Complexity Classes Without Machines: On Complete Languages for UP*

Juris Hartmanis
Lane Hemachandra

TR 86-746
April 1986

Department of Computer Science
Cornell University
Ithaca, NY 14853

* This research was supported by NSF Research Grant DCR 8301766. The second author was supported by a Hertz Foundation Fellowship. Part of this research was done during the "Complexity Year" at the Mathematical Sciences Research Institute in Berkeley. This paper will be presented at the 1986 Structure in Complexity Theory Conference.

Complexity Classes Without Machines: On Complete Languages for UP

Juris Hartmanis
Lane Hemachandra

Department of Computer Science
Cornell University
Ithaca, New York 14853

Abstract

This paper develops techniques for studying complexity classes that are not covered by known recursive enumerations of machines. Often, counting classes, probabilistic classes, and intersection classes lack such enumerations. Concentrating on the counting class UP, we show that there are relativizations for which UP^A has no complete languages and other relativizations for which $P^B \neq UP^B \neq NP^B$ and UP^B has complete languages. Among other results we show that $P \neq UP$ if and only if there exists a set S in P of Boolean formulas with at most one satisfying assignment such that $S \cap SAT$ is not in P. $P \neq UP \cap coUP$ if and only if there exists a set S in P of uniquely satisfiable Boolean formulas such that no polynomial-time machine can compute the solutions for the formulas in S. If UP has complete languages then there exists a set R in P of Boolean formulas with at most one satisfying assignment so that $SAT \cap R$ is complete for UP. Finally, we indicate the wide applicability of our techniques to counting and probabilistic classes by using them to examine the probabilistic class BPP. There is a relativized world where BPP^A has no complete languages. If BPP has complete languages then it has a complete language of the form $B \cap MAJORITY$, where $B \in P$ and $MAJORITY = \{f \mid f$ is true for at least half of all assignments$\}$ is the canonical PP-complete set.

1. Introduction

Mundane complexity classes such as P, NP, Δ_2^P, and $PSPACE$, have recursive enumerations of machines covering their languages. These enumerations give generic complete sets. In turn, the generic complete sets form a base from which other problems can be shown hard for the class.

This research was supported by NSF Research Grant DCR-8301766. The second author was supported by a Hertz Foundation Fellowship. Part of this research was done during the "Complexity Year" at the Mathematical Sciences Research Institute in Berkeley. This paper will be presented at the 1986 Structure in Complexity Theory Conference.

Recently, interest has turned to classes without obvious recursive enumerations of machines from the class. Do these classes have complete languages? If so, what form do these complete languages take? This paper develops techniques answering such questions.

The class UP consists of all NP languages that can be accepted by non-deterministic polynomial-time machines with unique accepting paths [Va]. Such languages play an important role in many applications and are of direct interest in public-key cryptography. In particular, it has been shown [GS] that there exist one-way functions (i.e., one-to-one deterministic polynomial-time functions whose inverses are not polynomial-time computable) if and only if $P \neq UP$. Similarly, there exist one-way functions whose range is in P if and only if $P \neq UP \cap coUP$.

Currently, it is not known whether UP has complete languages. The existence of complete languages for UP would yield a method of classifying UP problems; a proof of completeness for a problem would guarantee that the problem is as hard as any other UP problem. This classification program has been successful for such classes as NP and $PSPACE$.

However, it cannot succeed for UP. We exhibit two oracles A and B such that $P^B \neq UP^B \neq NP^B$ and UP^B has complete languages, and UP^A has no complete languages. Clearly, a proof that UP has no complete languages shows that $P \neq UP \neq NP$. Our oracle constructions are novel and considerably simpler and more transparent than oracle constructions tend to be. (For related work displaying oracles for which R, $NP \cap coNP$, and the Boolean closure of NP have no complete sets see, respectively [Si], [HI], and [CH].) We introduce a powerful renormalization technique that first uses PSPACE to collapse complexity classes, and then raises diagonalizations from the ashes.

One of our results shows that if UP has no complete languages, then for any sound axiomatizable formal system F there exists a language T in UP such that for no machine N_i accepting T is there a proof in F that N_i accepts with unique accepting paths. We also show that if UP has complete languages then there exists a set R in P of Boolean formulas with at most one satisfying assignment such that

$$R \cap SAT$$

is complete for UP. This result shows that if complete languages exist in UP

then they can be obtained by picking a sound axiomatizable proof system and considering the set R of Boolean formulas for which there are polynomial-time proofs that the formulas have at most one solution. If UP has complete languages, then for sufficiently strong formal systems, $R \cap SAT$ will be a complete language. Unfortunately, we do not know what formal system is powerful enough to yield a sufficiently rich set R and, even more fundamentally, whether any formal system can yield a sufficiently rich R.

Stated intuitively, the existence of complete languages for UP demands that the fact that F has at most one solution must be "*easily* provable in a *large* number of cases," so that we obtain a set R so rich that $R \cap SAT$ is UP-complete. We conjecture that this is not the case and that UP does not have complete languages.

Section 4 applies these techniques to the probabilistic complexity class BPP. For example, we show that with appropriate relativization BPP has no complete languages. Intriguingly, if BPP has complete languages then it has a complete language of the form $B \cap MAJORITY$, with B in P. Since $MAJORITY$ is a complete language for PP, we see that PP serves as the parent class of BPP in the same way that NP serves as the parent class for UP in the above results.

2. UP Languages

Let $M_1, M_2, \cdots$ and $N_1, N_2, \cdots$ be, respectively, standard enumerations of deterministic and nondeterministic polynomial-time machines with uniformly attached polynomial-time clocks. Let $T_1, T_2, \cdots$ be a standard enumeration of Turing machines.

We say that a machine N_i is *categorical* if for all inputs N_i accepts on at most one computation path. Thus for each input a categorical machine either rejects (by having no accepting paths) or accepts with exactly one accepting path.

$$UP = \{L(N_i) \mid N_i \text{ is categorical}\}.$$

For x in Σ^* let $|x|$ denote the number of symbols in x. We will denote Boolean formulas by F_i and the number of satisfying assignments by $\|F_i\|$. Thus, $SAT = \{F_i \mid \|F_i\| \geq 1\}$.

We will make considerable use of sets in P that are subsets of the set of Boolean formulas with at most one satisfying assignment. We call this class $PBF1$.

$$PBF1 = \{S \mid S \in P \text{ and } S \subseteq \{F_i \mid \|F_i\| \leq 1\}\}.$$

Theorem 1: $P \neq UP$ if and only if there exists a set S in $PBF1$ such that

$$S \cap SAT \notin P.$$

Proof: ($\Rightarrow$) Let N be categorical and $L(N) \in UP - P$. Then by Cook's theorem [Co] we know that for every x and N there corresponds an easily obtainable Boolean formula, $F_{N,x}$, such that $x \in L(N)$ if and only if $F_{N,x}$ is satisfiable. A careful inspection of Cook's proof [HU][GJ] shows that the translation is parsimonious, i.e., the number of different accepting paths of N on x is the same as the number of different satisfying assignments of $F_{N,x}$. Further, Cook's proof also shows that given a Boolean formula F, it is decidable in polynomial time whether there exists an x so F equals $F_{N,x}$. In essence, x and the machine description of N are clearly encoded in the formula $F_{N,x}$. Therefore $\{F_{N,x} \mid x \in \Sigma^*\}$ is in P and because N is categorical, $\|F_{N,x}\| \leq 1$ for $x \in \Sigma^*$. On the other hand, $\{F_{N,x} \mid x \in \Sigma^*\} \cap SAT$ is not in P, since otherwise $L(N)$ would be in P.

($\Leftarrow$) If S is in $PBF1$ and $S \cap SAT$ is not in P, then the machine N_i that deterministically determines if F is in S and then tries to guess a satisfying assignment is categorical and accepts $S \cap SAT$. Thus $S \cap SAT$ is in $UP - P$. ▸

Theorem 1 shows that $P \neq UP$ if and only if there is an easily recognizable set of formulas, each having 0 or 1 satisfying assignment, for which satisfiability testing is not in P. Now we show that $P \neq UP \cap coUP$ if and only if there is an easily recognizable set of formulas, each having exactly one satisfying assignment, for which no P machine can *find* the satisfying assignment. The "only if" direction of this result is a UP analogue of the work of Borodin and Demers [BD] on $NP \cap coNP$. Interestingly, no converse (analogous to our "if" direction) is known for the $NP \cap coNP$ case.

Theorems 1 and 2 both show that if the (co)unique acceptance model yields power beyond P, then sets with bizarre properties exist. However, we need not consider these results evidence that $P \neq UP \cap coUP$. Rather, we should view these results as reflections of the amazing power of logical formulas to describe computations— a power that spawned the theory of effective computability.

Theorem 2: $P \neq UP \bigcap coUP$ if and only if there is a set S so

1) $S \in P$ and $S \subseteq SAT$
2) $f \in S \Rightarrow f$ has exactly one solution
3) No P machine can find solutions for all formulas in S. That is,

$$g(f) = \begin{cases} 0 & f \notin S \\ \text{the unique satisfying assignment of } f & f \in S \end{cases}$$

is not a polynomial-time computable function.

Proof: ($\Rightarrow$) Let $L_0 \in (UP \bigcap coUP) - P$. Let N_0 and N_1 be categorical machines accepting, respectively, L_0 and $\bar{L}_0$.

Construct a machine N that on input x nondeterministically simulates $N_0(x)$ and $N_1(x)$. Now $L(N) = L(N_0) \bigcup L(N_1) = L_0 \bigcup \bar{L}_0 = \Sigma^*$. Since N_0 and N_1 are categorical, N has exactly one accepting path on each input. Thus, letting $F_{N,x}$ be the Cook's theorem formula for N's computation on x, $F_{N,x}$ has exactly one satisfying assignment (since Cook's reduction is parsimonious).

Let $S = \bigcup_{x \in \Sigma^*} F_{N,x}$. From the structure of Cook's reduction (as $F_{N,x}$ clearly displays N and x) S is in P. By the previous paragraph, $f \in S$ implies f has exactly one solution. Thus conditions 1 and 2 of Theorem 2 are met by S.

From the satisfying assignment to $F_{N,x}$ we can quickly determine whether $x \in L_0$ or $x \in \bar{L}_0$, by checking which path of the initial branching led to acceptance. Thus if some polynomial-time machine on input $f \in S$ output the (unique) satisfying assignment of f, then $L_0 \in P$. This contradicts our assumption that $L_0 \notin P$ and proves condition 3.

($\Leftarrow$) Let $S' = \{<f, a_1, a_2, \cdots, a_k> \mid f \in S$ and each a_i assigns some variable in f and $f(a_1, a_2, \cdots, a_k)$ is uniquely satisfiable$\}$. $f(a_1, ..., a_k)$ specifies the formula resulting from making the assignments $a_1, \cdots, a_k$ in f. For example, if $f = x_1x_2x_3$ and $a_1 =$ "x_2 is true," then $f(a_1) = x_1x_3$. $\bar{a}_1$ here would mean "x_2 is false" and $f(\bar{a}_1) =$ False.

If S' were in P, then we could use tree search to find the satisfying assignment for any formula in S, contradicting condition 3. So $S' \notin P$. It is obvious that $S' \in UP$.

To see that $S' \in coUP$, simply note that $\overline{S'} = \{<f, a_1, a_2, \cdots, a_k> \mid f \notin S$ or $[f \in S$ and $f^* = f(\overline{a}_1) \vee f(a_1, \overline{a}_2) \vee \cdots \vee f(a_1, a_2, \cdots, \overline{a}_k)$ is uniquely satisfiable$]\}$. f^* has at most one solution; it just picks up all assignments contradicting "$a_1, \cdots, a_k$." Thus $\overline{S'} \in UP$, so $S' \in coUP$. So $S' \in (UP \cap coUP) - P$. ▶

Of course, if $P = UP \cap coUP$ then Theorem 2 is of little interest. However, it is easy to diagonalize so that $P^A \neq UP^A \cap coUP^A$.

Fact 3: There is a recursive A so $P^A \neq UP^A \cap coUP^A$.

It is interesting to note that the proof technique of the previous results can be extended to characterize UP-complete languages if they exist.

Theorem 4: UP has complete languages if and only if there exists a set S in $PBF1$ such that $S \cap SAT$ is UP-complete.

Proof: ($\Rightarrow$) If $L(N)$, N categorical, is complete for UP then, as in the previous proof, $S = \{F_{N,x} \mid x \in \Sigma^*\}$ is in P. Furthermore, $S \cap SAT$ is UP-complete since $L(N)$ is reducible to $S \cap SAT$ by mapping $x \rightarrow F_{N,x}$.

($\Leftarrow$) Obvious. ▶

These results can be extended to yield necessary and sufficient conditions for the existence of UP-complete sets in terms of sets in P. For S and R in $PBF1$, S is *many-one s-reducible* to R if and only if there exists a polynomial-time function g such that

$$x \in S \cap SAT \Leftrightarrow g(x) \in R \cap SAT.$$

Corollary 5: There is a complete language in UP if and only if there exists an R_0 in $PBF1$ such that any other S in $PBF1$ is s-reducible to R_0.

Next we summarize some standard undecidability results about categorical machines to show the logical complexity of these problems. After that we observe that UP has complete languages if and only if there is a recursively enumerable list of categorical machines whose languages cover UP. (See also [Be, HI].) This result will play a major role in our diagonalization results.

Lemma 6:

a) $\{N_i \mid N_i$ is not categorical$\}$ is r.e. complete.

b) If $UP \neq NP$ then

$$\{N_i \mid L(N_i) \in UP\}$$

is Σ_2-complete in the Kleene hierarchy and

$$\{N_i \mid L(N_i) \in NP - UP\}$$

is Π_2-complete in the Kleene hierarchy.

Proof: a) Standard.

b) $\{N_i \mid L(N_i) \in NP - UP\}$ is equivalent to $\{N_i \mid L(N_i)$ is infinite$\}$ which is Π_2-complete. ▸

Lemma 7: There exists a complete language for UP if and only if there exists a recursively enumerable list of categorical machines $N_{i_1}, N_{i_2}, \cdots$, such that

$$\{L(N_{i_j}) \mid j \geq 1\} = UP.$$

Proof:(⇒) Let N_{i_0} be a categorical machine accepting a complete language in UP. Let $\{g_i\}$ be a standard enumeration of deterministic polynomial-time machines computing functions. Then, since $L(N_{i_0})$ is UP-complete, and $N_{i_0} \circ g_i$ is a categorical machine, $\{N_{i_0} \circ g_i \mid i \geq 1\}$ is a recursive enumeration of a set of categorical machines covering UP.

(⇐) Let $\{N'_{i_1}, N'_{i_2}, \cdots\}$ be a recursively enumerable set of categorical machines covering UP. Then, by padding these machines with new states that are never entered, we can obtain a set of equivalent machines $\{N_{i_1}, N_{i_2}, \cdots\}$ in P. Without loss of generality we can assume that N_{i_j} runs in time $n^{i_j} + i_j$. Then the language

$$L_U = \{N_{i_j} \# x \# 1^{|N_{i_j}|(|x|^{i_j} + i_j)} \mid N_{i_j} \text{ accepts } x\}$$

is accepted by a categorical machine that runs in polynomial time. Furthermore, it is easily seen that any other language in UP can be reduced to L_U. ▸

Thus if there are no complete languages for UP, then for any sound axiomatizable formal system F, there always will exist sets in UP for which no machine accepting them can be proven categorical in F.

3. Relativization Results

Theorem 8: There exists an oracle A such that UP^A has no complete languages.

Proof: From the previous lemma we know that UP has complete languages if and only if a polynomial-time machine accepts a set of categorical machines covering UP. Thus our goal will be to construct an oracle A such that for any M_i either M_i accepts some N_{i_j} which is not categorical or the categorical machines N_{i_j} in $L(M_i)$ do not accept the language D_i in UP [Li].

For each $i \geq 1$, let $D_i = \{1^n \mid (\exists k \geq 1)[n = p_i^k]$ and $(\exists x, y)[|x| = n$ and $x = 1y$ and $x \in A]\}$, where p_i is the i-th prime. The oracle $A = \cup_{i \geq 0} A_i$ is constructed in stages, with the help of a list I of canceled indexes.

In stage 0: $I = \emptyset$, $A_0 = \emptyset$.

In stage i, $i > 0$:

Consider the uncanceled machine N_{k_j}, $k, j \leq i$, $(k,j) \notin I$, for which $k+j$ is smallest. If no such machine exists let $I_i = I_{i-1}$ and go to stage $i+1$. Note that N_{k_j} is accepted by M_k. Consider a sufficiently long input 1^n, $n = p_k^s$, so that no oracle string of length n or longer has been queried in any previous stage.

Case 1: N_{k_j} can be made noncategorical on input 1^n by entering strings of length n in the oracle. Now M_k does not accept only categorical machines and we do not have to consider any further M_k accepted machines. Add all the M_k-accepted machines to the list I, i.e.,

$$I_i = I_{i-1} \bigcup \bigcup_{l \geq 0} \{(k,l)\},$$

freeze all oracle strings up to the longest queried string and go to stage $i+1$.

Case 2: If Case 1 does not hold, then N_{k_j} is categorical on input 1^n for all possible choices of strings of length n in the oracle. Thus for some sufficiently large n there exists a string x, $|x| = n$, such that for $A_i = A_{i-1} \bigcup \{x\}$:

$$L(N_{k_j}^{A_i}) \neq \{1^n \mid n = p_k^t, t \geq 1 \text{ and } (\exists x, y)$$
$$[|x| = n \text{ and } x = 1y \text{ and } x \in A_i]\} = D_k.$$

(The proof of this follows easily from that of Lemma 9.) Let $A_i = A_{i-1} \bigcup \{x\}$, cancel N_{k_j} (that is, add (k,j) to I), and go to stage $i+1$.

Proof of Correctness:

The above construction yields an A such that any polynomial-time machine M_k either enumerates a list of NP machines that are not all categorical or else none of the enumerated machines accepts the language D_k. Note that, in the latter case, D_k is in UP since A has exactly one string of length $n = p_k^i$, $i \geq 1$, and thus is accepted by a categorical machine that simply queries A until it finds the string of length n and then accepts iff the string starts with a "one." On the other hand, no machine accepted by M_k can do this. Note that if some machine in this list would have tried to do this, it would have been made noncategorical by the entry of several strings of length n in A. In this case, D_k would not necessarily be in UP; this is no loss since M_k is not capable of producing a list of categorical machines to construct a complete language for UP.

Thus no polynomial-time machine can accept a set of categorical machines whose languages cover UP. ▶

Lemma 9: For every machine N_i that is categorical for all oracles, there is an oracle C such that

$$L(N_i^C) \neq \{1^n \mid n \geq 1 \text{ and } C \cap \Sigma^n \neq \emptyset\} = L_0.$$

Proof: Let $T(s) = s^k + k$ bound the running time of N_i and let n be such that $\binom{2^n}{2} > 2^n(n^k + k)$. If $N_i^{\emptyset}(1^n)$ accepts then $L(N_i^{\emptyset}) \neq L_0$. If, for some x in Σ^n, $L(N_i^{\{x\}})$ rejects, then again $L(N_i^{\{x\}}) \neq L_0$, so set $C = \{x\}$. Thus $N_i^C(1^n)$ must accept for every $C = \{x\}$, $|x| = n$. We show that this is not possible for a categorical N_i.

Let p_x denote the set of strings queried on the accepting path of $N_i^{\{x\}}(1^n)$. Choose a pair (a,b) of length n strings, $a \neq b$, so $a \notin p_b$ and $b \notin p_a$. Note that such a pair must exist as of the $\binom{2^n}{2}$ pairs of length n strings (c,d), at most $(n^k + k)2^n$ satisfy "$c \in p_d \vee d \in p_c$." Now $N_i^{\{a,\, b\}}(x)$ accepts on 2 paths, p_a and p_b, contradicting our assumption that N_i was always categorical. ▶

Theorem 10: There exists an oracle B such that

$$P^B \neq UP^B \neq NP^B$$

and UP^B has complete languages.

Proof: For ease of understanding we will view the oracle B as consisting of three disjoint parts (say written on different alphabets).

$$B = PSPACE \oplus E \oplus S.$$

Each part of the oracle plays a definite role. *PSPACE* will be used to determine if a given machine N_i is categorical for all possible oracle choices E and S for a given input. This will be used to construct a list of machines $N_{\sigma(i)}$ which will behave like N_i as long as N_i has no possibility of being noncategorical; if a possibility is detected that N_i can be noncategorical, $N_{\sigma(i)}$ will reject the input and all larger inputs. Thus all $N_{\sigma(i)}$ will be categorical and will be shown to cover all UP^B languages, thus guaranteeing the existence of a complete language in UP^B by Lemma 7.

The set E contains no more than one element of each length n and will be so constructed that the language

$$L_1 = \{1^n \mid (\exists x, y)[|x| = n \text{ and } x = 1y \text{ and } x \in E]\}$$

is not in P^B. Since L_1 is accepted by a categorical machine, E guarantees that

$$P^B \neq UP^B.$$

Part S will force all N_i which have infinitely many possibilities of being noncategorical to be noncategorical and, furthermore, guarantee that those machines that are categorical do not accept the language

$$L_2 = \{0^n \mid (\exists x)[|x| = n, x \in S]\}.$$

Since L_2 is in NP^B, we have:

$$P^B \neq UP^B \neq NP^B.$$

On the other hand, the list of machines $N_{\sigma(i)}$, categorical by construction, is such that

$$\{L(N^B_{\sigma(i)}) \mid i \geq 1\} = UP^B.$$

To see this, recall that if N_i has the potential to be infinitely often noncategorical then N_i^B is noncategorical. Otherwise, only for a finite number of inputs does N_i have the potential of not being categorical and there exists an equivalent machine N_j so that for any E', S', $N_j^{PSPACE+E'+S'}$ is categorical. Thus N_j^B is categorical and $L(N^B_{\sigma(j)}) = L(N_j^B) = L(N_i^B)$. But then by Lemma 7, UP^B has complete languages.

Construction of $B = PSPACE \oplus E \oplus S$

The construction proceeds in stages.

Stage 0: Set $E_0 = \varnothing$ and $S_0 = \varnothing$.

Stage i, $i \geq 1$. Pick a large n such that during previous stages all queries have been of length less than n and such that the running times on inputs of length n of M_i and N_i are small compared to 2^n.

The stage i consists of three parts.

a) Consider M_i. Let $B_{i-1} = PSPACE \oplus E_{i-1} \oplus S_{i-1}$. $M_i^{B_{i-1}}(1^n)$ accepts or rejects by having received polynomially many negative answers about strings of length n. Since 2^n is larger than the running time of M_i there exist strings in Σ^n not queried by $M_i^{B_{i-1}}(1^n)$. If $M_i^{B_{i-1}}(1^n)$ rejected, insert an unqueried string starting with a one into E_{i-1} to get E_i. Freeze E_i and go to part b.

b) Pick a new larger n so that no query in part a reached or exceeded length n and such that running time of N_i on $x, |x| = n$, is small compared to 2^n. ($\binom{2^n}{2} > 2^n(n^k + k)$.)

Consider N_i. If there is some possibility of forcing $N_i^{B_{i-1}}$ on x, $|x| \geq n$, to be noncategorical by proper choice of S_i, then do so. (Note that this is a non-constructive step, but this can easily be avoided and with a bit more work the oracle can be made recursive.) Freeze S_i and go to stage $i+1$. If not, go to part c.

c) We now know N_i is categorical on 0^n for all possible additions of strings in Σ^n to S_{i-1}. But then we know that by Lemma 9 we can add strings of length n to S_{i-1} to get S_i such that

$$L(N_i^{PSPACE + E_i + S_i}) \neq L_2.$$

Freeze S_i and go to stage $i+1$.

End of Construction of B.

The construction insures that $P^B \neq UP^B$. To see that $UP^B \neq NP^B$ observe that part b of stage i creates noncategorical machines. Only machines N_i reaching part c may be categorical and none of these can accept the language L_2 in NP. Thus $UP^B \neq NP^B$.

Finally, to see that UP^B has a complete language, observe that the list of machines $N_{\sigma(i)}^B$ is categorical and that it covers UP^B (see comments at the beginning of this proof). Thus by Lemma 7 UP^B has complete languages. ▸

The proof of Lemma 7 implies that if UP has no complete languages then there are languages in UP for which we can never prove that they are accepted by a categorical machine. That is, we will never be able to prove constructively that they are in UP. It can be seen that if UP has complete languages then for every L in UP there is a categorical machine accepting L with a very "simple" proof that it is categorical.

Corollary 11: If UP has no complete languages then for any sound axiomatizable formal system F there exists R in UP such that for no N_i with $L(N_i) = R$, can it be proven in F that N_i is categorical.

Proof: If for every R in UP there is an N_i with $L(N_i) = R$ for which we can prove in F that N_i is categorical, then we would have an r.e. list of categorical machines covering UP. But this implies that UP has complete languages because of Lemma 7. ▸

Therefore, if UP has no complete languages there must exist for every sound axiomatizable formal system F some R in UP so no N_i accepting R can be proven in F to be categorical.

4. Applications to Probabilistic Computations

This section applies the methods of this paper to the probabilistic class BPP [Gi]– languages accepted by a polynomial time probabilistic Turing machine, M, with bounded error probability. For such a machine, $(\exists \varepsilon > 0)(\forall x)$ $[|Pr(M(x) \text{ accepts}) - \frac{1}{2}| \geq \varepsilon]$. We say $L(M) = \{x \mid Pr(M(x) \text{ accepts}) \geq \frac{1}{2} + \varepsilon\}$, and say M "accepts" such x.

One must be careful in generalizing the UP noncompleteness result of Theorem 8. US [BG], a close cousin to UP, has complete languages in every relativized world. Nonetheless, our techniques yield interesting results for probabilistic computation models.

In this section, we answer a question from [Gi] by displaying a world where BPP has no complete languages. Related completeness (PP) and noncompleteness (R) results appear, respectively, in [Gi] and [Si]. We then develop a probabilistic version of Theorem 4, which showed that if UP has a complete set, then it has a complete set of the form $B \cap SAT$, where $B \in P$. Theorem 15 shows that if BPP has a complete set, then it has a complete set of the form $B \cap MAJORITY$, $B \in P$. $MAJORITY = \{f \mid f$ is true for at least half of all variable assignments$\}$ is the standard PP-complete set [Gi, p. 688]. Thus PP serves here as the parent class of BPP in the same way that NP serves in Theorem 4 as the parent class of UP.

Theorem 13: There exists a recursive oracle A such that BPP^A has no complete languages.

First we need the following lemma, analogous to Lemma 7. The proof is similar, and is omitted. However, note that the machines are clearly clocked and have a clearly known error bound.

Lemma 14: BPP^A has complete languages iff for every $0<\varepsilon<\frac{1}{2}$ there is an r.e. enumeration $\{M_{i_j}\}$ so that $\bigcup_{j \geq 0} L(M_{i_j}^A) = BPP^A$ and $(\forall j,x)$ $[|Pr(M_{i_j}^A(x)$ accepts$) - \frac{1}{2}| \geq \varepsilon]$.

Proof of Theorem 13:

By padding, the r.e. enumeration of Lemma 14 can be converted into a polynomial-time set of machines converting the same class of languages. So, it suffices to show: For every set S_i in P, either

1) $(\exists y \in S_i)(\exists x)[|Pr(M_y^A(x)$ accepts$) - \frac{1}{2}| < \frac{1}{4}]$

$(*)$

OR

2) $(\exists L_i \in BPP^A)(\forall j \in S_i)[L(M_j^A) \neq L_i]$.

Let $L_i = \{0^n \mid (\exists k \geq 1)[n = p_i^k$ and at least half of the strings at length n are in $A]\}$, where p_i is the i^{th} prime. We'll construct $A = \cup A_j$ by stages.

Stage $j = 0$: Set $A_0 = \varnothing$

Stage $j>0$:

From pairs (l,m), $l<j$, $m<j$, satisfying 1) $M_l(m)$ accepts, and 2) M_l has not been "emasculated," and 3) the pair (l,m) has not previously been chosen, choose a pair so $l+|m|$ is as small as possible. (If no pairs satisfy the conditions, set $A_j := A_{j-1}$ and go to the next stage.) For the chosen (l,m), we will now insure that either 1) M_m^A is not a BPP^A machine with error bound $\frac{1}{4}$, or 2) $L(M_m^A) \neq L_l$. By $(*)$ above, this proves the theorem.

Let w be 1) a number larger than the length of any string previously referenced in A, and 2) so large that $p(w) < \frac{1}{4} \cdot 2^w$, where $p(\cdot)$ is the polynomial time bound of M_m, and 3) a power of l.

Case 1: For all subsets S of the length w strings, $Pr(M_m^{A_{j-1} \cup S}(O^w) \text{ accepts}) > \frac{1}{4}$. In this case, put nothing of length j in the oracle and freeze all things of size up to $p(w)$. Set $A_j := A_{j-1}$. Thus $O^w \notin L_l$ yet $M_m^A(O^w)$ does not reject so $L(M_m^A) \neq L_l$.

Case 2: There are subsets S of length w strings for which $Pr(M_m^{A_{j-1} \cup S}(O^w)$ accepts) $\leq \frac{1}{4}$. Let S be a maximal such subset.

Case 2a: $|S| \geq \frac{3}{4} 2^w$. Set $A_j := A_{j-1} \bigcup S$. Thus M_m fails to accept L_l as $O^w \in L_l$ but $O^w \notin L(M_m^A)$.

Case 2b: $|S| < \frac{3}{4} 2^w$. Thus $|\bar{S}| \geq \frac{1}{4} 2^w$. By our maximality assumption, for each string $z \in \bar{S}$, $Pr(M^{A_{j-1} \cup S \cup z}(O^w) \text{ accepts}) > \frac{1}{4}$. However, if for one of these the probability is still less than $\frac{3}{4}$, by condition 1 of (✱) we've totally eliminated M_l from consideration and can mark it "emasculated."

Otherwise, we have the amazing situation that each of $\geq \frac{1}{4} 2^w$ strings, when added to $A \bigcup S$, jumps the probability of acceptance from $\frac{1}{4}$ to over $\frac{3}{4}$. We now show, by a counting argument, that this is impossible; probabilistic machines cannot react so dramatically to that many different events.

When we run $M_m^{\text{oracle}}(O^w)$, we may think of the machine as taking $2^{p(w)}$ bits of input (the "flips-set") to specify its coin flips. Each of the $2^{p(w)}$ "flips-sets" contributes $\frac{1}{2^{p(w)}}$ of the output probability. If changing the oracle from $A_{j-1} \bigcup S$ to $A_{j-1} \bigcup S \bigcup z$ moves the acceptance probability from at most $\frac{1}{4}$ to at least $\frac{3}{4}$, then z must be queried along the computation path of at least $\frac{1}{2} 2^{p(w)}$ of our flip-sets. So, since size of $|\bar{S}|$ is $\geq \frac{1}{4} 2^w$, this means we must reserve $\geq \frac{1}{4} \cdot 2^w \cdot \frac{1}{2} 2^{p(w)}$ slots along our computation paths. However, each of the $2^{p(w)}$

paths is only $p(w)$ long, so the total number of slots is at most $p(w)2^{p(w)}$. w was chosen so $p(w) < \frac{1}{4}2^w$, so there just are not enough slots available. The "amazing situation" we claimed impossible indeed is impossible.

END OF CASES

Thus, for each M_l either 1) M_l accepts no machine accepting L_l, and $L_l \in BPP^A$ (this happens when all machines in $L(M_l)$ trigger cases 1 or 2a; L_l is in BPP^A in this case) or 2) M_l accepts some machine that is not BPP^A with error bound $\frac{1}{4}$. By (✱), we are done. ▸

Now, we prove that if BPP has a complete set, then BPP has a complete set that is the intersection of a set from P with $MAJORITY$. $MAJORITY = \{f \mid f$ is true for at least $\frac{1}{2}$ its assignments$\}$ is PP-complete [Gi]. Theorem 15 is the probabilistic analogue of Theorem 4, and shows that PP serves here as the parent class of BPP in the same way that NP serves as the parent class of UP in Theorem 4.

Theorem 15: If BPP has a complete set, then it has a complete set of the form $B \cap MAJORITY$, where $B \in P$.

Proof: Let S be a BPP set accepted by machine M. W.l.o.g. suppose $(\forall x)$ $[|Pr(M(x) \text{ accepts}) - \frac{1}{2}| \geq \frac{1}{4}]$. Run a probabilistic version of Cook's reduction on $M(x)$. This yields a formula F_x that codes the run $M(x)$. F_x will have "flip variables," describing the random choices, and other variables: $F_x = F(y;z)$, y the flip variables. Loosely, F_x looks like: (start in initial state) $\wedge \cdots \wedge \bigwedge_{step\ k} ((y_k \wedge \cdots) \oplus (\bar{y}_k \wedge \cdots))$.

Write $Pr(F)$ for the probability that F is true when each of its variables is randomly set to True or False. Since for each choice y of flips the other bookkeeping variables z are completely determined,

$$1)\ x \in S \Rightarrow Pr(F_x) \geq \frac{(\frac{3}{4}2^{|y|})}{(2^{|y|+|z|})} = \frac{3}{4} \cdot \frac{1}{2^{|z|}}$$

$$2)\ x \notin S \Rightarrow Pr(F_x) \leq \frac{1}{4}\frac{1}{2^{|z|}}.$$

Let $F'_x = F_x \vee G$. $G = u_1 \wedge (w_1 \vee w_2 \vee \cdots w_{|z|+1})$, where $u_1, w_1, \cdots, w_{|z|+1}$ are new variables. By cases, ($**$): $Pr(F'_x) = Pr(F_x \vee G) = Pr(F_x) + Pr(G) -$

$Pr(F_x)Pr(G)$. Clearly, $Pr(G) = \frac{1}{2}(1-\frac{A}{2})$, where $A = \frac{1}{2^{|z|}}$. All we have to do is note that $Pr(F'_x \mid x \in S) > \frac{1}{2}$ and $Pr(F'_x \mid x \notin S) < \frac{1}{2}$. Why do these hold? Since $Pr(F_x \vee G)$ is monotonic in $Pr(F_x)$, by ($**$) above we have,

$$Pr(F'_x \mid x \in S) \geq Pr(F'_x \mid P(F_x) = \frac{3}{4}A)$$
$$\geq \frac{1}{2} + \frac{A}{8} + \frac{3A^2}{16} > \frac{1}{2}$$

$$Pr(F'_x \mid x \notin S) \leq Pr(F'_x \mid P(F_x) = \frac{A}{4})$$
$$\leq \frac{1}{2} - \frac{A}{8} + \frac{A^2}{16} < \frac{1}{2}$$

Let $B = \bigcup_{x \in \Sigma^*} F'_x$. $B \in P$ since we can look at a formula and tell if it came from the machine M. By the arithmetic above, we know $F'_x \in MAJORITY$ if and only if $x \in S$. Since S is BPP-complete and F'_x is easily computed from x, $B \cap MAJORITY$ is also BPP-complete. (Given L' in BPP, on input x reduce to a query to S, reduce that to a formula F, and convert that to a formula $F' \in B$.) ▶

As a final note, the set S of Theorem 4 satisfied the (UP-like) property that its formulas had at most one satisfying assignment. Each formula F' in our set B of Theorem 15 has the (BPP-like) property that the probability that F' is satisfiable after a random assignment of the flip variables is bounded away from $\frac{1}{2}$.

References

[BD]A. Borodin and A. Demers. Some Comments on Functional Self-Reducibility and the *NP* Hierarchy. Department of Computer Science Technical Report TR76-284, July 1976. Cornell University, Ithaca, New York.

[BG]A. Blass and Y. Gurevich. On the Unique Satisfiability Problem. *Information and Control 55* (1982), 80-82.

[Be] P. Berman. Relations Between Density and Deterministic Complexity of *NP*-Complete Languages. *Proceedings Symposium on Mathematical Foundations of Computer Science*, 1978, Springer-Verlag, 63-71.

[CH]J. Cai and L. Hemachandra. The Boolean Hierarchy: Hardware over NP. To appear in *Proceedings of the Structure in Complexity Theory Conference, Lecture Notes in Computer Science* (1986), Springer-Verlag.

[Co] S.A. Cook. The Complexity of Theorem-Proving Procedures. *Proceedings ACM Symposium on Theory of Computation* (1971), 151-158.

[Gi] J. Gill. Computational Complexity of Probabilistic Turing Machines. *SIAM Journal on Computing 6* (1977), 675-695.

[GJ] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Co., 1979.

[GS]J. Grollmann and A.L. Selman. Complexity Measures for Public-Key Cryptosystems. *Proceedings IEEE Symposium on Foundations of Computer Science* (1984), 495-503.

[HI] J. Hartmanis and N. Immerman. On Complete Problems for $NP \cap CoNP$. *Automata Languages and Programming, Lecture Notes in Computer Science 194* (1985), Springer-Verlag, 250-259.

[HU]J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Massachusetts, 1979.

[Li] M. Li. *Lower Bounds in Computational Complexity.* Ph.D. Dissertation, Cornell University, 1985.

[Si] M. Sipser. On Relativization and the Existence of Complete Sets. *Automata, Languages and Programming, Lecture Notes in Computer Science 140* (1982), Springer-Verlag, 523-531.

[Va] L. Valiant. Relative Complexity of Checking and Evaluating. *Information Processing Letters 5* (1976), 20-23.