Decision trees and downward closures* (Extended Abstract)

Russell Impagliazzo and Moni Naor
University of California, Berkeley

1. Introduction

A general principle of complexity theory is that smaller complexity classes are easier to separate than larger ones. This is supported by various upward closure results, proved by trivial padding arguments, which state that if a relationship between complexity measures holds for all problems of a fixed complexity, it will hold for all problems of greater complexity. Along similar lines, we know that for certain hierarchies of complexity classes (for example the polynomial hierarchy), a collapse at a low-level would collapse the entire hierarchy. In both situations, a separation between two larger classes would imply the separation of corresponding smaller classes.

It is also true that methods from concrete complexity have been successfully used to separate small classes; for example, AC^0 from NC^1 [FSS],[Y2], or the following easy proof separating sub-linear deterministic and non-deterministic times. Consider the problem: is there a bit in the input which is a one? Non-deterministicly, one can simply examine an arbitrary bit, accepting if the bit is one; deterministicly, any algorithm to solve this problem must look at all bits of the input. Thus, the problem is in $NTIME(logn)$, but requires $DTIME(n)$. Less trivially, consider the following problem and its complement: given an n node undirected graph in the form of an adjacency matrix, does it contain a star of size n as a subgraph? (Here, a star is a graph that consists of a node connected to all other nodes). The nondeterministic and co-nondeterministic complexities of this problem are n, since one can guess a node connected to all others, or a missing edge at every node. The deterministic complexity is $\Theta(n^2)$.

In this paper we show that, at least for some measures of complexity, this intuition is misleading. Specifically, we give several *downward* closure results, showing that if $P=NP$, then $DTIME(polylog) = NTIME(polylog) \bigcap Co-NTIME(polylog) = RandomTIME(polylog)$. Thus any result separating the above low-level classes would actually prove $P \neq NP$! We do this by giving uniform versions of simulations in the decision tree model of concrete complexity. The only other result we can think of which is at all similar is that due to Hartmanis, Sewelson and Immerman [HSI] proving that the E=NE question is equivalent to the existence of

*This work was supported in part by NSF grant DCF-85-13926.

sparse sets in NP - P.

Our results correspond closely to *robust* and *generic* oracle results ([HH],[BI] respectively). However, not only do we present a new interpretation of these results, but many of the theorems presented here give corresponding new results in these models. In particular, the issue of probabilistic computation in these models was hitherto unresolved. Also, we weaken the assumption needed for the simulation of $NP \bigcap Co-NP$ type computations in these models. Note that, because of the standard upward closure results if we could prove these collapses without assumptions we would have shown $P=NP \bigcap Co-NP$ and $P=BPP$ respectively.

Notation

We will denote by DPL the class of predicates computable in deterministic polylog time (see section 2 for our model of sublinear computation). Similarly, NPL denotes nondeterministic polylog time , Co-NPL denotes co-nondeterministic polylog time, and BPPL denotes probabilistic polylog time computation with two-sided error bounded away from 1/2.

Organization of the paper

In section two we present our model of sub-linear computation, and show the equivalence of questions involving sub-linear computation with corresponding ones involving robust or generic oracles. The result that, if $P=NP$, then $DPL=NPL \bigcap Co-NPL$ then follows from either [HH] or [BI].

In section three, we show that if $P=NP$ than $DPL=BPPL$. We also extend this proof to a general technique for making decision tree simulations uniform.

In section four, we address the problem of whether the full power of the $P=NP$ assumption is necessary for these results. For the DPL vs. $NPL \bigcap Co-NPL$ question, we give a weaker condition under which collapse still occurs. This assumption, that it is easy to find an accepting path for a poly-time NTM accepting all of $\{0,1\}^*$, is a fairly natural question presented, to our knowledge, for the first time in this paper. We show that , at least in relativized worlds, this assumption is not equivalent to $P=NP \bigcap Co-NP$; this can be interpreted as showing that, for $NP \bigcap Co-NP$ type computation, search problems do not reduce to decision problems.

Section five consists of conclusions and open problems.

2. Various Models and their Equivalence

The model of Turing Machine we use is standard in all but one respect. In order for sub-linear time computation to be nontrivial, we need to have random access to bits of the input. For this purpose, we give our model of Turing Machine a special query tape. When the machine enters one of a designated set of query states, the bit of the input addressed by the contents of the query tape appears in the first square of this tape. This is similar to the model used by Chandra Kozen and Stockmeyer [CKS], and Ruzzo [R] for alternating time.

Decision Tree Complexity Measures

Decision tree complexity is a model of concrete complexity in which the only charge is for examining a bit of the input [SW],[Y1]. Algorithms in this model correspond to binary trees, whose nodes are labeled with addresses of bits of the input, and whose leaves are labeled either "accept" or "reject". To run the algorithm on an input, start at the root.

Find the value of the bit of the input written there. If this value is 0, go to the right child; if 1, the left. Recurse. When a leaf is reached, accept or reject according to its label. Thus, each such tree with nodes labeled from 1 to n determines a subset of $\{0,1\}^n$. The cost of a tree will be its depth. For $L \subseteq \{0,1\}^n$, let D(L) be the least cost of a tree recognizing L.

We can also define measures corresponding to non-deterministic and probabilistic computation in this model. Let $L \subseteq \{0,1\}^n$. A 1-certificate for L is a sequence of indices of bits of the input along with contents at these places (e.g., "third bit=1, tenth bit=0, twelfth bit=0, ...") so that any input x which agrees with the sequence in the specified places is in L. A 0-certificate for L is a 1-certificate for the complement of L. The 1-sided non-deterministic complexity of L, $N_1(L)$, is the maximum length, over x in L, of the shortest 1-certificate for L agreeing with x. (Thus, a prover who knows the input x can convince a verifier that x is in L by revealing fewer than $N_1(L)$ bits of x.) Let $Co-N_1(L)$ be N_1 of the complement of L, and let $N(L)$ be the maximum of $N_1(L)$ and $Co-N_1(L)$. (Problems with small N complexity correspond intuitively to problems in $NP \bigcap Co-NP$.)

We will consider randomized algorithms that *allow* an error, but the error will be *bounded* away from 1/2. A probabilistic decision tree looks like a deterministic decision tree, except that certain nodes are designated coin-flipping nodes rather than labeled with a bit of input. At these nodes, the algorithm decides randomly whether to move to the left or right child. The cost of a probabilistic decision tree is the expected number of input nodes transversed during a computation on the worst case input. Such a tree recognizes a subset $L \subseteq \{0,1\}^n$ if , $\forall x \in L$, it accepts with probability greater than 3/4, and $\forall x \notin L$, it accepts with probability less than 1/4. let BP(L) be the least cost of a tree recognizing L. (In this paper, as opposed to [SW] who insist on Las-Vegas algorithms, we allow probabilistic trees to have a constant error on both sides.

Theorem 2.1[Blum]: For any $L \subseteq \{0,1\}^n$, $D(L) \leq N^2(L)$.

Proof: Note that any 1-certificate and any 0-certificate must intersect in at least one bit at which they disagree. Otherwise, we can find an input containing as substrings both certificates; this input must thus be both in L and in $\bar{L}$.

Consider the following deterministic decision tree algorithm: start by choosing any 1-certificate of length $\leq N(L)$ and querying all the indices mentioned in it. If all agree with the 1-certificate we are done. Else, continue choosing another 1-certificate consistent with the information about the input we have so far. Repeat N(L) times, or until we can no longer find a short 1-certificate consistent with our information.

We claim that this algorithm eventually finds either a 1-certificate or a 0-certificate. Assume our input x is not in L. Then x has a 0-certificate C of length $\leq N(L)$. If the algorithm runs for fewer than N(L) rounds, it is because it has found either a 0-certificate or a 1-certificate (impossible in this case). Else, at each round i, $1 \leq i \leq N(L)$, the algorithm finds a 1-certificate D_i consistent with the information so far. Each D_i must disagree with C at some index a_i. Now, for $i<j$, $a_i \neq a_j$, since otherwise, at time j we would already have asked about bit a_j. Then D_j would have to be consistent with the input, and hence with C, at this place, contrary to the definition of a_j. This means that, for each run of the algorithm, we find a

new bit of C, so by N(L) runs we have found all of C. Thus, for every input x not in L, our algorithm finds a 0-certificate ; hence, if we have not found a 0-certificate , we can conclude the input is in fact in L. □

Robust and Generic Oracles

For a OTM (Oracle Turing Machine) M and an oracle O, let L_{M^O} represent the language accepted by M with oracle O. We say two oracle machines M,N are *robustly equivalent* if for every oracle O, $L_{M^O}=L_{N^O}$. We define *robustly complementary* analogously. (See [HH].)

Let w be a finite sequence of 0's and 1's. The interval determined by w is the set of oracles which agree with w on the first $|w|$ bits. Say w *forces* $M=N$ iff for every oracle O in the interval determined by w, $L_{M^O}=L_{N^O}$. A oracle G is generic, if for every OTM's M and N so that $L_{M^G}=L_{N^G}$, there is a finite prefix w of G, with w forcing $M=N$. (See [BI]. This definition should actually be for 1-genericity [Ku].)

Theorem 2.2: The following are equivalent

i) $DPL=NPL \bigcap Co-NPL$

ii)For every robustly complementary polynomial time nondeterministic OTM's N,M, there exist a deterministic polynomial time OTM Q robustly equivalent to M.

iii)For a generic oracle G, $P^G=NP^G \bigcap Co-NP^G$.

Proof: We will prove (i) = > (ii) = > (iii) = > (i).

(i) = > (ii) Let M,N be NPOTM's. Let M',N' be polylog NTM's which treat their inputs as consisting of two parts: an input of size k, and an oracle of size $2^{poly(k)}$ where poly is some polynomial time bound for both M and N. M',N' simulate M and N accordingly, and thus take time poly(k) (which is O(log n) where n is the size of the input). Since M,N disagree on every input x and every oracle O, M',N' must recognize complementary languages. Thus, there will be a deterministic polylog machine Q' recognizing $L_{M'}$. Let Q be the deterministic polynomial OTM which simulates Q' in a converse manner to that described for M' and N'.

(ii) = > (iii) is from [BI].

(iii) = > (i) Let M', N' be polylog time non-deterministic machines accepting complementary languages. Let M and N be polynomial time NOTM's which, on input k, simulate M' and N', respectively, on an input of length k, making query (k,i) to the oracle whenever the polylog machine would ask about bit i of its input. Note that M and N are robustly complementary, and that the parts of the oracle queried on distinct inputs are disjoint. Since M and N are robustly complementary, L_M^G and L_N^G are complementary for G generic, and thus, by assumption, there is a polynomial time deterministic machine D with oracle G accepting L_M^G. Since G is generic, there is a finite prefix of G w forcing $D=M$. Let D' be a polylog time machine simulating D, but with w "hard-wired" into the program; i.e., on an input of length k, D' simulates D, but when D makes an oracle query in the domain of w, D' answers according to w, when D makes a query of the form (k,i), D' queries bit i of its input, all other queries being answered "0". A run of D' simulates a run of D on some oracle consistent with w, and D' will therefore agree with M' on all inputs k where no bit (k,i) is in the range of w. Since the range of w is finite, we can thus hardwire a finite list of strings into D' to get a deterministic machine D" always agreeing with M'. □

Similar equivalences will hold for

almost all identities involving different types of computation; in particular, $DPL = BPPL$ is equivalent to $P = BPP$ relative to a generic oracle.

Theorem 2.3[BI],[HH]: If $P = NP$, then for any pair of robustly complementary NPOTM's M,N there exists a deterministic polynomial time OTM Q robustly equivalent to M.

We prove a somewhat sharper version of this theorem in section 4.

Corollary 2.4: If $P = NP$, then $DPL = NPL \bigcap Co-NPL$.

3. Random time

In this section we compare $BPPL$ with DPL, showing that, if $P = NP$, these classes collapse. As before, this is a uniform version of a result in the decision tree model.

Theorem 3.1(Nisan[N]): Let $L \subseteq \{0,1\}^n$, then $4 \cdot BP^2(L) \geq N(L)$.

We provide Nisan's proof to illustrate the non-constructive nature of the proof.

Proof: Let x be an n-bit string, and let s be a set of bits. We say L is sensitive to s on x if $x \in L \leq > x_s \notin L$, where x_s is x with all of the bits in s flipped. The block sensitivity of L on x is the maximum b so that there exist disjoint sets of bits, $s_1, \ldots, s_b$ with L sensitive to s_j on x for all $1 \leq j \leq b$. The block sensitivity of L, bs(L), is the maximum over all $x \in \{0,1\}^n$ of the block sensitivity of L on x. Let L be sensitive to s on x. Any probabilistic algorithm which, with probability $> 1/2$ on input x, asks no bit of s, will fail with probability $> 1/4$ either on x or on x_s. Thus, if L has block sensitivity b on x, any probabilistic decision tree for L with error probability $<$ 1/4 must ask at least $b/2$ queries on average on input x. Therefore, $bs(L) \leq 2 \cdot BP(L)$.

We now show that $N(L) \leq bs^2(L)$. Given x, we will find a certificate for x of length $\leq bs^2(L)$. Without loss of generality, assume $x \in L$. Say s is a minimal block for x if L is sensitive to s on x, but to no proper subset of s on x. Note that any set s for which L is sensitive on x contains some minimal block s' for x. We claim that the size of any minimal block is is no more than $bs(L)$. This is because, if s is a minimal block for x, L is sensitive to every bit of s on x_s. Let $s_1, s_2, \cdots, s_b$ be a maximal collection of disjoint minimal blocks for x. We claim that the bits of x the union of the s_j's form a 1-certificate. For, if not, there is a $y \notin L$ agreeing with x at all these bits Let t be the set of bits where x and y disagree. L is sensitive to t on x, hence t contains a minimal block t' for x. t' is disjoint from each s_j contradicting the maximality of $s_1, s_2, \cdots, s_b$. Since by definition $b \leq bs(L)$ and each $|s_j| \leq bs(L)$ we have the certificate of the desired length. □
The proof gives no way of recognizing a certificate.

Theorem 3.2: Assume $P = NP$, then $DPL = BPPL$.

Proof: We know by Corollary 2.4 that it is enough to show that, assuming $P = NP$, then $BPPL \subseteq NPL \bigcap Co-NPL$. We will show that $BPPL \subseteq NPL$; $BPPL \subseteq Co-NPL$ is similar.

Let $L \in BPPL$, $L_n = \{w \in L : |w| = n\}$ then has probabilistic decision tree complexity $\in O(log^k n)$. Hence, by theorem 3.1 it has nondeterministic decision tree complexity $\in O(log^{2k} n)$. We would like to apply that fact, by guessing non-deterministically a short 1-certificate for the instance, and then verifying that it is indeed a certificate. However, the proof of Theorem 3.1 does not yield a constructive way of checking if a string is a certificate. The only constructive algorithm we have is the probabilistic $O(\log^k n)$ algorithm for L.

We will make use of that algorithm, to give an Arthur Merlin (actually an Merlin Arthur). game for the problem of 1-certificate recognition. Since we are assuming $P=NP$, and bounded rounds AM belongs to the polynomial hierarchy we have that $AM=P$, and hence we have a polynomial (in the length of the certificate) time algorithm for 1-certificate recognition. This yields an NPL algorithm for L.

From the discussion above, it is enough if we put the complement of the problem of 1-certificate recognition in Babai's class MA [B]. In a Merlin Arthur game, an all powerful Merlin makes a polynomial length statement to a probabilistic poly time machine Arthur, who is supposed to verify the statement with bounded probability of error. Formally, a MA game is a polytime predicate of 3 inputs x,y and r, $Q(x,y,r)$ with $|y|,|r|$ a fixed polynomial of $|x|$. The probability of accepting x is the maximum over all y of $Prob[Q(x,y,r)=1]$ where all the r's are assumed to have equal probability. A language $L \in MA$ if there is a Merlin Arthur game $Q(x,y,r)$ such that, $\forall x \in L$ $Prob[Q \text{ accepts } x] > 2/3$ and $\forall x \notin L$ $Prob[Q \text{ accepts } x] < 1/3$.

The protocol to test for not being a 1-certificate for L_n goes as follows: If a set of bit S is not a 1-certificate, it can be extended to an input x, $x \notin L_n$. Thus, since L_n has nondeterministic decision tree complexity $\in O(\log^{2k} n)$, x contains a 0-certificate of length $O(\log^{2k} n)$. This 0-certificate is consistent with the (fallacious) candidate for 1-certificate. Thus, Merlin's move is to give an extension T of S, adding fewer than $O(\log^{2k} n)$ bits, such that T is a 0-certificate. Arthur than simulates the probabilistic algorithm for L, answering queries about bits mention in T consistent with T, and all other queries with a zero. Arthur accepts iff the algorithm rejects. If S was a 1-certificate, then any extension T is such, and hence T with 0's at the other places $\in L_n$. Therefore the algorithm will accept with $Prob > 2/3$ and Arthur rejects with $Prob > 2/3$ no matter what Merlin's move was. If S was not a 1-certificate, Merlin can make T a 0-certificate, so the algorithm rejects with $Prob > 2/3$ and Arthur accepts with $Prob > 2/3$. $\square$

This gives us a general technique for making decision tree simulations uniform. For example, we can define equivalents of AM games for both decision tree and sublinear time. Using the same proof as Theorem 3.1, we find that $max(AM(P), Co-AM(P))$ is polynomially related to $N(P)$ in the decision tree model. We can then conclude, by similar arguments to the previous theorem that if $P=NP$, $DPL = AM(polylog) \bigcap Co-AM(polylog)$ This combines both Theorems 2.3 and 3.2.

4. Tautology Search and relativization results

An obvious problem that arises from the previous section is to determine the minimum complexity assumptions that would still result in the aforementioned collapses. Since these collapses in polylog time classes imply $P=NP \bigcap Co-NP$ and $P=BPP$ respectively we will need assumptions at least as strong. We introduce the notion of tautology search (TS), and show that the assumption that it can be done in polynomial time can replace the assumption $P=NP$ in the first collapse. "$TS(poly) \in P$" abbreviates: for all NPTM's, N, such that $L_N = \{0,1\}^*$, there exists a polynomial time function, $f_N(x)$ which finds an accepting N path for x. "$TS(poly) \in P$" is equivalent to saying that, for problems in $NP \bigcap Co-NP$, both decision and search is in P.

Theorem 4.1:$TS(poly) \in P$ implies $DPL = NPL \bigcap Co-NPL$.
Proof: Let M and N be complementary polylog time NTM's. Recall the proof that $D(P) \leq N(P)^2$. The main idea was that asking all the queries along any 1-certificate for P shortens by one query every 0-certificate consistent with the information about the input thus found. Think of accepting paths for M as being 1-certificates, those for N as 0-certificates. (Here we mean computation paths on any input of length n which is accepted. We identify a path with the set of bits of the input queried along the path.) Thus, if we could find a sequence of polylog accepting paths for M, each consistent with the actual input at places queried in the previous ones, we would have asked all the places of the input queried by any accepting path for N. By symmetry, a corresponding sequence for N would be just as good. Thus, at each stage of our algorithm, we would like to find either an accepting path for N or one for M consistent with all the information we have about the input to date. We then query all the places mentioned along the path, updating our information. We repeat 2*polylog +1 times, where polylog is a time bound for both M and N, accepting if the last path was an accepting path for M rather than N.

To make use of our complexity assumption, we need to reduce the problem of finding an accepting path for either N or M consistent with the information we have so far to a search version of a problem in $NP \bigcap Co-NP$. On any fixed input, either M has an accepting path or N does. For certain dramatically compressible inputs, say those consisting mostly of 0's, the problem of deciding which machine accepts will be in NTIME $\bigcap$ Co-NTIME polynomial in the compressed length, and thus, by our assumption, finding some accepting path will be feasible in time polynomial in the compressed length.

Formally, let T be the following NPTM: T has two inputs: a length n (in binary) and a sequence $b_1, .. b_k$, where $b_i \leq n$. T non-deterministically choses one of M or N, and simulates, again, non-deterministically, a computational path for the machine chosen, say M, on an input of length n. When M asks about place i of the input, T checks whether i is an element of the sequence $b_1,..b_k$. If so, the simulated answer is 1; if not, 0. Clearly, T accepts iff either M or N accepts the input of length n which is 1 in the places mentioned in the sequence, and 0 elsewhere; i.e., T always accepts. Thus, by our assumption, it is possible in polynomial time (in n and k) to find an accepting path for T, which will yield a corresponding path for either M or N.

Our polylog algorithm to recognize L_M follows the basic outline sketched above. We let n be the length of the input, and initialize the sequence $b_1,..b_k$ to the empty sequence. We find an accepting path for T on this input, and query in all places mentioned along this path. We update the sequence b_i to include all the 1's found in the querying process. We repeat $2 \cdot polylog + 1$ times. We then test to see whether the last path found is accepting for M, accepting if it is.

Remark 1
$P = NP \Rightarrow TS(poly) \in P \Rightarrow P = NP \bigcap Co-NP$.
The implications are strict, at least in relativized worlds.

Proposition 4.2:There are oracles A,B such that
i) $P^A = NP^A \bigcap Co-NP^A$, but $TS(poly) \notin P^A$.
ii) $TS(poly) \in P^B$, but $P^B \neq NP^B$.
Proof: The intuition behind the con-

struction of oracle A is as follows: Assume there is an onto one-way function f from 2n bit string to n bits strings. Then the NTM n which guessed, on input x, a string y, $|y|=2|x|$, accepting if $f(y)=x$, would always accept. However , finding such a y would not be feasible in polynomial-time Formally, we construct a class of oracles which define onto functions from 2n bits to n bits strings. The oracle construction is to first take an oracle A_1, with $P^{A_1}=NP^{A_1}$, then adjoin an oracle A_2 which is generic in the above oracle class. (The class will be a natural class of oracles in the sense of [BI]; thus it will have generic elements.)

For B, we give what is essentially the construction in [BGS] of an oracle relative to which $P=NP \bigcap Co-NP \neq NP$. Call an oracle linearly sparse if it has at most one element of any length. The class of linearly sparse oracles is a natural class, and thus has generic elements. Let B consist of any oracle A with $P^A=NP^A$, together with a generic linearly sparse oracle L. Let T be any NPTM with oracle B which always accepts. (From now on, we treat A as "background"; i.e., we treat T as having oracle L, and assume P=NP in the "real" world; i.e, complexity relative to A.) L will then have a finite prefix l forcing T to always accept; i.e., T with any linearly sparse oracle L' agreeing with l always accepts. On input x, we find an accepting path for T as follows: first, find an accepting path for T on x with the oracle L' which is 0 on all strings not in l. Check to see if all places in the oracle queried by T are in fact consistent with L. If so, we have found an accepting path for T on L; if not, we have found at least one string in L which is not in L'; update L' accordingly. Since L' is always a subset of L, and since L has at most n elements of length <= n, we can repeat this process at most poly(|x|) times, where poly is a time bound for T, before we find an accepting path. Since L' never exceeds poly(|x|) + c elements, and since P=NP relative to A, each phase of the algorithm can be done in polynomial time.□

Remark 2:We can define the $TS(polylog) \in DPL$ question analogously to $TS(poly) \in P$. Somewhat surprisingly, the polylog version of this question is simply false, regardless of complexity assumptions.

Proposition 4.3 $TS(polylog) \notin DPL$
Proof:From Ramsey theory [GRS], we know that every graph on n nodes has either a clique or an independent set of size at least $1/2\log n$. On the other hand, we know that there exist graphs on $n^{1/4}$ nodes without such cliques or independent sets. Thus a machine which treats inputs of size $\binom{n}{2}$ as graphs on n nodes, guesses nondeterministicly $1/2\log n$ nodes, accepting if they form a clique or independent set has an accepting path on every input. However, the existence of large graphs without $1/2\log n$ cliques or independent sets leads to an adversary argument showing that an deterministic algorithm must examine at least $\Omega(n^{1/4})$ bits of the input before finding the desired clique or independent set.

This argument also shows that
Proposition 4.4: For G generic $TS^G(poly) \notin P^G$.

This leads to an oracle construction showing that it is improbable that $DPL=NPL \bigcap Co-NPL$ implies $TS(poly) \in P$:

Proposition 4.5: There exists an oracle O, such that

$$DPL^O=NPL^O \bigcap Co-NPL^O,$$

but $TS^O(poly) \not\subseteq P^O$

Proof: If A and B are oracles, let $A \oplus B$ be the oracle defined by $A \oplus B(1,x) = A(x)$ and $A \oplus B(0,x) = B(x)$. Let O be any oracle of the form $A \oplus G$ where $P^A = NP^A$ and G is generic with respect to A. From proposition 4.4 relativized to A we have $TS^O \not\subseteq P^O$. Let H be an oracle generic with respect to $O = A \oplus G$. Then $G \oplus H$ is generic with respect to A, and so since $P^A = NP^A$, from a relativized version of theorem 2.3 we have that

$$P^{A \oplus (G \oplus H)} = NP^{A \oplus (G \oplus H)} \bigcap Co-NP^{A \oplus (G \oplus H)}.$$

In other words, in the world of O, a generic oracle H makes $P^H = NP^H \bigcap Co-NP^H$. Thus, from Theorem 2.2 relativized to O, we can conclude that

$$DPL^O = NPL^O \bigcap Co-NPL^O.$$

For the same construction and by similar arguments we have

Proposition 4.6: There exists an oracle O such that $DPL^O = BPPL^O$, but $P^O \neq NP^O$.

5. Conclusions and Open Problems

Our results show that sub-linear time computation has enough power to code interesting questions in polynomial-time complexity. Perhaps this is true of other "low-level" complexity classes. If this is a general phenomenon, it would be a damper on a current research program, in which researches hope to gradually separate classes "from the bottom up", with ever more powerful combinatorial techniques. On the other hand, an eternal optimist might claim that these results suggest a hope that a separation of P from NP is within our grasp.

A broad issue our research suggests is the question of what other consequences collapses of high level complexity classes have on low level classes. The nearest parallel we are aware of is the result of Hartmanis,Swelson and Immerman [HSI83], stating that E=NE is equivalent to the non-existence of sparse sets in NP-P.

Another line for future inquiry, is whether we can weaken the assumptions under which downward closures occur. Specifically, does $P = NP \bigcap Co-NP$ imply $DPL = NPL \bigcap Co-NPL$? The current assumption seems not much stronger than $P = NP \bigcap Co-NP$ We have no oracles relative to which $P = NP \bigcap Co-NP$ or $P = BPP$, yet the corresponding collapses fail to occur.

6. Acknowledgements

Discussions with Leonid Levin and Umesh Vazirani lead to this research.

7. References

[A] M. Ajtai, Σ_1^1- formulae on finite structures, Pure and Apllied Logic, 24, (1983), pp. 1-48.

[B] L. Babai, Trading Group theory for Randomness, Proc. 17th STOC, (1985), pp. 421-429.

[BGS]
T. Baker, J. Gill and R. Solovay, Relativizations of the P=NP question, SIAM J. on Computing 4, (1975), pp. 431-452.

[BI] M. Blum and R. Impagliazzo Generic Oracles and Oracle Classes, Proc. 28th FOCS, (1987).

[CKS]
A. Chandra, D. Kozen, L. Stockmeyer, Alternation, Journal of the ACM 28 (1981), pp. 114-133.

[FSS]M. Furst, J. Saxe and M. Sipser, Parity circuits and the polynomial time hierarchy, Mathematical System Theory, 17, (1984), pp. 113-28.

[GRS]
R. Graham, Rothshild and J. Spencer, Ramsey Theory,Wiley, New-York, 1980.

[HH]J. Hartmanis and L. Hemachandra, One-way Functions, Robustness, and the Non-Isomorphism of NP-complete Sets Structure in Complexity Theory, 1987

[HSI]J. Hartmanis, V. Sewelson and N. Immerman, Sparse Sets in NP-P : EXPTIME vs. NEXPTIME, Proc 15th STOC, (1983).

[Ku] S. Kurtz, Notions of Weak Genericity, J. of Symbolic Logic, 48, 3, (Sep. 83), pp. 724-743. (1983)

[N] N. Nisan, Probabilistic vs. Deterministic Decision trees and CREW PRAM Complexity, Manuscript, UC Berkeley, (1987).

[R] L. Ruzzo, On Uniform Circuit Complexity, Journal of Computer and Systems Sciences 22 (1981), pp. 365-383.

[SW]Probabilistic Boolean Decision Tree and the Complexity of Evaluating Games, Proc. 27th FOCS, (1986), pp. 29-38.

[Y1] A.C. Yao, Probabilistic Computations: Toward a Unified Measure of Complexity, Proc. 18th FOCS, (1977).

[Y2] A.C. Yao, Separating the Polynomial Time Hierarchy by Oracles, Proc. 26th FOCS, (1985) pp. 1-10.