqq i$ is reserved for $\bar{D}_k$,

(iii) $c_j^{(\log i)^{m-1}} \cdot p_j(i) < 2^{(\log i)^m}$ (Recall Definition 2.2);

(iv) there is no admissible $y = 0^d 1x10^t$ with $|y| \geqq i$ such that there is an $\varepsilon = 0$ or 1 so that $ML_{j,m-1}^{D_k(i)}$, in all its possible computations on input 0^i, queries *all* strings of the form yw where $w \in \{0, 1\}^*$ and $|w| = (\log|x|)^k + \varepsilon$.

Simulate the machine $M = ML_{j,m-1}^{D'}$, where $D' = D_k(i) \cup$ {the odd length strings you

just added by Routine A in this stage}, on input $z = 0^i$. In this simulation, reserve for $\bar{D}_k$ all strings of length at least i queried during any possible computation of M on z and which are not members of D'. If M accepts z then add no element to D_k in this routine at this stage But, if M rejects z, then add to D_k some string of the form zv such that $|v| = (\log i)^m$ and zv is not queried during any possible computation of M on z. Such a string exists because of conditions (i) and (iii) of this routine. Cancel the index e and go to stage $i+1$.

End of stage i.

It is easy to see that for a given $e = \langle j, m\rangle$, conditions (i), (ii) and (iii) of Routine B will be satisfied for large enough i. Condition (iv) will also be satisfied because of the following:

For large enough i and for any admissible $y = 0^d 1x10^t$ such that

$$i \leqq |y| \leqq |x|^{(\log|x|)^{1/k}} = 2^{(\log|x|)^{(k+1)/k}})$$

$ML^{D_k(i)}_{j,m-1}$, will reserve at most

$$\begin{aligned} c^{(\log i)^{m-1}} \cdot p_j(i) &\leqq c^{(\log(2^{(\log|x|)^{(k+1)/k}}))^{m-1}} \cdot p_j(2^{(\log|x|)^{(k+1)/k}}) \\ &\leqq c_j^{(\log|x|)^{(k+1)(m-1)/k}} \cdot p_j(2^{(\log|x|)^{(k+1)/k}}) \\ &< 2^{(\log|x|)^k} \quad \text{for large enough x} \end{aligned}$$

since $(k+1)(m-1)/k < k$ if $2 \leqq m \leqq k$ and $k \geqq 2$.

Hence, every index e is eventually canceled, thus satisfying requirement (2). It is straightforward to see that Routine A works for requirement (1).

Our construction given above is based on Robertson's [8] observations that $(\exists X)[\mathcal{P}^X \subsetneq \mathcal{P}^X_n = \mathcal{NP}^X]$. His proof, in turn, is similar to that of Theorem 4.5 in the next section.

4. Closure under complementation. We do not know whether $\mathcal{PL}_k$ is closed under complementation for any $k \geqq 2$. However in the relativized case we can exhibit oracle sets for each side of the question.

THEOREM 4.1. *For each $k \geqq 2$, there is an oracle E_k such that $\mathcal{PL}^{E_k}_k$ is not closed under complementation.*

Proof. The proof is by a construction very similar to that of Theorem 3.2. □

It is obvious that if $\mathcal{P}^A = \mathcal{NP}^A$, then $\mathcal{PL}^A_k$ is closed under complementation for all k, since $\mathcal{P}^X$ is closed under complementation for all oracles X. Thus, the closure of a particular $\mathcal{PL}^X_k$ under complementation depends on the oracle X. However, we can exhibit an oracle F_k such that $\mathcal{PL}^{F_k}_k$ *is closed* under complementation but $\mathcal{P}^{F_k} \neq \mathcal{PL}^{F_k}_k$.

LEMMA 4.2 (Constant speed-up). *For any time-constructible $g(n)$ and any oracle X, $\mathcal{P}^X_{g(n)} = \mathcal{P}^X_{g(n)/2}$.*

Proof. Clearly, $\mathcal{P}^X_{g(n)/2} \subseteq \mathcal{P}^X_{g(n)}$. Suppose $L \in \mathcal{P}^X_{g(n)}$. Let $M = M^X_{i,g(n)}$ be a machine accepting L and making $g(n)$ nondeterministic moves of fan-out $c = c_i$. Then construct a new machine M' which initially makes $g(n)/2\,c^2$-ary nondeterministic moves to write a string of length $g(n)/2$ on a special guess tape using a special alphabet $\{\gamma_{ij} | 1 \leqq i \leqq c;\ 1 \leqq j \leqq c\}$. Then M' simulates M; whenever M needs to make a nondeterministic move, the nondeterminism in coded form is available on this guess tape with two c-ary nondeterministic moves available on each square of this tape. Thus $L \in \mathcal{P}^X_{g(n)/2}$. □

We shall use the following languages in the construction of F_k. Observe that if a language A is $\mathscr{C}$-complete for a class $\mathscr{C}$, then $\bar{A}$ is (co-$\mathscr{C}$)-complete where co-$\mathscr{C}$ is the class of complements of the languages in $\mathscr{C}$.

DEFINITION 4.3. For every $k \geqq 1$, and any oracle X, let $A_k(X) = \{0^i 1x10^t |$ some computation of $ML^X_{i,k}$ accepts x in no more than t steps$\}$.

LEMMA 4.4. *$A_k(X)$ is $\mathcal{PL}_k^X$-complete. Moreover, $\mathcal{PL}_k^X$ is closed under complementation if and only if $\overline{A_k(X)} \in \mathcal{PL}_k^X$.*

Proof. Clearly, $A_k(X) \in \mathcal{PL}_k^X$. Suppose $L \in \mathcal{PL}_k^X$, say L is accepted by $ML_{d,k}^X$ for some d. Set $f(x) = 0^d 1x 10^{p_d(|x|)}$. Then $f(x)$ is computable in polynomial time. Now $x \in L$ if and only if $ML_{d,k}^X$ accepts x in no more than $p_d(|x|)$ steps making no more than $(\log |x|)^k$ nondeterministic moves. So $x \in L$ if and only if $f(x) \in A_k(X)$. Hence $A_k(X)$ is $\mathcal{PL}_k^X$-complete.

If $\mathcal{PL}_k^X$ is closed under complementation then obviously $\overline{A_k(X)} \in \mathcal{PL}_k^X$. Conversely, suppose $\overline{A_k(X)} \in \mathcal{PL}_k^X$. Let $\overline{A_k(X)}$ be accepted by a machine $M = ML_{j,k}^X$ for some j. For any $L \in \mathcal{PL}_k^X$, since L is polynomial-time reducible to $A_k(X)$, there exists a function f such that $x \in \bar{L}$ if and only if $f(x) \in \overline{A_k(X)}$ and $|f(x)| \leqq |x|^b$ for some constant b. Construct a machine N to accept $\bar{L}$ in the following manner. Given a string x, N transforms x to $f(x)$ and then simulates M on $f(x)$. Obviously the number of nondeterministic moves made by N is bounded by $(\log |f(x)|)^k \leqq b^k \cdot (\log |x|)^k$. Invoking Lemma 4.2, we can now infer that $\bar{L} \in \mathcal{PL}_k^X$. □

THEOREM 4.5. *For every $k \geqq 2$, there is an oracle F_k such that*

(1) *$\mathcal{PL}_k^{F_k}$ is closed under complementation and*

(2) *$\mathcal{P}^{F_k} \subsetneq \mathcal{PL}_k^{F_k}$.*

Proof. For the given k and any oracle X, define

$$L_k'(X) = \{w \mid |w| = 2^{m+1} \text{ for some } m \geqq 0;\ (\exists y)[|y| = m^k;\ wy \in X]\}.$$

Clearly, $L_k'(X) \in \mathcal{PL}_k^X$. F_k will be constructed in such a way that

(1) for any string u, $u \in \overline{A_k(F_k)}$ if and only if $(\exists v)[|v| = (\log |u|)^k$ *and* $uv \in F_k]$

(2) $L_k'(F_k) \notin \mathcal{P}^{F_k}$.

Then $\overline{A_k(F_k)} \in \mathcal{PL}_k^{F_k}$ from (1), so that by Lemma 4.4 $\mathcal{PL}_k^{F_k}$ will be closed under complementation; and since $L_k'(F_k) \in \mathcal{PL}_k^{F_k}$ we can infer from (2) that $\mathcal{P}^{F_k} \neq \mathcal{PL}_k^{F_k}$.

As usual, F_k will be constructed in stages. At stage i, we decide the membership in F_k of all strings of length i. In the course of construction, some strings will be reserved for $\bar{F}_k$, that is, designated as nonmembers of F_k. An index e will be *canceled* at some stage when we ensure that $M_{e,0}^{F_k}$ (i.e., the eth machine of the class of deterministic polynomial-bounded query machines) does not recognize $L_k'(F_k)$. Let $F_k(i)$ denote the set of strings placed into F_k prior to stage i; $n_{-1} = 0$, $F_k(0) = \varnothing$; and start at stage 0.

Stage i: If $i = 2^m + m^k + j$ for some m, j such that $0 \leqq j < 2^m$, then go to Routine A. If $i = 2^{m+1} + m^k$ for some $m \geqq 0$ then go to Routine B. Else go to Routine C. Observe that for every integer $n \geqq 1$, $n = 2^m + j$ for m and j such that $0 \leqq j < 2^m$. The following "integer line," for sufficiently large m, might be helpful to the reader in determining which routines are executed at various stages of the construction of F_k.

$$\underset{2^m + m^k}{\text{Routine C}}\Big[\underset{2^{m+1} + m^k}{\text{Routine A}}\Big]\Big[\underset{2^{m+1} + (m+1)^k}{\text{Routine C}}\Big]\Big[\underset{}{\text{Routine A}}\Big]$$

Routine A. Here $i = 2^m + m^k + j$ for some $m \geqq 0$ and $0 \leqq j < 2^m$.

Notice that $\log (2^m + j) = m$. For every string z of length i, not reserved for $\bar{F}_k$ at an earlier stage, determine the prefix u of z having length $2^m + j$. If $u = 0^d 1x 10^t$ then place z into F_k if and only if $ML_{d,k}^{F_k(i)}$ does not accept x in fewer than t steps making no more than $(\log |x|)^k$ nondeterministic moves. If u is not of the above form, then place z into F_k. Go to Stage $i+1$.

Routine B. Here $i = 2^{m+1} + m^k$ for some $m \geqq 0$.

Let e be the least uncanceled index. Choose $n_e = 2^{m+1}$ if and only if the following conditions are satisfied:

(i) No string of length $\geqq i$ is reserved for $\bar{F}_k$;

(ii) $n_e > p_{e-1}(n_{e-1})$;

(iii) $p_e(n_e) < 2^{(\log(n_e)-1)^k}$.

If 2^{m+1} does not satisfy the above conditions then go to Routine C. Otherwise, simulate the machine $M = M_{e,0}^{F_k(i)}$ on input $x = 0^{n_e}$ and reserve for $\bar{F}_k$ all strings of length $\geqq i$ queried during the computation of M on x. If M accepts x then add no element to F_k at this stage. But if M rejects x then add to F_k some string of the form xv such that $|v| = m^k$. Thus $|xv| = i$. Such a string exists because M is deterministic and hence could have queried at most $p_e(n_e) < 2^{(\log(n_e)-1)^k} = 2^{m^k}$ strings in its computation on x. Cancel index e and go to stage $i+1$.

Routine C. Add no element to F_k at this stage and go to stage $i+1$.

End Stage i.

Every index e is eventually canceled because for a given e and some sufficiently large m conditions (i), (ii), (iii) will be satisfied after index $(e-1)$ is canceled. When index e is canceled at stage i, our construction guarantees that $M_{e,0}^{F_k}$ does not recognize $L'_k(F_k)$, satisfying requirement (2).

At any stage i of the form $(2^{m+1}+m^k)$, fewer than 2^{m^k} strings are reserved for $\bar{F}_k$. Thus, for any j such that $0 \leqq j < 2^m$, fewer than $2^{0^k}+2^{1^k}+\cdots+2^{(m-1)^k} < 2^{m^k}$ strings of length 2^m+m^k+j are reserved for $\bar{F}_k$ before stages of the form $i = 2^m+m^k+j$. Therefore, every string u of length 2^m+j is the prefix of at least one string z of length $(2^m+j)+m^k$ which is never reserved for $\bar{F}_k$. Moreover the computations in Routine A when processing $u = 0^d 1 x 1 0^t$ at stage 2^m+j+m^k never query strings of length greater than 2^m+j. The memberships of the strings of length not exceeding 2^m+j in F_k are determined at earlier stages. Hence, by construction, any string u of length 2^m+j is in $\overline{A_k(F_k)}$ if and only if u is the prefix of a string of length 2^m+j+m^k in F_k. Therefore, $\overline{A_k(F_k)} \in \mathcal{PL}_k^{F_k}$, satisfying requirement (1). □

Acknowledgment. The authors would like to thank Professors Edward Robertson and Joel Seiferas for many helpful discussions. They are indebted to Professor Seiferas for pointing out a serious mistake in an earlier version of Theorem 3.4. They are also grateful to the referees for their careful reading of the paper and their helpful comments and suggestions.

REFERENCES

[1] I. BAKER, J. GILL AND R. SOLOVAY, *Relativization of the P = ? NP question*, SIAM J. Comput., 4 (1975), pp. 431–442.

[2] S. COOK, *The complexity of theorem proving procedures*, Proc. 3rd Annual Symp. on Theory of Computing (1971), pp. 151–158.

[3] J. HARTMANIS AND R. E. STEARNS, *On the computational complexity of algorithms*, Trans. Amer. Math. Soc., 117 (1965), pp. 285–306.

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

[5] C. M. R. KINTALA, *Computations with a restricted number of nondeterministic steps*, Ph.D. dissertation, Pennsylvania State University, University Park, 1977.

[6] C. M. R. KINTALA AND P. C. FISCHER, *Computations with a restricted number of nondeterministic steps*, Proc. 9th Annual Symp. on Theory of Computing (1977), pp. 178–185.

[7] V. PRATT, *Every prime has a succinct certificate*, SIAM J. Comput., 4 (1975), pp. 214–220.

[8] E. L. ROBERTSON, private communication, 1977.