On Strong Separations from AC^0*

Eric Allender†
Department of Computer Science
Rutgers University
New Brunswick, NJ 08903
allender@cs.rutgers.edu

Vivek Gore‡
Department of Computer Science
Rutgers University
New Brunswick, NJ 08903
gore@paul.rutgers.edu

September 14, 1995

*A preliminary version of this paper appeared as [AG91a].

†Supported in part by NSF grants CCR-8810467 and CCR-9000045.

‡Supported in part by a DIMACS research assistantship.

DIMACS is a cooperative project of Rutgers University, Princeton University, AT&T Bell Laboratories and Bellcore.

DIMACS is an NSF Science and Technology Center, funded under contract STC-88-09648; and also receives support from the New Jersey Commission on Science and Technology.

SUMMARY

As part of a study of almost-everywhere complex sets, we investigate sets that are immune to AC^0; that is, sets with no infinite subset in AC^0. We show that such sets exist in P^{PP} and in $\mathrm{DSPACE}(\log n \log^* n)$.

Our main result is an oracle construction indicating that any improvement in these immunity results will represent a significant advance, in that we show that any answer to the question:

Are there sets in NP that are immune to AC^0?

will provide non-relativizable proof techniques suitable for attacking the Ntime vs Dtime question. That is, we show that the existence or nonexistence of AC^0-immune sets in NP has consequences concerning the complexity of sets in deterministic and nondeterministic exponential time.

1 Introduction

1.1 Immunity

What does it mean to show that a set is complex? As a first attempt, one might show that the set does not lie in some "easy" complexity class such as $\mathrm{DTIME}(n^5)$. However, if a set L is not in $\mathrm{DTIME}(n^5)$, it means merely that for each machine M accepting L, M must run for more than n^5 steps on infinitely many inputs – although this set of inputs could be *extremely* sparse – and it might even be the case that M runs in linear time on the overwhelming majority of inputs. In some settings, such a set L would not be considered sufficiently "complex," and thus two stronger notions of complexity have often been considered: randomness (or Church-randomness) and almost-everywhere complexity.

A set L is said to be *Church-random* with respect to a class $\mathcal{C}$ if for each set $A \in \mathcal{C}$ the probability that A and L differ on a random string of length n approaches $\frac{1}{2}$. (That is, the sequence $|\{x \in \Sigma^n : x \in A \Delta L\}|/2^n$ approaches $\frac{1}{2}$.) Very tight Church-randomness hierarchies are known; for all functions t and T such that $\mathrm{DTIME}(t(n))$ is known to be properly contained in $\mathrm{DTIME}(T(n))$, it is known that there is a set in $\mathrm{DTIME}(T(n))$ that is Church-random with respect to $\mathrm{DTIME}(t(n))$ [Wi83]. Also, it is known that the Parity language is Church-random with respect to AC^0 [Cai89, Bab87, Hås87]. (AC^0 is the class of languages accepted by families of constant-depth, polynomial-size circuits of

unbounded fan-in AND and OR gates.)

However, even languages that are Church-random may have infinitely many "easy" special-case inputs. For example, any string of the form 0^n is trivially not in the Parity language, and any string of the form 0^n1 is trivially in the language. A language is *almost-everywhere complex* if there are no infinite classes of "special-case" inputs of this form. More formally, a set L is said to be almost-everywhere complex with respect to DTIME($T(n)$) if any machine recognizing L must run for more than $T(|x|)$ steps on *all* large inputs x. As with Church-randomness, there are very tight hierarchies for almost-everywhere complexity [GHS91]. However, unlike the case of Church-randomness, little is known about sets that are almost-everywhere complex with respect to AC^0. Indeed, since the size and depth of a circuit is the same on all inputs of a given length, it is not at all clear what it should mean for a set to be almost-everywhere complex with respect to a circuit complexity class such as AC^0. For guidance, we turn to the better-understood notions of time and space complexity.

Almost-everywhere complexity has been studied in a variety of settings dealing with time and space complexity [BS85, GHS91, GK90, ABHH90], and in all instances it has been shown to be intimately connected with immunity. (An infinite set L is *immune* to a class $\mathcal{C}$ if L has no infinite subset in $\mathcal{C}$.) For example, in [BS85] it is shown that for any time-constructible function T, a set L is almost-everywhere complex with respect to DTIME($T(n)$) if and only if both L and its complement are immune to DTIME($T(n)$). Thus, a study of sets that are immune to a complexity class is necessary in order to study the notion of almost-everywhere complex sets.

Because of these and other considerations, immunity has often been studied in complexity theory. (For example, see [Li90] and the papers cited there.) In the literature, a class $\mathcal{D}$ is said to be *strongly separated from* a class $\mathcal{C}$ if there is a set in $\mathcal{D}$ that is immune to $\mathcal{C}$. The goal of this work is to study the immunity properties of circuit complexity classes, beginning with the best understood and smallest interesting class; we study strong separations from AC^0.

1.2 Uniform AC^0

No set is immune to non-uniform AC^0, since every infinite set has an infinite sparse subset, and every sparse set is trivially in non-uniform AC^0. Thus in order for there to be a meaningful study of sets immune to AC^0, there must be some agreement on a notion of uniformity for the circuit families under consideration. Since uniformity conditions are necessary in order to draw important connections between circuit complexity and machine-based complexity, the issue of uniformity has been addressed often before [Ruz81, BIS90, BCGR90]. In particular, for very small complexity classes such as AC^0, this issue has been addressed in the papers [BIS90] and [BCGR90], and they provide persuasive arguments for the use of Dlogtime uniformity.

One argument in favor of using Dlogtime uniformity is that there are many appealing alternative characterizations of Dlogtime-uniform AC^0; we will work exclusively with these alternative characterizations, and thus we need not define and discuss the uniformity condition in detail. Here is a list of some of these alternative characterizations:

Theorem 1 [BIS90] The following classes of sets are equal:

- The class of sets accepted by Dlogtime-uniform circuits of AND and OR gates of unbounded fan-in, constant depth and polynomial size.
- The class of sets accepted in $O(1)$ time on a CRAM with polynomially many processors.
- The class of sets accepted by alternating Turing machines in $O(\log n)$ time and $O(1)$ alternations (i.e., the logtime hierarchy defined by Sipser in [Sip83]).
- The class of sets definable in first-order logic with additional predicates $<$ (linear order) and BIT (where $BIT(i, j)$ means that bit i of the binary representation of j is 1). This class is sometimes denoted $FO+ < +BIT$.

It has also been shown recently that the logspace-rudimentary reductions defined by Jones [Jon75] are precisely the functions computed by Dlogtime-uniform AC^0 circuits [AG91b]. Additional characterizations of AC^0 may be found in [Clo90].

A secondary reason for considering Dlogtime uniformity in this paper is that our main results point out the significance of finding sets immune to AC^0, even with this most restrictive notion of uniformity. If less restrictive uniformity is used, then immunity will be correspondingly more difficult to establish.

Throughout this paper, all references to circuit complexity classes assume the Dlogtime uniformity condition (e.g., "AC^0" denotes Dlogtime-uniform AC^0) unless we explicitly specify otherwise (as for example in "P-uniform AC^0" or "non-uniform AC^0").

1.3 Outline of the paper

In Section 2, we exhibit a few complexity classes that have sets that are immune to AC^0. We also prove a somewhat stronger result in Theorem 4 about immunity with respect to ACC, which is a larger complexity class than AC^0. Theorem 4 states:

- There is a set in P^{PP} that is immune to ACC.

In Section 3 we talk about the possibility of finding sets in NP that are immune to AC^0. In Proposition 6 we show the following:

- If there is a set in E that is immune to the class of rudimentary sets then there is a set in P that is immune to AC^0.

We also present an oracle relative to which the hypothesis in the above proposition holds.

In Sections 3.1 to 3.3, we show how any proof that shows that NP contains sets that are immune to AC^0 gives rise to non-relativizable proof techniques for attacking questions about deterministic and nondeterministic exponential time. In Section 3.1 we mention some connections between the E-solvability of NE predicates and a version of time bounded Kolmogorov complexity of sets. In Section 3.2 we relate the E-solvability of NE predicates to the nonexistence of AC^0-immune sets in NP (Theorem 8), and present a few other results that build up to Corollary 10 which is as follows:

- If E is equal to the class of rudimentary sets and all NE predicates are E-solvable, then no set in NP is immune to AC^0.

In Section 3.3 we construct an oracle relative to which the hypothesis of Corollary 10 holds. This construction makes use of the results mentioned in Section 3.1.

In Section 4 we present the conclusion and analyze the results in the paper.

2 Immunity to AC^0

It is easy to find sets with small space complexity that are immune to AC^0. Merely note that AC^0 is contained inside DSPACE($\log n$), and then use the almost-everywhere complexity hierarchy of [GHS91] to obtain the following result:

Proposition 2 Let S be a space-constructible function such that $\log n = o(S(n))$. Then there is a set in DSPACE($S(n)$) that is immune to AC^0.

An essentially identical proof yields the following:

Proposition 3 Let T be a time-constructible function such that $n^k = o(T(n))$ for all k. Then there is a set in DTIME($T(n)$) that is immune to AC^0.

Neither of the preceding two propositions makes any use of any properties of AC^0 at all, other than the trivial observation that AC^0 is contained in DSPACE($\log n$). The results that follow, on the other hand, do rely on the special characteristics of constant-depth circuits. Although our focus in this paper is on AC^0, the immunity results that follow hold also for the slightly larger complexity class ACC, which consists of those languages accepted by Dlogtime-uniform families of polynomial-size, constant-depth circuits of AND, OR, and MOD_m gates, for a fixed modulus m that does not depend on the input length. For more definitions and details concerning ACC, see [Yao90, BT91].

Theorem 4 There is a set in P^{PP} that is immune to ACC (and hence is also immune to AC^0).

Proof: Building on the work of [Yao90, BT91], it is shown in [AG92] that there is a set $Y \in \text{PP}$ such that $ACC \subseteq \text{DTIME}^Y(n^2)$. Now observe merely that the almost-

everywhere hierarchy of [GHS91] relativizes, so that there is a set in $\mathrm{DTIME}^Y(n^3)$ that is immune to $\mathrm{DTIME}^Y(n^2)$, and thus is also immune to ACC. ∎

In [AG92], the containment $ACC \subseteq \mathrm{DTIME}^Y(n^2)$ is used to show that ACC is not equal to PP; that is, there is a set in PP that is not in ACC. However it remains an open question if there is any set in PP that is immune to ACC. In fact it is not known if there are sets in $\mathrm{PP}^{\mathrm{PH}}$ that are immune to AC^0 (where PH denotes the polynomial hierarchy).

3 Immunity within NP

The immunity results presented in the previous section seem initially to be rather disappointing, since AC^0 is a tiny complexity class, unable even to compute the parity of an input string, whereas P^{PP} is a very powerful complexity class, containing the entire polynomial hierarchy. Since $AC^0 \neq NC^1$, it would seem reasonable to hope to be able to present sets in NC^1, or at least in P, that are immune to AC^0.

The results of this section show that such hopes may not be reasonable, after all. Even for the fairly "large" complexity class NP, the question of whether there are sets in the class that are immune to AC^0 cannot be answered without resolving some long-standing issues in complexity theory. In order to make this precise, the first step is to draw some elementary connections between AC^0 and the class of rudimentary sets.

Definition: The class of *rudimentary* sets is the class of languages accepted by alternating Turing machines making $O(1)$ alternations, and running for linear time. We will usually denote this class by $\bigcup_k \Sigma_k\mathrm{Time}(n)$.

The rudimentary sets were originally defined by Smullyan [Smu61], and they have been studied extensively (see, e.g., [Wra78, Lip78, Boo78, Vol83, PD80]). One result of [Wra78] shows that the rudimentary sets can also be characterized in terms of linear-time nondeterministic oracle Turing machines, in analogy to the usual definition of the polynomial-time hierarchy. We will have reason to be interested in the relationships among the rudimentary sets and deterministic and nondeterministic exponential time. Following standard practice, we use E to denote $\mathrm{DTIME}(2^{O(n)})$ and NE to denote

NTIME($2^{O(n)}$).

Proposition 5 $L \in \bigcup_k \Sigma_k\text{Time}(n) \Leftrightarrow un(L) \in AC^0$, where $un(L) = \{0^n : n \in L\}$.

Proof: (This result is proved using elementary translational (i.e, "padding") techniques. The proof is somewhat easier if one observes (as is done in [Sip83]) that, if L is accepted by an alternating Turing machine M in time $O(\log n)$ with $O(1)$ alternations, then L is accepted by a similar machine that, along any computation path, uses its address tape to access the input only once.)

($\Rightarrow$) Let $L \in \bigcup_k \Sigma_k\text{Time}(n)$. Therefore, there exists an alternating Turing Machine M and some k such that M accepts L in linear time and makes k alternations. Consider the machine M' that behaves as follows:

On input 0^n,
    Compute n (in binary)
    Check that M accepts n (by simulating M on n)
end.

It is obvious that M' accepts $un(L)$ in time $O(\log n)$ and makes at most k alternations. Therefore, $un(L) \in \bigcup_k \Sigma_k\text{Time}(\log n)$.

($\Leftarrow$) Suppose there exists an alternating Turing machine M that accepts $un(L)$ in $O(\log n)$ time and makes a constant number of alternations. Consider the machine M' that behaves as follows:

On input n, M' starts simulating M. If, at some point during this simulation, M queries the input via its input address tape, M' compares the number m stored on the address tape of M with the number n, and continues the simulation of M either from the state saying "the m-th input symbol is 0" (if $m \leq n$), or from the state saying "the input has length shorter than m" (if $m > n$); thus M' on input n effectively simulates M on input 0^n. Also, since we are assuming without loss of generality that M uses its random access only once along any computation path, the running time of M is $O(\log n)$, which

is $O(|n|)$. Thus M accepts L in linear time and makes a constant number of alternations. ■

Proposition 5 yields the following immediate corollary.

Proposition 6 If there is a set in E that is immune to $\bigcup_k \Sigma_k\text{Time}(n)$ then there is a set in P that is immune to AC^0.

Proof: If L is in E and is immune to $\bigcup_k \Sigma_k\text{Time}(n)$, then $un(L)$ is in P and is immune to AC^0. ■

Note that if A is any oracle relative to which P=NP, then it follows easily that relative to A, E contains a set that is immune to $\bigcup_k \Sigma_k\text{Time}(n)$. Thus Proposition 6 shows that *if* it can be shown that NP contains no sets that are immune to AC^0, then this yields a non-relativizing proof showing that no set in E is immune to $\bigcup_k \Sigma_k\text{Time}(n)$.

More significantly and surprisingly, non-relativizing proof techniques also result from any proof that NP *does* contain sets that are immune to AC^0. The rest of this section is devoted to making this precise. First, we present some necessary definitions.

3.1 NE Predicates and Kolmogorov Complexity

The P=NP question is usually formulated as a question of the complexity of *recognizing languages*, but of course it is equivalent to the question of *finding witnesses* (e.g., finding a satisfying assignment, instead of merely reporting that a satisfying assignment exists). One can draw a similar distinction between "language recognition" and "witness finding" for other complexity classes – but it turns out that the questions are not always clearly equivalent.

For example, the E=NE question is the natural exponential-time analog of the P=NP question, using the "language recognition" framework. The related "witness-finding" question: "Is every NE predicate solvable in exponential time?" was initially studied in [AW90]. In [IT89] it was shown that there is an oracle relative to which E=NE but not all NE predicates are solvable in exponential time. Thus at least in some relativized worlds, assuming that all NE predicates are solvable in exponential time is strictly stronger than merely assuming E=NE.

We will show that the question of whether or not all NE predicates are solvable in exponential time is related to questions about AC^0. In order to present these relationships, we will need to make use of some connections between the complexity of NE predicates and a version of time-bounded Kolmogorov complexity.

The following paragraphs introduce the notion of time bounded Kolmogorov complexity that we use. The definitions below were introduced in [Lev84, Al89]; more formal definitions and background may be found there and in [Al92]. Since it will sometimes be necessary to speak about Kolmogorov complexity relative to an oracle, we present the definitions relative to an arbitrary oracle A.

Definition: $Kt^A(x) = \min\{|y| + \log t : M_u^A(y) = x \text{ in at most } t \text{ steps}\}$ where M_u is a "universal" Turing machine. Let $L \subseteq \{0,1\}^*$. Then $K_L^A(n) = \min\{Kt^A(x) : x \in L^{=n}\}$.

Of course, we omit the superscript A in the unrelativized case, so that $Kt(x)$ and $K_L(n)$ denote $Kt^\emptyset(x)$ and $K_L^\emptyset(n)$, respectively.

It turns out that questions about the difficulty of NE-predicates can actually be expressed as questions about the K_L complexity of sets L in P. This is made precise in Proposition 7.

Definition: Let $\langle i, x \rangle$ denote a standard one-one pairing function mapping $\mathbf{N} \times \Sigma^*$ into $\mathbf{N}$ (e.g., let $\langle i, x \rangle$ denote the number whose binary representation is the ASCII code of the string "i, x", where i is written in binary). Let $M_1, M_2, \ldots$ be an enumeration of nondeterministic exponential-time oracle Turing machines; note that without loss of generality we may assume that for every NE predicate there is a machine M_i realizing the predicate, such that on every input x, the computations of M_i on input x have length exactly $\langle i, x \rangle$. For any set A, define $S(A)$ to be the set:

$\{y : |y| = \langle i, x \rangle$ and y encodes an accepting path of the NE machine M_i^A on input $x\}$.

Proposition 7 For any oracle A, the following are equivalent:

(a) Every NE^A-predicate is E^A-solvable.

(b) For every set L in P^A, $K_L^A(n) = O(\log n)$.

(c) $K_{S(A)}^A(n) = O(\log n)$.

Proof: The implication (a) $\Rightarrow$(b) is proved (in the unrelativized case) as Theorem 6 in [Al92] (see also [AW90, Theorem 4]). Since $S(A)$ is clearly in P^A, it follows that (b) implies (c). For the remaining implication, suppose $K_{S(A)}^A(n) \leq c \log n$ for some constant c. To solve the NE^A-predicate defined by the NE^A machine M_i^A, we can use the following algorithm:

On input x, let $m = \langle i, x \rangle$. Since $K_{S(A)}^A(m) \leq c \log m$, the universal Turing machine M will produce a string of $S(A)$ of length m (if one exists) on some input of length $\leq c \log m$ with oracle A. Try running M^A on all possible inputs of length $\leq c \log m$ to see if a string y of length m is ever produced. If so, check if y encodes an accepting path of M_i^A on x. It is rather easy to see that the above computation requires $2^{O(|x|)}$ time. ∎

3.2 P-uniform AC^0

We can now prove our results showing connections between NE predicates and the existence of sets in NP that are immune to AC^0.

Theorem 8 If all NE predicates are solvable in exponential time, then no set in NP is immune to P-uniform AC^0.

Proof: It is shown in [AW90, Theorem 4] that if all NE predicates are solvable in exponential time, then every infinite set in P has an infinite P-printable subset.[1] By Theorem 7 of [AR88], this implies that every infinite set in NP has an infinite P-printable subset. But clearly every P-printable set is in P-uniform AC^0; on input n one can in polynomial time form a list of all strings in L of length n, and then build a trivial constant-depth circuit accepting precisely the strings that appear on that list. ∎

Theorem 8 does not say anything about Dlogtime-uniform AC^0, speaking instead only of P-uniform AC^0. The next theorem shows under what conditions these classes coincide.

Theorem 9 $\mathrm{E} = \bigcup_k \Sigma_k\mathrm{Time}(n)$ iff P-uniform AC^0 = Dlogtime-uniform AC^0.

Proof: ($\Leftarrow$) Suppose P-uniform AC^0 = Dlogtime-uniform AC^0. It suffices to show that $\mathrm{E} \subseteq \bigcup_k \Sigma_k\mathrm{Time}(n)$. Let $L \in \mathrm{E}$. Note that $un(L)$ is a tally set in P. Any tally set in P is trivially in P-uniform AC^0. Using the hypothesis, $un(L) \in$ Dlogtime-uniform AC^0. Hence, by Proposition 5, $L \in \bigcup_k \Sigma_k\mathrm{Time}(n)$.

[1] A set L is *P-printable* if there is a polynomial-time routine that, on input n, outputs a list of all strings in L of length n. Note that all P-printable sets are sparse.

($\Rightarrow$) To prove this direction, we first need the simple fact that for all k, the language

$$L_k = \{\langle x, C\rangle : C \text{ is a depth } k \text{ circuit of AND and OR gates that evaluates to 1 on input } x\}$$

is in AC^0. To see this, note that $\langle x, C\rangle \in L_2$ iff

($\exists i$) bit position i in C is the start of the name of the output gate g and

[g is an AND gate and
($\forall j$)
[(if j is the start of the name of an AND gate h connected to g, then
($\forall k$) (if (the negation of) bit k of x is an input to h,
then bit k is 1 (0)))
and (if j is the start of the name of an OR gate h connected to g, then
($\exists k$) ((the negation of) bit k of x is an input to h
and bit k is set to 1 (0)))]]
or [g is an OR gate and
($\exists j$)
[(j is the start of the name of an AND gate h connected to g, and
($\forall k$) (if (the negation of) input bit k is an input to h,
then bit k is set to 1 (0)))
or (j is the start of the name of an OR gate h connected to g, and
($\exists k$) ((the negation of) bit k of x is an input to h
and bit k is set to 1 (0)))]].

Using the characterization of AC^0 in terms of first-order logic as presented by [BIS90], it is clear that L_2 is in AC^0. It is easy to see how to generalize this to show that each L_k is in AC^0.

To proceed, assume that $\mathrm{E} = \bigcup_k \Sigma_k\mathrm{Time}(n)$, and let L be a language in P-uniform AC^0. Thus there is a family of depth k circuits $\{C_n\}$ recognizing L, such that the set $A = \{0^{\langle n,i\rangle} : \text{bit } i \text{ of the description of } C_n \text{ is } 1\}$ is in P. Under the assumption $\mathrm{E} = \bigcup_k \Sigma_k\mathrm{Time}(n)$, it follows from Proposition 5 that A is in Dlogtime-uniform AC^0.

An alternating log-time Turing machine for L can now be described. On input x, simulate the AC^0 algorithm for L_k on input $\langle x, C_n\rangle$. That is, when the algorithm for L_k

uses its address tape to look at input position m, look at the m-th bit of x if $m \leq |x|$, and otherwise test if $0^{\langle n,m-n\rangle} \in A$. ■

Theorems 9 and 8 yield immediately the following corollary.

Corollary 10 If $\mathrm{E} = \bigcup_k \Sigma_k\mathrm{Time}(n)$ and all NE predicates are solvable in exponential time, then no set in NP is immune to AC^0.

We conjecture that the hypothesis to Corollary 10 is false; note that if $\mathrm{E} = \bigcup_k \Sigma_k\mathrm{Time}(n)$, it follows that alternating linear time is equal to $\Sigma_k\mathrm{Time}(n)$ for some k, which puts very low limits on the power of deterministic exponential time. Thus it may seem contradictory to assume simultaneously that deterministic time $2^{O(n)}$ is powerful enough to solve any NE predicate. Note also that if $\mathrm{E} = \bigcup_k \Sigma_k\mathrm{Time}(n)$, then the polynomial hierarchy collapses. Nonetheless, the following theorem shows that the hypothesis to Corollary 10 cannot be shown to be false by any relativizing proof technique.

3.3 The Oracle Construction

Theorem 11 There is an oracle relative to which every NE predicate is solvable in exponential time and $\mathrm{E} = \Sigma_2\mathrm{Time}(n)$.

Before proving Theorem 11, let us remind the reader that we have only one reason for being interested in this unusual combination of conditions on the complexity of exponential time: it implies that no set in NP is immune to AC^0. Hence, finding sets in P or in NP that are almost-everywhere complex with respect to AC^0 will necessarily involve the development of nonrelativizing proof techniques for attacking questions concerning deterministic and nondeterministic time classes.

Proof: To prove Theorem 11, it suffices to build an oracle A such that $K^A_{S(A)}(n) = O(\log n)$ and $\mathrm{E}^A = \Sigma_2\mathrm{Time}^A(n)$.

Notice first that there is a deterministic oracle machine M_S running in time n^2 such that for every A, M^A_S accepts $S(A)$.

Let $L(A) = \{i\#x10^k : \text{the } i^{th}\ \mathrm{E}^A \text{ machine } M^A_i \text{ accepts } x \text{ in } \leq 2^k \text{ steps}\}$. It is easy to see that there is a deterministic oracle machine M^A_L running in time 2^{2n} such that for every A, M^A_L accepts $L(A)$. It is also obvious that $L(A)$ is complete for E^A under linear time reductions.

Thus to prove Theorem 11, it suffices to build an oracle A such that $K^A_{S(A)}(n) = O(\log n)$ and $L(A) \in \Sigma_2\mathrm{Time}(n)$. That is, there are two kinds of encoding that need to be performed. The encoding done to ensure $\mathrm{E}^A = \Sigma_2\mathrm{Time}^A(n)$ will henceforth be called L-encoding and the encoding done to ensure $K^A_{S(A)}(n) = O(\log n)$ will be referred to as S-encoding.

The oracle will be constructed in stages. In each stage we do some L-encoding followed by some S-encoding. These encodings reserve certain strings for A and $\overline{A}$. Note that once a string has been reserved for A or $\overline{A}$, its fate does not change later. Let A_i and $\overline{A_i}$ denote the sets of strings reserved for the oracle and its complement respectively after the completion of stage i. Define $A = \cup_i A_i$.

In a particular L-encoding phase, we ensure that

$$(\forall x)\,(x \in L(A) \Leftrightarrow (\exists y)(\forall z)\,1xyz \in A) \text{ where } |y| = |z| = 5|x|.$$

This is done for all x of a particular length under consideration in that phase.

More precisely, let A'_i denote the oracle that has been constructed up to a certain point in the L-encoding phase of stage i, and let x be the next string under consideration. Run

$M_L^{A_i'}$ on input x, and reserve all of the strings that were queried on that computation for A_i or $\overline{A_i}$, according to whether or not they are in A_i'. If $M_L^{A_i'}$ accepts, find some y of length $5|x|$ such that no string of the form $1xyz$ where $|z| = 5|x|$ is reserved for $\overline{A_i}$, and put all of the $2^{5|x|}$ strings of the form $1xyz$ into A_i. On the other hand, if $M_L^{A_i'}$ rejects, then for each y of length $5|x|$ find some z with $|z| = |y|$ and reserve $1xyz$ for $\overline{A_i}$.

In the S-encoding phase, we try to ensure that $K^A_{S(A)}(m) = O(\log m)$ for the length m currently under consideration by encoding a string $w \in S(A)$ by the strings $0^{m^2+1}, 0^{m^2+2}$, $0^{m^2+3}, \ldots, 0^{m^2+m}$. The way this encoding works is that $(\forall j)\, 1 \leq j \leq m$, the answer to the question "$0^{m^2+j} \in A$?" specifies the j^{th} bit of w. Note that any string w of length m encoded in this way can be reconstructed very quickly from the description "m" (of length logarithmic in $|w|$) and thus $Kt^A(w) = O(\log |w|)$.

More precisely, in considering strings of length m, there are two possibilities (again, let A_i' denote the oracle that has been constructed this far in stage i):

1. $M_S^{A_i'}$ accepts a string w of length m.
2. $M_S^{A_i'}$ does not accept any string of length m.

In the first case, we simply go ahead and encode one such w as described above. In the second case, try to determine if there is a possible extension[2] of A_i' and $\overline{A_i'}$, say A_i'' and $\overline{A_i''}$, so that $M_S^{A_i''}$ accepts some string w of length m. There are two possibilities again:

- $M_S^{A_i''}$ does not accept any string of length m for any possible extension A_i''; in this case, no encoding is required.

[2]Note that any extension that is considered is not allowed to change the status of any string that has already been reserved for A_i' or $\overline{A_i'}$.

- $M_S^{A_i''}$ does accept a string w of length m for some extension A_i''; in this case, extend A_i' to A_i'' and proceed with the encoding of w. (Note that A_i'' need differ from A_i' on at most m^2 strings, since only those strings queried by M_S on input w need be reserved.)

Let $A_0 = \emptyset$, and assume that no strings have been reserved so far. The construction is done as follows:

During stage i,
    do the L-encoding for length i;
    do the S-encoding for lengths m where $2^i \leq m < 2^{i+1}$
end of construction.

Now the only thing left is to prove the correctness of the construction. Consider stage i of the construction:

In the L-encoding phase we want that

$$\text{for all } x \text{ of length } i,\ x \in L(A) \Leftrightarrow (\exists y)(\forall z)\, 1xyz \in A \text{ where } |y| = |z| = 5i$$

Consider the encoding for a typical x of length i. We need to show that there are enough unreserved strings beginning with $1x$ to enable us to do the required encoding. The number of strings that have already been considered for L-encoding is $< (2+2^2+\cdots+2^i) < 2^{i+1}$. For each of these strings, it is possible that M_L^A might have queried the oracle. But since M_L^A runs in time 2^{2n}, the maximum number of strings that could have been reserved due to queries made while processing those strings is $< (2^{i+1} \Leftrightarrow 1) \cdot 2^{2i} < 2^{3i+1}$.

The L-encoding phase also reserves strings for the oracle that are of the form $1uvw$. But since we are only interested in counting strings of the form $1xyz$ for the string x currently under consideration, the above mentioned strings can be ignored.

Note that the S-encoding has been done for lengths $\leq 2^i \Leftrightarrow 1$. The machine M_S^A could possibly have queried strings for all lengths $\leq 2^i \Leftrightarrow 1$. Since M_S^A runs in time n^2, the maximum number of strings that could have been reserved due to queries made while doing that S-encoding is $< \sum_{m=1}^{2^i-1} m^2 < 2^{3i}$. In each S-encoding phase, the strings that are specially reserved to encode strings always start with 0 so they can be ignored too. Therefore, the total number of strings with 1 in the first position that could have been queried or reserved so far is $< 2^{3i+1} + 2^{3i} < 2^{5i}$ $\forall\, i \geq 1$. Since less than 2^{5i} strings have been reserved we can find a string y of length $5i$ such that for all z of length $5i$, none of $1xyz$ have been reserved. Similarly, for each string y of length $5i$ there is some z of length $5i$ such that $1xyz$ is not reserved. Thus it is possible to find the unreserved strings that are required in order to carry out the L-encoding.

Now consider the S-encoding phase. During stage i we do the S-encoding for each length m such that $2^i \leq m < 2^{i+1}$. Encoding is necessary only if there is some w of such a length m such that M_S accepts w with the oracle constructed so far.

Since $|w| = m$, it should be encoded by strings 0^{m^2+j} where $1 \leq j \leq m$. But we must make sure that none of these strings have been reserved as yet.

Since $i \leq \log m$, the L-encoding has only been done for lengths $\leq \log m$. Therefore the longest string with a 0 in the first position that could have been queried during any of the L-encoding phases has length $\leq 2^{2\log m} = m^2$. The strings of the form $1xyz$ that are reserved during the L-encoding phases start with a 1 in their first positions so they do not interfere with the encoding in the current phase. S-encoding has been done for lengths $\leq m \Leftrightarrow 1$, so the longest strings queried or reserved have length $\leq (m \Leftrightarrow 1)^2 + (m \Leftrightarrow 1) < m^2$ whenever $m \geq 1$.

During the current phase, to make $M_S^{A_i'}$ accept an input of length m (by possibly extending the oracle), the longest string that could have been queried and reserved has length $\leq m^2$ (because the machine $M^{A_i'}$ runs in time n^2). Therefore, any string queried so far has length $\leq m^2$. Hence we can encode w with 0^{m^2+j} $1 \leq j \leq m$. This completes the proof of correctness of the construction and we have an oracle $A = \cup_i A_i$ such that $\mathrm{E}^A = \Sigma_2\mathrm{Time}^A(n)$ and $K^A_{S(A)}(n) = O(\log n)$. ∎

4 Conclusion and Analysis

We have begun an investigation into the immunity properties of AC^0. In addition to observing that there are sets in superlogarithmic deterministic space classes and superpolynomial deterministic time classes, we have also shown that there are sets in P^{PP} that are immune to AC^0 (Theorem 4).

We also showed that it would represent a significant advance if one were able to prove the existence (or nonexistence) of sets in NP that are immune to AC^0. More specifically, we presented the following pair of results:

- If $\mathrm{E} = \bigcup_k \Sigma_k\mathrm{Time}(n)$ and every NE predicate is solvable in exponential time, then no set in NP is immune to AC^0. Furthermore, there is an oracle relative to which this hypothesis holds.
- If there is a set in E that is immune to $\bigcup_k \Sigma_k\mathrm{Time}(n)$, then there is a set in P that is immune to AC^0. Furthermore, there is an oracle relative to which this hypothesis holds.

Combining these two results, we see that *any* answer to the question

Are there sets in NP that are immune to AC^0?

yields non-relativizing proof techniques for attacking questions concerning deterministic and nondeterministic time-bounded computation.

Earlier work by Impagliazzo and Naor [IN88] showed that many relationships among very small complexity classes (such as Dtime($\log^{O(1)} n$) and Ntime($\log^{O(1)} n$)) cannot be resolved without first answering long-standing questions such as whether P=NP$\cap$coNP. However, this is the first time that questions about AC^0 have been shown to have a bearing on questions about fairly large classes like E and NE.

It is important to stress that our results concerning non-relativizing proof techniques all use the usual notion of relativized deterministic and nondeterministic time. In particular, we do not deal with relativized notions of AC^0. Indeed, it is not clear that it makes much sense to talk about relativized AC^0; for example, if L is any set that is complete for P under AC^0 reductions, then $\mathrm{P}^L = AC^{0L}$, although clearly $\mathrm{P} \neq AC^0$.

Of course, it is now recognized that the mere existence of an oracle, relative to which some condition X fails to hold, does not preclude the existence of an easy proof that X does hold in the unrelativized case. However, the non-relativizing proof techniques developed in [LFKN90, Sha89, BFL90] have not been shown to be relevant for questions concerning deterministic and nondeterministic time classes; new non-relativizing proof techniques are still needed to answer these questions. One (optimistic) way to view the results of this paper is as an indication that the relatively well-understood complexity class AC^0 offers a possible starting place for the development of such techniques.

In [AG92], we have shown that there is a set $Y \in \mathrm{PP}$ such that AC^0 is contained in Dtime$(n^2)^Y$; we used this result to prove our Theorem 4. It is an interesting open question if there is any set Y' in the polynomial hierarchy such that AC^0 can be recognized

in deterministic time n^k relative to Y', for some k. If so, then there are sets in the polynomial hierarchy that are immune to AC^0. However, this would also have significant consequences concerning the complexity of the rudimentary sets.

Proposition 12 If there is a set L in the polynomial hierarchy and a constant k such that $AC^0 \in \mathrm{Dtime}(n^k)^L$, then there exist constants c and l such that $\bigcup_k \Sigma_k\mathrm{Time}(n) \subseteq \Sigma_l\mathrm{Time}(2^{cn})$.

The conclusion of Proposition 12 is a much better inclusion than is known to hold, and it is reasonable to conjecture that this inclusion fails relative to some oracle, but this is not known to be the case.

Let us end on a note of speculation. Recall that $AC^0 = FO+ < +BIT$. One reason it seems unlikely that this class would be contained in $\mathrm{Dtime}(n^k)$ for any fixed k, is that a first-order formula with $k+1$ quantifiers, with variables ranging over the set $\{1, \ldots, n\}$ cannot be evaluated in any straightforward way in time less than n^{k+1}. However it should be noted that the class $FO+ <$ is exactly the star-free regular sets, and thus each language in that class has a linear-time algorithm [MP71] (see also [Lad77]). Some (but clearly not all) of the combinatorial techniques that are used to prove this characterization of $FO+ <$ apply also to the system $FO+ < +BIT$. It seems possible that techniques could be developed in this setting to prove non-relativizing results, using the logic-based characterizations of uniform AC^0.

Acknowledgments

We wish to thank Jack Lutz for interesting conversations. The first author acknowledges stimulating discussions with Steven Homer, Ulrich Hertrampf, Birgit Jenner, Seinosuke Toda, and Gábor Tardos. He thanks also Pierre McKenzie and Denis Thérien for orga-

nizing the 1991 Barbados Workshop on Complexity Theory, where some of this work was done.

References

[Al89] E. Allender. Some consequences of the existence of pseudorandom generators. *Journal of Computer and System Sci.* 39 (1989) 101–124.

[Al92] E. Allender. Applications of time-bounded Kolmogorov complexity in complexity theory. In *Kolmogorov Complexity and Computational Complexity*, Osamu Watanabe, ed., EATCS Monograph Series, Springer-Verlag, 1992.

[ABHH90] E. Allender, R. Beigel, U. Hertrampf, and S. Homer. A note on the almost-everywhere hierarchy for nondeterministic time. In *Proc. 7th Symposium on Theoretical Aspects of Computer Science,* Lecture Notes in Computer Science 415, pages 1–11. Springer-Verlag, 1990.

[AG91a] E. Allender and V. Gore. On strong separations from AC^0. In *Proc. 8th International Conference on Fundamentals of Computation Theory (FCT '91),* Lecture Notes in Computer Science 529, pages 1–15, Springer-Verlag, 1991.

[AG91b] E. Allender and V. Gore. Rudimentary reductions revisited. *Information Processing Letters* 40 (1991) 89–95.

[AG92] E. Allender and V. Gore. A uniform circuit lower bound for the permanent. DIMACS Technical Report 92-30. Preliminary versions of these results may be found in [AG91a].

[AR88] E. Allender and R. Rubinstein. P-printable sets. *SIAM Journal on Computing* 17 (1988) 1193–1202.

[AW90] E. Allender and O. Watanabe. Kolmogorov complexity and degrees of tally sets. *Information and Computation* 86 (1990) 160–178.

[Bab87] L. Babai. Random oracles separate PSPACE from the polynomial time hierarchy. *Information Processing Letters* 26 (1987) 51–53.

[BFL90] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. Computational Complexity 1 (1991) 3–40.

[BS85] J. Balcázar and U. Schöning. Bi-immune sets for complexity classes. *Mathematical Systems Theory* 18 (1985) 1–10.

[BT91] R. Beigel and J. Tarui. On *ACC*. In *Proc. 32nd IEEE Symposium on Foundations of Computer Science*, pages 783–792, 1991.

[BIS90] D. Mix Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences* 41 (1990) 274–306.

[Boo78] Ronald V. Book. Simple representations of certain classes of languages. *JACM* 25 (1978) 23–31.

[BCGR90] S. Buss, S. Cook, A. Gupta, and V. Ramachandran. An optimal parallel algorithm for formula evaluation. Submitted, 1990.

[Cai89] J. Cai. With probability 1, a random oracle separates PSPACE from the polynomial-time hierarchy. *J. Computer and System Science* 38 (1989) 68–85.

[Clo90] P. Clote. Bounded Arithmetic and Computational Complexity. *Proc. 5th Structure in Complexity Theory Conference*, pages 196–199, 1990.

[GHS91] J. Geske, D. Huynh, and J. Seiferas. A note on almost-everywhere-complex sets and separating deterministic time complexity classes. *Information and Computation* 92 (1991) 97–104.

[GK90] J. Geske and D. Kakihara. Almost-everywhere complexity, bi-immunity, and nondeterministic space. In *Advances in Computing and Information – ICCI '90,* Lecture Notes in Computer Science 468, pages 44–51. Springer-Verlag, 1990.

[Hås87] J. Håstad. *Computational limitations for small depth circuits.* PhD thesis, Massachusetts Institute of Technology, 1987.

[IN88] R. Impagliazzo and M. Naor. Decision trees and downward closures. In *Proc. 3rd Structure in Complexity Theory Conference*, pages 29–38, 1988.

[IT89] R. Impagliazzo and G. Tardos. Decision versus search in super-polynomial time. In *Proc. 30th IEEE Symposium on Foundations of Computer Science*, pages 222–227, 1989.

[Jon75] N. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences* 11 (1975) 68–85.

[Lad77] R. Ladner. Application of model theoretic games to discrete linear orders and finite automata. *Information and Computation* 33 (1977) 281–303.

[Lev84] L. Levin. Randomness conservation inequalities; information and independence in mathematical theories. *Information and Control* 61 (1984) 15–37.

[Lip78] R. Lipton. Model theoretic aspects of computational complexity. In *Proc. 19th IEEE Symposium on Foundations of Computer Science*, pages 193–200, 1978.

[Li90] G. Lischke. Impossibilities and possibilities of weak separation between NP and exponential time. In *Proc. 5th Structure in Complexity Theory Conference*, pages 245–253, 1990.

[LFKN90] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. In *Proc. 31st IEEE Symposium on Foundations of Computer Science*, pages 2–10, 1990.

[MP71] R. McNaughton and S. Papert. Counter-Free Automata MIT Press, Cambridge, Mass., 1971.

[PD80] J. Paris and C. Dimitracopoulos. Truth definitions for Δ_0 formulae. In *Logic and Algorithmic, An International Symposum Held in Honour of Ernst Specker, Monographie no. 30 de L'Enseignement Mathématique*, pages 317–329, 1982.

[Ruz81] W. Ruzzo. On uniform circuit complexity. *Journal of Computer and System Sciences* 21 (1981) 365–383.

[Sha89] A. Shamir. IP = PSPACE. In *Proc. 31st IEEE Symposium on Foundations of Computer Science*, pages 11–15, 1990.

[Sip83] M. Sipser. Borel sets and circuit complexity. In *Proc. 15th ACM Symposium on Theory of Computing*, pages 61–69, 1983.

[Smu61] R. Smullyan. Theory of formal systems. In *Annals of Math. Studies* 47. Princeton University Press, 1961.

[Vol83] H. Volger. Rudimentary relations and Turing machines with linear alternation. In *Logic and Machines : Decision Problems and Complexity,* Lecture Notes in Computer Science 171, pages 131–136. Springer-Verlag, 1983.

[Wi83] R. Wilber. Randomness and the density of hard problems. In *Proceedings, 24th IEEE Symposium on Foundations of Computer Science* pages 335–342, 1983.

[Wra78] C. Wrathall. Rudimentary predicates and relative computation. *SIAM Journal on Computing* 7 (1978) 194–209.

[Yao90] A. Yao. On ACC and Threshold Circuits. In *Proc. 31st IEEE Symposium on Foundations of Computer Science*, pages 619–627, 1990.