Reducing the Complexity of Reductions

Manindra Agrawal[*]
Dept. of Computer Science
Indian Institute of Technology
manindra@iitk.ernet.in

Eric Allender[†]
Dept. of Computer Science
Rutgers University
allender@cs.rutgers.edu

Russell Impagliazzo[‡]
Dept. of Computer Science
UCSD
russell@cs.ucsd.edu

Toniann Pitassi[§]
Dept. of Computer Science
University of Arizona
toni@cs.arizona.edu

Steven Rudich
Dept. of Computer Science
Carnegie Mellon University
rudich@cs.cmu.edu

Abstract

We prove that the Berman-Hartmanis isomorphism conjecture is true under AC^0 reductions.

More generally, we show three theorems that hold for any complexity class $\mathcal{C}$ closed under (uniform) TC^0-computable many-one reductions. **Isomorphism:** The sets complete for $\mathcal{C}$ under AC^0 reductions are all isomorphic under isomorphisms computable and invertible by AC^0 circuits of depth three. **Gap:** The sets that are complete for $\mathcal{C}$ under AC^0 and NC^0 reducibility coincide. **Stop Gap:** The sets that are complete for $\mathcal{C}$ under $AC^0[\text{mod } 2]$ and AC^0 reducibility do not coincide. (These theorems hold both in the non-uniform and P-uniform settings.) To prove the second theorem for P-uniform settings, we show how to derandomize a version of the switching lemma, which may be of independent interest. (We have recently learned that this result is originally due to Ajtai and Wigderson, but it has not been published.)

1 Introduction

The notion of *complete sets* in complexity classes provides one of the most useful tools currently available for classifying the complexity of computational problems. Since the mid-1970's, one of the most durable conjectures about the nature of complete sets is the Berman-Hartmanis conjecture [BH77], which states that all sets complete for NP (under polynomial-time many-one reductions) are p-isomorphic; essentially this conjecture states that the complete sets are all merely different encodings of the same set. (Two sets A and B are considered p-isomorphic if there is a bijection on Σ^* mapping A onto B that is polynomial-time computable and polynomial-time invertible.) Although the isomorphism conjecture was originally stated for the NP-complete sets, subsequent work has considered whether the complete sets for other complexity classes $\mathcal{C}$ (and under other notions of reducibility) collapse to an isomorphism type. In this paper, we prove such an analogue of the Berman-Hartmanis conjecture in a very natural setting: all sets complete for NP under AC^0 reductions are isomorphic under AC^0-computable isomorphisms. (The results in this paper hold for P-uniform and non-uniform AC^0 and NC^0 reductions. Throughout the paper, AC^0 and NC^0 refer to P-uniform circuits unless otherwise indicated. Although we do not explicitly refer to non-uniform circuits, it is easy to see that our proofs hold also in that case. Sometimes we also refer to Dlogtime-uniform circuits; in all such cases, note that using the more restrictive Dlogtime-uniformity condition makes our results stronger than if we used a less-restrictive uniformity condition.) Two major ingredients of our proof of this result are: (1) a 'gap' theorem wherein we show that all sets complete for NP under AC^0 reductions are also complete under NC^0 reductions, and (2) derandomization of a version of the switching lemma wherein we show how to construct, in polynomial time, a restriction for any given AC^0 circuit that reduces the circuit to a NC^0 circuit.

In full generality, the three main theorems of this paper can be stated as follows. For any class $\mathcal{C}$ closed under Dlogtime-uniform TC^0-computable many-one reductions:

Isomorphism Theorem: The sets complete for $\mathcal{C}$ under AC^0 reductions are all isomorphic under isomorphisms computable and invertible by depth-three AC^0 circuits.

Gap Theorem: The sets that are complete for $\mathcal{C}$ under AC^0 and NC^0 reducibility coincide.

Stop Gap Theorem: There is a set that is complete for $\mathcal{C}$ under $AC^0[\text{mod } 2]$ reductions, but not under AC^0

[*]Part of this research was done while visiting the University of Ulm under an Alexander von Humboldt Fellowship.

[†]Supported in part by NSF grant CCR-9509603.

[‡]Research Supported by NSF Award CCR-92-570979, Sloan Research Fellowship BR-3311, and USA-Israel BSF Grant

[§]Research supported by NSF Award CCR-94-57782, and USA-Israel BSF Grant

reductions.

The work in this paper builds on [AA96], where it is shown that the Gap Theorem and a weaker form of the Isomorphism Theorem hold for the particular case of $\mathcal{C} = \mathrm{NC}^1$.

1.1 The Isomorphism Theorem

The Berman-Hartmanis conjecture has inspired a great deal of work in complexity theory, and we cannot review all of the previous work here. For an excellent survey, see [KMR90]. We do want to call attention to two general trends this work has taken, regarding (1) one-way functions, and (2) more restrictive reducibilities.

One-way functions (in the worst case sense) are functions that can be computed in polynomial-time, but whose inverse functions are not polynomial-time computable. Beginning with [JY85] (see also [KMR95, Se92, KLD86], among others) many authors have noticed that if worst case one-way functions exist, then the Berman-Hartmanis conjecture might not be true. In particular, if f is one-way, nobody has presented a general technique for constructing a p-isomorphism between SAT and f(SAT), even though f(SAT) is clearly NP-complete.

An even stronger notion than one-way functions is that of *average case* one-way functions: these are polynomial-time computable functions whose inverse can be efficiently computed *only on a negligible fraction* of the range. Advances in the theoretical foundations of cryptography have shown that *average case* one-way functions can be used to construct "pseudo-random" functions that are computable in polynomial-time [HILL90]. Intuitively, if f is a 1-1 polynomial time, random-like function, f(SAT) is NP-complete, but has no apparent structure to facilitate the construction of an isomorphism to SAT. Kurtz, Mahaney, and Royer [KMR95] are able to make this intuition technically precise in the random oracle setting. They show that when f is a truly random function, f(SAT) is not isomorphic to SAT even when f is given as an oracle. This suggests that a pseudo-random f might similarly guarantee that no isomorphism to f(SAT) is possible.

Our Isomorphism Theorem *negates* the above intuition in two important special cases. Firstly, it is easy to observe that *there are worst-case one-way functions in Dlogtime-uniform NC^0* if there are any one-way functions at all [AA96]. Thus, if the worst-case one-way functions are sufficiently easy to compute, the intuition that worst-case one-way functions cause the isomorphism conjecture to fail is incorrect.

Secondly, we prove that all sets complete for NP under AC^0 reductions are complete under reductions that are computable via *depth two* AC^0 circuits, and these sets are all isomorphic to SAT under isomorphisms computable and invertible by AC^0 circuits of *depth three*. But it is known that there are functions computable in AC^0 that are average-case hard to compute for AC^0 circuits of depth three, and that this allows one to produce output that appears pseudorandom to AC^0 circuits of depth three [Ni92]. Using these tools, one can construct one-one functions f, many of whose output bits look random to depth three circuits. Although one might believe that for such a function f, f(SAT) should appear random to AC^0 circuits of depth three, *there is nonetheless a reduction from SAT to f(SAT) computable in depth two, and an isomorphism computable and invertible in depth three.*

1.2 The Gap and Stop Gap Theorems

A curious, often observed fact is that all sets known to be NP-complete under polynomial-time, many-one reductions remain NP-complete under many-one, AC^0 reductions. This is interesting because AC^0 is known to compute a much smaller class of functions than polynomial-time. It was not previously known if perhaps *each* polynomial-time reduction to an NP-complete set could *always* be replaced by an AC^0 reduction. In this paper, we prove that such a gap in the power of reductions does exist between AC^0 and NC^0: any set that is NP-complete under AC^0 reductions remains NP-complete under NC^0 reductions. As is the case for polynomial-time versus AC^0, the difference in computational power between AC^0 and NC^0 is enormous. For example, each output of an NC^0 computable function can depend on only *finitely* many inputs. Thus, NC^0 can't even compute an AND of all its inputs (in contrast, the unbounded fan-in AND is an AC^0 function). Nonetheless, our theorem shows that AC^0 and NC^0 are equivalent from the point of view of performing NP-completeness reductions. It follows that all known NP-complete problems are complete under NC^0 reductions. The fact that in an NC^0 reduction each output bit depends on only finitely many of the input bits means that NC^0 reductions are local and simple by nature. Intuitively, NC^0 reductions correspond to a radically simple form of "gadget reduction." (To understand the result better, the reader may verify that the standard, "simple" reduction from 3SAT to Clique is *not* NC^0-computable.)

The gap between AC^0 and NC^0 reductions does not extend any further since, as we show, there exists a set that is complete for NP under AC^0[mod 2] reductions but not under AC^0 reductions.

2 Basic Definitions and Preliminaries

We assume familiarity with the basic notions of many-one reducibility as presented, for example, in [BDG88]. In this paper, **only many-one reductions will be considered**.

A *circuit family* is a set $\{C_n : n \in \mathbf{N}\}$ where each C_n is an acyclic circuit with n Boolean inputs $x_1, \ldots, x_n$ (as well as the constants 0 and 1 allowed as inputs) and some number of output gates $y_1, \ldots, y_r$. $\{C_n\}$ has *size* $s(n)$ if each circuit C_n has at most $s(n)$ gates; it has *depth* $d(n)$ if the length of the longest path from input to output in C_n is at most $d(n)$. A family $\{C_n\}$ is *uniform* if the function $n \mapsto C_n$ is easy to compute in some sense. In this paper, we will consider only Dlogtime-uniformity [BIS90] and P-uniformity [Al89] (in addition to non-uniform circuit families).

A function f is said to be in AC^0 if there is a circuit family $\{C_n\}$ of size $n^{O(1)}$ and depth $O(1)$ consisting of unbounded fan-in AND and OR and NOT gates such that for each input x of length n, the output of C_n on input x is $f(x)$. Note that, according to this strict definition, a function f in AC^0 must satisfy the restriction that $|x| = |y| \Longrightarrow |f(x)| = |f(y)|$. However, the imposition of this restriction is an unintentional artifact of the circuit-based definition given above, and it has the effect of disallowing any interesting results about the class of sets isomorphic to SAT (or other complete sets), since there could be no AC^0-isomorphism between a set containing only even length strings and a set containing only odd length strings – and it is precisely this sort of indifference to encoding details that motivates much of the study of isomorphisms of complete sets. Thus we allow AC^0-computable functions to consist of functions computed by circuits of this sort, where some simple convention is used to encode inputs of different lengths (for example, "00" denotes zero, "01" denotes one, and "11" denotes end-of-string; other reasonable conventions yield exactly the same class of functions). For technical reasons, we will adopt the following specific convention: each C_n will have $n^k + k\log(n)$ output bits (for some k). The last $k \log n$ output bits will be viewed as a binary number r, and the output produced by the circuit will be the binary string contained in the first r output bits. It is easy to verify that this convention is AC^0-equivalent to the other convention mentioned above, and for us it has the advantage that only $O(\log n)$ output bits are used to encode the length. It is worth noting that, with this definition, the class of Dlogtime-uniform AC^0-computable functions admits many alternative characterizations, including expressibility in first-order with $\{+, \times, \leq\}$ [Li94, BIS90] the logspace-rudimentary reductions of Jones [Jo75, AG91], logarithmic-time alternating Turing machines with $O(1)$ alternations [BIS90] and others. This lends additional weight to our choice of this definition.

TC^0 is the class of functions computed in this way by circuit families of MAJORITY gates of size $n^{O(1)}$ and depth $O(1)$; NC^1 and NC^0 are the classes of functions computed in this way by circuit families of size $n^{O(1)}$ and depth $O(\log n)$ (or $O(1)$, respectively), consisting of fan-in two AND and OR and NOT gates. Note that for any NC^0 circuit family, there is some constant c such that each output bit depends on at most c different input bits. The class of functions in NC^0 was considered previously in [Hå87].

For technical reasons and for simplicity of exposition, we do *not* allow an NC^0 circuit C_n to produce outputs of different lengths for different inputs of length n, although we *do* allow AC^0 and NC^1 circuits to do this by following the convention mentioned above. That is, if f is computed by NC^0 circuit family $\{C_n\}$ where each C_n has $s(n)$ output bits, then for all inputs x of length n, $|f(x)| = s(n)$. Our chief justification for imposing this restriction is that Theorem 5 shows that any set hard for NP (or other complexity classes) under AC^0 reductions (using the less-restrictive convention allowing outputs of different lengths) is in fact hard under NC^0 reductions (using the more-restrictive convention). Thus we are able to obtain our corollaries about sets complete under AC^0 reductions without dealing with the technical complications caused by allowing NC^0 reductions to output strings of different lengths. Also note that, even with this restriction, the NC^0 reductions we consider are still more general than the first-order projections considered in [ABI93].

For a complexity class $\mathcal{C}$, a $\mathcal{C}$-isomorphism is a bijection f such that both f and f^{-1} are in $\mathcal{C}$. Since only many-one reductions are considered in this paper, a "$\mathcal{C}$-reduction" is simply a function in $\mathcal{C}$.

The theorems we prove in this paper hold for most complexity classes that are of interest to theoreticians; we require only closure under certain easily-computable reductions. To make this precise, we say that a class of languages $\mathcal{C}$ is a *proper complexity class* if $\mathcal{C}$ is closed under Dlogtime-uniform TC^0 reductions. (That is, if A is in $\mathcal{C}$, and B is reducible to A via a many-one reduction computable in TC^0, then B is in $\mathcal{C}$.) Note that most complexity classes, such as NP, P, PSPACE, BPP, etc. are proper complexity classes.

In fact, inspection of our proofs shows that our results hold even for any class $\mathcal{C}$ that is closed under reductions computed by Dlogtime-uniform threshold circuits of depth *five*. (The number five can probably be reduced.) We do not know how to weaken the assumption to closure under reductions computed in ACC^0; it is easy to see that our results do *not* hold for some classes closed under AC^0 reductions. (For instance, the sets $\{1\}$ and $\{1, 11\}$ are both hard for AC^0 under AC^0 reductions, but they are not isomorphic, and they are not hard under NC^0 reductions.)

A function is *length-nondecreasing* (*length-increasing*, *length-squaring*) if, for all x, $|x| \leq |f(x)|$ ($|x| < |f(x)|$, $|x|^2 \leq |f(x)|$); it is $\mathcal{C}$-invertible if there is a function $g \in \mathcal{C}$ such that for all x, $f(g(f(x))) = f(x)$.

3 Main Results

3.1 Superprojections

In this section, we show how a more careful application of the main technical contribution of [AA96] leads to improved isomorphism theorems, and isomorphisms computed in depth three. We will need the notion of a "superprojection".

Definition 1 *An NC^0 reduction $\{C_n\}$ is a* superprojection *if the circuit that results by deleting zero or more of the output bits in each C_n is a projection wherein each input bit (or its negation) is mapped to some output. (Stated another way, it is a superprojection if, for each input bit x_i, there is an output bit whose value is completely determined by x_i. That is, this output bit is either x_i or $\neg x_i$.)*

Note that every superprojection has an inverse that is computable in AC^0: On input y, we want to determine if there is an x such that $f(x) = y$. The AC^0 circuit will have a subcircuit for each $n \leq |y|$ (since a superprojection is by definition one-one and length-nondecreasing) checking to see if there is such an x of length n. This subcircuit will find the n output bits that completely determine what x must be (if such an x exists), and then will check to see if $f(x) = y$.

Theorem 1 *[AA96] For every proper complexity class $\mathcal{C}$, every set hard for $\mathcal{C}$ under NC^0 reductions is hard under one-one, length-squaring superprojections.*

The following corollaries improve on related corollaries in [AA96].

Corollary 2 *For every proper complexity class $\mathcal{C}$, every set hard for $\mathcal{C}$ under NC^0 reductions is hard under reductions computable by depth two AC^0 circuits and invertible by depth three AC^0 circuits.*

Proof. First note that since a superprojection is an NC^0 reduction, it can be computed in depth two simply by expressing each output bit in DNF or CNF form.

By referring to the proof of Theorem 1 in [AA96], the reader will find that if A is hard for $\mathcal{C}$ under NC^0 reductions, then it is hard under superprojections f of the form $h \circ g$ where h is a superprojection when restricted to strings in the range of g, and strings in the range of g have the form

$y10^k$. Furthermore, for each n there is an (easily-computed) m such that, for each string x of length n, if $f^{-1}(x)$ exists, then there exist y and k such that $|g(y)| = |y10^k| = m$, $f^{-1}(x) = y$, and $h(y10^k) = x$. (The point here is that m depends only on $n = |x|$.) Now to compute f^{-1} for inputs of length n, it suffices to consider the circuit computing h on inputs of length m. For inputs x of length n, $f^{-1}(x)$ exists if and only if $h^{-1}(x)$ exists and is of the form $y10^k$ for k in the correct range. If $h^{-1}(x)$ exists, then there are m bits of x that directly encode the bits of $y10^k$, since h is a superprojection.

Thus our circuit to compute $f^{-1}(x)$ first takes the string $y10^k$ that is available on the input level of the circuit (as determined by m bits of input x) and that is a candidate for $h^{-1}(x)$. Then (in depth two) it computes $h(y10^k)$, and checks that all of the bits of $h(y10^k)$ and x agree. This is an AND of several NC^0 predicates, and by expressing the NC^0 predicates in CNF and merging the two levels of AND gates we obtain a depth two circuit producing output $y10^k$ if $h^{-1}(x) = y10^k$. Since our goal is to produce output y (and also output $|y|$ in the length-encoding field) we obtain a depth three circuit by taking the OR over all possible values of $r = |y|$ of the predicate "$h^{-1}(x) = y10^k$ AND the last $m - r$ bits of $y10^k$ are in 10^*". ∎

Corollary 3 *For every proper complexity class $\mathcal{C}$, all sets complete for $\mathcal{C}$ under NC^0 reductions are AC^0-isomorphic. Furthermore, these isomorphisms are computable and invertible by AC^0 circuits of depth three.*

Proof.

This result should be contrasted with the main result of [ABI93], where it is shown that all sets complete under "first-order projections" are isomorphic under Dlogtime-uniform AC^0 isomorphisms. Projections are much more limited than superprojections, in the sense that there are sets that are complete under superprojections but not under projections [ABI93], whereas our Gap Theorem, combined with Theorem 1, shows that all sets complete under AC^0 reductions are already complete under superprojections. Most of the work in [ABI93] involves showing that things can be done uniformly, and they do not achieve depth k for any constant k. On the other hand, although we do provide depth three isomorphisms, we do not acheive Dlogtime-uniformity.

Let A and B be complete for $\mathcal{C}$ under NC^0 reductions. Thus there are superprojections f and g reducing A to B and reducing B to A, respectively. Our goal is to construct an isomorphism mapping A onto B. We first construct a depth four isomorphism between A and B, and then improve it to depth three. As in most other work constructing isomorphisms (see [BH77] for example), given an input x, we will need to compute the length of the "ancestor chain" of x, and output $f(x)$ if the length of the chain is even, and output $g^{-1}(x)$ if the length of the chain is odd.

Note that if the k^{th} ancestor exists, then (just as in the case $k = 1$ in the proof of Corollary 2) the bits of the k^{th} ancestor are available at the input level. Thus one can determine in depth three if the length of the ancestor chain is exactly r. (Namely, for all $k < r$ the appropriate inverse image of the k^{th} ancestor exists, and it does not exist for the r^{th} ancestor.) Now, the i^{th} bit of the output would be

$$((\bigvee_{k \text{ odd}} r = k) \bigwedge (i^{th} \text{ bit of } f(x))) \bigvee$$
$$((\bigvee_{k \text{ even}} r = k) \bigwedge (i^{th} \text{ bit of } g^{-1}(x))).$$

This gives a depth six circuit, however, note that the top two levels are of fan-in two, and therefore, can be "pushed down" and collapsed with the bottom two levels. This results in a depth four circuit.

To reduce the depth further, we observe that we do not need to explicitly check for the existence of the inverse at level two (as is done in the proof of Corollary 2). Instead, we distribute this work to the top two levels: Let $C_{k,m}$ be the NC^0 circuit that outputs a sequence of ones iff the k^{th} ancestor exists and has length m. Let C'_k be the depth two AC^0 circuit (with top level AND gates) that outputs a sequence containing at least one zero iff the k^{th} ancestor does *not* exist. (To see how to construct C'_k, note that if the k^{th} ancestor does exist, then there is a $k-1^{th}$ ancestor z of some length m that is completely determined by n and k, and the k^{th} ancestor is a string y where $h^{-1}(z) = y10^r$ where r cannot be too large. That is, in addition to the local consistency checks (each bit of which can computed in NC^0), the only other condition that must be checked is to say that the k^{th} ancestor does *not* exist if $h^{-1}(z) = y10^r$ ends in too many zeros. This can clearly be checked by a CNF circuit.) Let $D_{r,l,\vec{m}}$ be the circuit computing

$$(\bigwedge_{k \leq r} \text{output}(C_{k,m_k}) \in 1^*) \bigwedge (l^{th} \text{ bit of output}(C'_{r+1}) = 0).$$

(Here, $\vec{m}$ is a vector of $O(\log n)$ bits encoding a sequence $m_1, \ldots, m_r$ of numbers such that $(m_i)^2 \leq m_{i+1} \leq n$. Since f and g are both length-squaring, the sequence of lengths occurring in the ancestor chain can be encoded as such a vector. $D_{r,l,\vec{m}}$ is still a depth two circuit since the AND gate at the top can be merged with the AND gates on top of the C_{k,m_k}s and C'_{r+1}s. The i^{th} bit of the output can now be defined as:

$$((\bigvee_{r \text{ odd}} \bigvee_{l} \bigvee_{\vec{m}} D_{r,l,\vec{m}}) \bigwedge (i^{th} \text{ bit of } f(x))) \bigvee$$
$$((\bigvee_{r \text{ even}} \bigvee_{l} \bigvee_{\vec{m}} D_{r,l,\vec{m}}) \bigwedge (i^{th} \text{ bit of } g^{-1}(x))).$$

As argued before, this is a depth three circuit. A similar circuit computes the inverse of the isomorphism. ∎

3.2 The Gap Theorem and The Isomorphism Theorem

3.2.1 Explicit Restrictions of AC^0 reductions

It has been a folklore theorem since [FSS84, Aj83] that a randomly-chosen restriction of the input variables to an AC^0 circuit family results in an NC^0 family with high probability. (See [Ar95], for example.)

For our purposes, it is very useful to know that such a restriction can be constructed quickly deterministically, and that furthermore this restriction does not set too many adjacent variables. The following lemma makes this precise.

Lemma 4 *For any AC^0 reduction computed by a family of circuits $\{C_m\}$, there exists an $a \in \mathbf{N}$ such that, for all large m of the form r^{2a}, there is a restriction τ_m such that τ_m transforms C_m into an NC^0 circuit, and τ_m assigns $*$ to at least three variables in* each *block of length r^{2a-1}. Furthermore, τ_m can be computed in time polynomial in m, if $\{C_m\}$ is P-uniform.*

(A proof of a stronger version of Lemma 4 is presented in the appendix.)

3.2.2 The Gap Theorem

Theorem 5 (Gap Theorem) *Let $\mathcal{C}$ be any proper complexity class. The sets hard for $\mathcal{C}$ under AC^0 reductions are hard for $\mathcal{C}$ under NC^0 reductions.*

(Note that, by Theorem 1, we obtain the stronger conclusion that these sets are actually hard under superprojections.)

Proof. Let $\mathcal{C}$ be any proper complexity class, i.e., $\mathcal{C}$ is closed under Dlogtime-uniform TC^0 reductions. Let A be any set hard for $\mathcal{C}$ under AC^0-reductions. Let B be any set in $\mathcal{C}$. Clearly, B is AC^0-reducible to A. We seek to show that B is actually NC^0-reducible to A.

The proof strategy will be to define a set $B' \in \mathcal{C}$, and use the AC^0-reduction C_n from B' to A as a starting point for a reduction from B to A. B' will have been chosen so that a suitable restriction of C_n will give us an NC^0 reduction from B to A.

We define B' to be the set of strings accepted by the following procedure:

> On input y, let $y = 1^k0z$. Reject if k does not divide $|z|$. Otherwise, break z into blocks of k consecutive bits each. Let these be $u_1u_2u_3\cdots u_q$. For each i, $1 \le i \le q$, let $v_i = 0$ if the number of ones in u_i equals 0 modulo 3; $v_i = 1$ if the number of ones in u_i equals 1 modulo 3; and $v_i = \epsilon$ otherwise. Accept iff $v_1v_2\cdots v_q \in B$.

It is easy to see that B' is Dlogtime-uniform TC^0 reducible to B. Hence, by the definition of proper complexity class, $B' \in \mathcal{C}$. Since A is hard for $\mathcal{C}$ under AC^0-reductions, there must exist an AC^0 circuit family C_n computing a reduction from B' to A. Let d be a bound on the depth of the family C_n. W.l.o.g. we can assume that C_n takes n input bits and has no more than n^d output bits. (Recall that the final $O(\log n)$ bits are used to encode a number that indicates how many of the output bits to use in the reduction. We refer to these bits as the "length-encoder bits".)

Let a be the constant (depending on the depth d of C_n) from Lemma 4. Let C'_m be the family of circuits for inputs of length $m = (2q)^{2a}$, where C'_m is obtained by taking circuit C_n for $n = 1+(2q)^{2a-1}+m$ and setting the first $1+(2q)^{2a-1}$ bits to $1^{(2q)^{2a-1}}0$.

By Corollary 4, for all large m, there is a restriction τ_m such that τ_m transforms C'_m into an NC^0 circuit, and τ_m assigns $*$ to at least three variables in *each* block of length $(2q)^{2a-1}$. We will now show how to extend τ_m to obtain a further restriction of C'_m having only q variables, and having the length-encoder bits set to constant values. We will call this new circuit family D_q. This circuit family D_q will be our NC^0 reduction from B to A.

Each of $O(\log n)$ length encoder bits depends only on a constant number of remaining input bits. Thus, the encoder bits depend only on $O(\log n)$ blocks. For each of these $O(\log n)$ blocks, fix *all* the remaining bits to constants so that the number of 1's in each block is 2 modulo 3 (this is always possible because we have at least 3 unset bits in each block). The length-encoder bits are fixed and we still have $2q - O(\log q)$ blocks that we have not tampered with. Pick all but the first q blocks and also fix their inputs so that the number of 1's in each of them is 2 modulo 3. For each of the remaining q blocks, set all but one bit in each block so that the total number of 1's in the block is 0 modulo 3 (again this is possible since there are at least three unset bits in each block). The result is a circuit with exactly q input bits and a fixed output size. (That is, all of the length-encoder bits have been set to constant values by setting the bits on which they depend. Let this value be r. Thus we can delete the length encoder bits and all but the first r output bits.) Call this circuit family D_q. Notice that it has size polynomial in q because q is $\Omega(n^\epsilon)$ for some $\epsilon > 0$ and C_n is of size polynomial in n. Also note that D_q is obtained from C_n by restricting attention to inputs of the form $y = 1^{(2q)^{2a-1}}0z$, where z is a string with exactly q $*$'s. For any string x of length q, denote by $y(x)$ the result of plugging the q bits of x into the q unset positions in z, and note that the algorithm for B' accepts $y(x)$ if and only $x \in B$ (because the algorithm for B' decodes z to obtain x).

Since C_n reduces B' to A, we see that D_q reduces B to A. This is the desired NC^0-reduction from B to A. ∎

3.2.3 The Isomorphism Theorem

The Isomorphism Theorem is an immediate consequence of the Gap Theorem and Corollary 3.

Theorem 6 (Isomorphism Theorem) *Let $\mathcal{C}$ be any proper complexity class. All sets complete for $\mathcal{C}$ under AC^0 reductions are AC^0-isomorphic. Furthermore the isomorphisms are computable and invertible by depth three AC^0 circuits.*

Note that this is, in some sense, a true analog of the Berman-Hartmanis conjecture, since it presents a natural notion of computation (which then yields natural notions of reducibility and isomorphism) and it shows that in this setting the complete sets coincide with the isomorphism type of the standard complete set.

It is worth mentioning that depth three is optimal (proof is omitted in this abstract):

Theorem 7 *For any class $\mathcal{C}$ closed under AC^0 reductions, the sets that are AC^0-complete for $\mathcal{C}$ are not all isomorphic under depth-two AC^0-computable isomorphisms.*

3.3 Where the gap stops.

In light of theorem 5, it is natural to ask how large a gap in the power of reducibilities is possible. In this regard, note that some non-gap theorems for uniform notions of reducibility were already known. For instance, it is shown in [AA96] that the P-uniformity condition in Theorem 5 cannot be replaced by Dlogtime-uniformity, and it is shown in [Ag95] that if DSPACE(n) is not equal to E, then there is a set complete for PSPACE under poly-time reductions but not under logspace reductions. Nonetheless, it is consistent with what was previously known that nonuniform NC^0 reductions and P/poly reductions have the same power! However, our next theorem shows that our gap theorem is essentially optimal.

Theorem 8 (Stop Gap Theorem) *There is a set that is NP-complete under (NC^1 uniform) AC^0[mod 2] reductions, but not under (non-uniform) AC^0 reductions.*

(Remark: In the statement of this theorem, "NP" can be replaced by any other proper complexity class closed under uniform NC^1 reductions.)

Proof. Let SAT be the set of satisfiable boolean formulas encoded as binary strings in any natural way. SAT is NP-complete. Let PARITY be the set of all binary strings with an even number of ones. PARITY is clearly in NP.

Let f be a function from n-bit binary strings to $n^{O(1)}$-bit binary strings that computes an error correcting code capable of correcting a constant fraction of errors. I.e., for any two distinct n-bit strings x and y, $f(x)$ and $f(y)$ differ in at least a constant fraction δ of positions. Justesen [J72], provides a particular construction of such an f (for some $\delta > 0$) that is also very easy to compute. Each output bit of $f(x)$ is simply the parity of some input bits of x, and hence the Justesen code f is computable in AC^0[mod 2]. Also, there is a uniform NC^1 circuit that, on input 1^n, produces the AC^0[mod 2] circuit for inputs of size n, and thus f is in NC^1-uniform AC^0[mod 2].

Let $S = \{\ f(x) :\ x \in \text{ SAT }\}$. S is clearly NP-complete under AC^0[mod 2] reductions. Suppose, for a contradiction, that S is NP-complete under AC^0 reductions. In particular, there must be a AC^0 reduction from PARITY to S. Thus, there is an AC^0 circuit C_n with n inputs and $m = n^{O(1)}$ outputs that reduces an instance of PARITY to an instance of S.

From lemma 4, we know that there is a restriction of C_n that leaves $O(n^\epsilon)$ variables unset and transforms C_n into an NC^0 circuit where each output bit depends on at most b input bits for some constant b. Let C_n have m output bits. Note that there are no more than $2b/\delta = O(1)$ unset input variables that influence $\geq \delta m/2$ output bits. Thus we can set $O(1)$ additional input variables and obtain an NC^0 circuit family on n' input variables that also reduces PARITY to S, and has the property that no input variable influences more than $\delta m/2$ output bits. Call this new circuit family $D_{n'}$, and let g be the function computed by the family D.

Consider two strings x_1, x_2 of length n' that are in PARITY and that differ in exactly two locations i and j. We claim that $g(x_1) = g(x_2)$. Otherwise $g(x_1)$ and $g(x_2)$ differ in at least δm locations (since they map to two distinct codewords in f(SAT)), and these δm locations are influenced by variables i and j, in contradiction to the construction of $D_{n'}$. Since any string x of length n' in PARITY can be obtained from $0^{n'}$ by a sequence $0^{n'} = x_1, x_2, \ldots, x_r = x$ where x_i and x_{i+1} differ in exactly two locations, it follows that the strings of length n' in PARITY can be characterized as the set $\{x \mid g(x) = g(0^{n'})\}$, and thus PARITY is in AC^0, which is a contradiction. ∎

Stating what we have proved in full generality, we get:

Theorem 9 *Let $\mathcal{C}$ be any complexity class that is closed under AC^0[mod 2] reductions and that has a set that is complete under $\mathcal{R}$ reductions. It follows that $\mathcal{C}$ contains a set that is complete under $\mathcal{R} \cup AC^0$[mod 2] reductions, but not under AC^0 reductions.*

Although the Stop-Gap Theorem shows that not all sets complete under AC^0[mod 2] reductions are AC^0-isomorphic, it is natural to wonder if they are all AC^0[mod 2]-isomorphic, or if there is some other sort of Gap Theorem that still awaits discovery. In this regard, it is worth noting that the sets constructed in the proof of the Stop Gap Theorem are, in fact, all AC^0[mod 2]-isomorphic to SAT. (*Sketch of proof:* The sets we construct are all complete under reductions computable by depth 1 circuits consisting entirely of parity gates. Reductions of this sort are trivial to invert; If the string y is given, and we want to see if there is an x such that $f(x) = y$, then the conditions on the x_i form a system of linear equations in the y_j, and in fact each x_i is the parity of some subset of the y_j. Thus we simply find what the x_i would have to be if they map to y, and then do a few consistency checks. At this point the techniques of [AA96] can be used to build the isomorphisms.) It is not clear how to extend this observation to handle sets complete under (PARITY of AND) or (AND of PARITY) reductions.

4 Conclusions

In closing, let us summarize our results. Berman and Hartmanis conjectured in [BH77] that all sets complete for NP under poly-time many-one reductions are P-isomorphic. Following the lead of [ABI93, AA96] we have considered the analogous question, where polynomial-time reductions and isomorphisms are replaced by AC^0-computable reductions and isomorphisms. We use our results about NC^0reducibility, superprojections, and an inherent gap in the power of reductions to prove a true analog of the Berman-Hartmanis conjecture. (That is, the sets complete under AC^0 reductions are all AC^0-isomorphic.) Finally, we show that the gap theorem does not extend as far as AC^0[mod 2] reductions.

We especially call attention to the following problems:

1. Does the Berman-Hartmanis conjecture hold for AC^0[mod 2] reductions?
2. Assuming the existence of a function that is one-way in a very strong average case sense, is it possible to construct a counter-example to the original Berman-Hartmanis conjecture?
3. Is there any class $\mathcal{C}$ such that Dlogtime-uniform AC^0-complete sets for $\mathcal{C}$ are all Dlogtime-uniform AC^0-isomorphic?

Acknowledgments

We acknowledge helpful conversations with O. Goldreich, J. Lafferty, M. Ogihara, D. van Melkebeek, R. Pruim, M. Saks, D. Sivakumar, and D. Spielman.

References

[Ag95] M. Agrawal, *DSPACE(n)$\stackrel{?}{=}$NSPACE(n): A degree theoretic characterization*, in Proc. 10th Structure in Complexity Theory Conference (1995) pp. 315–323.

[AA96] M. Agrawal and E. Allender, *An Isomorphism Theorem for Circuit Complexity*, in Proc. 11th Annual IEEE Conference on Computational Complexity (1996) pp. 2–11.

[Aj83] M. Ajtai, Σ_1^1 *formulae on finite structures*, Annals of Pure and Applied Logic 24, 1-48.

[Al89] E. Allender, *P-uniform circuit complexity*, J. ACM 36 (1989) 912–928.

[ABI93] E. Allender, N. Immerman, and J. Balcázar, *A first-order isomorphism theorem*, to appear in SIAM Journal on Computing. A preliminary version appeared in Proc. 10th Symposium on Theoretical Aspects of Computer Science, 1993, Lecture Notes in Computer Science 665, pp. 163–174.

[AG91] E. Allender and V. Gore, *Rudimentary reductions revisited*, Information Processing Letters **40** (1991) 89–95.

[AS92] N. Alon and J. Spencer, *The Probabilistic Method*, John Wiley and Sons, (1992).

[Ar95] Sanjeev Arora, *AC^0-reductions cannot prove the PCP theorem*, manuscript, 1995.

[BDG88] J. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity I and II*, Springer-Verlag, 1988, 1990.

[BIS90] David Mix Barrington, Neil Immerman, Howard Straubing, *On Uniformity Within NC^1*, J. Computer Sys. Sci. **41** (1990), 274-306.

[BH77] L. Berman and J. Hartmanis, *On isomorphism and density of NP and other complete sets*, SIAM J. Comput. **6** (1977) 305–322.

[FSS84] Merrick Furst, James Saxe, and Michael Sipser, *Parity, Circuits, and the Polynomial-Time Hierarchy,* Math. Systems Theory **17** (1984), 13-27.

[Hå87] J. Håstad, *One-Way Permutations in NC^0*, Information Processing Letters **26** (1987), 153-155.

[HILL90] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby, *Construction of a pseudo-random generator from any one-way function*, ICSI Technical Report, **No. 91-068** (1990).

[Jo75] Neil Jones, *Space-Bounded Reducibility among Combinatorial Problems,* J. Computer Sys. Sci. **11** (1975), 68–85.

[JY85] D. Joseph and P. Young, *Some remarks on witness functions for non-polynomial and non-complete sets in NP*, Theoretical Computer Science **39** (1985) 225–237.

[J72] J. Justesen, *A class of constructive asymptotically good algebraic codes*, IEEE Trans. Inform. Theory, **18** (1972), 652–656.

[KLD86] Ker-I Ko, Timothy J. Long, and Ding-Zhu Du, *On one-way functions and polynomial-time isomorphisms*, Theoretical Computer Science **47** (1986) 263–276.

[KMR90] S. Kurtz, S. Mahaney, and J. Royer, *The structure of complete degrees*, in A. Selman, editor, *Complexity Theory Retrospective*, Springer-Verlag, 1990, pp. 108–146.

[KMR95] S. Kurtz, S. Mahaney, and J. Royer, *The isomorphism conjecture fails relative to a random oracle*, J. ACM **42** (1995), 401–420.

[Li94] Steven Lindell, *How to define exponentiation from addition and multiplication in first-order logic on finite structures*, (manuscript). This improves an earlier characterization that appears in: Steven Lindell, *A purely logical characterization of circuit uniformity*, Proc. 7th Structure in Complexity Theory Conference (1992) pp. 185–192.

[Ni92] Noam Nisan, *Using Hard Problems to Create Pseudorandom Generators*, MIT Press (1992).

[Se92] A. Selman, *A survey of one way functions in complexity theory*, Mathematical Systems Theory **25** (1992) 203–221.

Appendix

5 Derandomizing the Switching Lemma

In the following section is a proof that the switching lemma can be carried out feasibly. More precisely, given a circuit C of depth d, and size $S = n^k$, with $n = rm$ underlying variables arranged into r blocks, each of size $m = r^a$ (a depends on d and k), there exists a restriction ρ to the variables such that $C\lceil_\rho$ only depends on a constant number of variables, and each of the r blocks has at least r^2 variables left unset. Furthermore, we give a uniform algorithm for finding ρ in time polynomial in the size of C.

The switching lemma statement and proof that we will follow is a simplification of that due to Furst, Saxe and Sipser but with two additional complications: (1) we need to take the blocks into account and (2) we need to give polynomial-time algorithms for finding the restrictions.

Let C be a depth d, size $S = n^k$ circuit. We will assume without loss of generality that S is arranged into d alternating levels of AND and ORs, and at the leaves of the circuit are constant-depth c decision trees. The proof will proceed in d steps. At step one, we will apply c successive restrictions in order to replace the bottom levels of (ANDs of constant-depth c decision trees) by (constant-depth c^2 decision trees), or similarly, in order to replace the bottom levels of (ORs of constant-depth c decision trees) by (constant-depth c^2 decision trees). In general in step i, we will be applying c^{2^i} restrictions in order to replace the bottom levels of ANDs of constant depth c^{2^i} decision trees by depth $c^{2^{i+1}}$ decision trees. Thus after d steps, the total number of restrictions applied will be $c + c^2 + c^{2^2} + ... + c^{2^d} = O(c^{2^d})$, with c^{2^i} restrictions at step i, for d steps. The underlying variables will always be grouped into r blocks, where the block size will be $m = r^l$ for some l. (l will shrink at each step.) After applying one restriction, we will still have r blocks, and exactly $r^{l/4}$ variables will remain unset within each block. (Each restriction will consist of a first part where $rr^{l/2}$ variables are chosen uniformly to be set to *, and with the condition that no block will have size less than $r^{l/4}$, and then a second clean-up part where we set additional variables so that each block will have uniform size $r^{l/4}$.)

We will now describe one step. Assume that the bottom level subcircuits have the form: AND of depth-c decision trees. Then each such subcircuit can be expressed as an AND of size-c ORs. Let $S_1, ..., S_q$ be the set of (polynomially-many) ANDs of size-c ORs. We proceed in c stages as follows.

In Stage 1, we will find a restriction ρ such that for each i, $S_i\lceil_\rho$ has a partial decision tree of constant depth c, and where each leaf is labelled by either a constant, or by an AND of size $(c-1)$ ORs. The restriction ρ is obtained by using Algorithm A. Stage j is the same as stage 1 , but now the set of formulas under consideration (the S_i's) are the nonconstant formulas labelling the leaves of the decision trees that have been created thus far. After stage j, we have created partial decision trees for the original S_i's, where now the leaves of the tree are either labelled by constants or they are labeled by ANDs of size $c-j$ ORs. For each stage, we use Algorithm A to find the restriction. Finally after c stages, we have decision trees for the original S_i's where all leaves are labelled by constants.

After one step, we have gone from a depth d size S circuit with rm underlying variables, arranged into r blocks, where each block has size m, to a depth $d-1$, size S circuit, where now the number of underlying variables is rm', $m' = m^{1/4^c}$, again arranged into r blocks, and where each block has size m'. Note that now the bottom level consists of decision trees of depth c^2. After repeating this for d steps, we will have obtained a depth c^{2^d} decision tree representing the original circuit on the remaining variables. At the end, there will be rm'' remaining variables, where $m'' = m^{-4^{c^{2^d}}}$, and as before, the variables are arranged into blocks of size r, with each block of size at least m''.

We will now describe Algorithm A.

5.1 Algorithm A

The input to this algorithm is a collection of polynomially many formulas $Q_1, ..., Q_q$, where each Q_i is an AND of size c ORs. (Or alternatively, each Q_i is an OR of size c ANDs. This case is handled dually so we will not consider it here.) There are rm underlying variables, arranged into blocks $b_1, ..., b_r$, where each block has size m (The value of m will be r^a for a appropriately chosen. Thus, m is a polynomial in r.) The output is a restriction ρ such that: (1) ρ assigns exactly $m^{1/4}$ *'s to each block, and all other variables are set to 0 and 1; (2) for each Q_i, we can construct a depth c decision tree for $Q_i\lceil_\rho$ such that the leaves of the decision tree are all labelled by either a constant, or by an AND of size $(c-1)$-ORs.

Given a Q_i, we define a set of clauses $Maxset(Q_i)$ as follows. First find the lexicographically first set of clauses in Q_i that are variable-disjoint. If the number of clauses in this set is greater than $f \log m$, then let $Maxset(Q_i)$ be the lexicographically first $f \log m$ of these clauses. (So $|Maxset(Q_i)| \leq f \log m$.) We divide the Q_i's into two disjoint sets: First, $\{n_1, ..., n_s\}$, the narrow formulas, are those Q_i's such that $|Maxset(Q_i)| < f \log m$. Secondly, $\{w_1, ..., w_t\}$, the wide formulas, are those Q_i's such that $|Maxset(Q_i)| = f \log m$. We will find a restriction ρ setting at most $rm^{1/2}$ variables such that:

(1) ρ assigns at most $m^{1/4}$ *'s to each block;

(2) for each n_i, the number of underlying literals in $Maxset(n_i)$ that are set to * by ρ is at most c; and

(3) for each w_j, at least one clause in $Maxset(w_j)$ is set to 0 by ρ.

Once we have found such a restriction, we set additional variables in order to set exactly $m^{1/4}$ variables per block. Secondly, for each w_j, we can create the trivial decision tree for $w_j\lceil_\rho$ labelled by 0. Thirdly, for each n_i, we can create a depth c decision tree for $n_i\lceil_\rho$ by querying the *'d variables in $Maxset(n_i)\lceil_\rho$. By property (2), there are at most c such variables. Once these have all been queried, we are left at each leaf with either a constant or with an AND of size $(c-1)$ ORs, since every other clause intersects at least one variable of $Maxset(n_i)$, and all variables in $Maxset(n_i)$ have been set.

The following three lemmas show that such a restriction ρ exists.

Lemma 10 *Let $\{b_1, ..., b_r\}$ be a partition of the underlying rm variables into r blocks. Let B_i be the event that block b_i has less than $m^{1/4}$ *'s after ρ is applied. Then $\sum_i Pr[B_i] \leq 1/4$, where the probability is over all restrictions ρ setting exactly $rm^{1/2}$ variables to *.*

Proof. Let p be the probability that a particular element is *'d. Then $p = 1/\sqrt{m}$. Let the size of b_i be m and let $l = m^{1/4} - 1$. Then we have, for all large m,

$$\begin{aligned} Pr[B_j] &= \sum_{i=0}^{l} \binom{|b_j|}{i} p^i (1-p)^{|b_j|-i} \\ &\leq \sum_{i=0}^{l} e^{-mp} \binom{m}{i} \\ &\leq l(m^l e^{-pm}) \\ &\leq 2^{-m^{1/4}}. \end{aligned}$$

Summing up over all B_j, the total probability is at most 1/4. ∎

Lemma 11 *Let $S = s_1, ..., s_l$ where each s_i is a collection of at most $cf \log m$ literals, where l is a polynomial in m, and where the s_i's are pairwise disjoint. (For a given narrow formula n_i, s_i's is the set of variables that underly the clauses in $Maxset(n_i)$; since there are less than $f \log m$ clauses in $Maxset(n_i)$, the total number of variables in s_i is $cf \log m$.) Let N_i be the event that s_i has more than c *'s after ρ. is applied. (I.e., N_i is the bad event that the narrow formula n_i does not satisfy property (2) above.) Then $\sum_i Pr[N_i] \leq 1/4$.*

Proof. Let m be the original number of variables, and let m' be the number of *'d variables in ρ. Then $p = m'/m$ is the probability that a particular variable is set to $*$ by a random ρ. We will first get an upper bound on $Pr[N_i]$. The expected number of elements in s_i set by $*$ is $|s_i|p \leq O(\log m)p$. Using Chernoff bounds on the tail of the distribution, the probability that there are more than c *'s in s_i is at most $(\frac{e|s_i|p}{c})^c$. Summing up over all s_i, the total probability is at most 1/4. ∎

Lemma 12 *Let $\{w_1, ..., w_l\}$ be polynomially many ANDs of size c ORs, where for each w_i, $|Maxset(w_i)| = f \log m$. (The w_i's are the wide formulas.) Let the underlying universe be of size rm. Then with probability at most $1/4$, a random restriction setting $rm^{1/2}$ variables to * has the property that for some w_i, no clause in $Maxset(w_i)$ is set to 0 by ρ.*

Proof. For a given w_i, let W_i be the bad event that no clause in $Maxset(w_i)$ is set to zero. Let $s_1, ..., s_{f \log m}$ denote the underlying clauses in $Maxset(w_i)$. We will first show that for a given polynomial $p(m)$ there exists an f (depending on $p(m)$ and c) such that $Pr[W_i] < a$, where $a < 1/4p(r)$.

$$
\begin{aligned}
Pr[W_i] &\leq \Pi_{j=1}^{f \log m} Pr[s_j \text{ is not all zero}] \\
&= \Pi_{j=1}^{f \log m}(1 - Pr[s_j \text{ is all zero}] \\
&= \Pi_{j=1}^{f \log m}(1 - (\frac{rm - r\sqrt{m}}{2rm})^c) \\
&\leq \Pi_{j=1}^{f \log m}(1 - (1/4)^c) \\
&= (1 - (1/4)^c)^{f \log m} \\
&\leq e^{-(f \log m)/4^c}.
\end{aligned}
$$

For f chosen appropriately, the above probability is less than $1/4p(m)$. Since the total number of w_i's is $p(m)$, the total probability that some w_i does not have a clause in $Maxset(w_i)$ that is set to 0, is at most $1/4$. ∎

We now want to obtain a good ρ using the method of conditional probabilities [AS92]. We will obtain ρ by choosing one element at a time to be set. That is, we first choose one of the $2rm$ literals and set it to 1. Then we choose one of the remaining $2(rm-1)$ literals and set it to 1. The process terminates after we have set a total of $rm - rm^{1/2}$ variables.

The algorithm for finding ρ proceeds as follows. First, for each of the $2rm$ literals l, we calculate the following quantities exactly: $Pr(B_i \mid l)$, $Pr[N_j \mid l]$ and $Pr[W_k \mid l]$, where $Pr(B_i \mid l)$ is the probability of event B_i, over a randomly chosen ρ, given that literal l is set to 1. Each of these quantities can be calculated exactly in polynomial time. We choose a literal l to be set to 1 such that the sum $\sum_i Pr[B_i \mid l] + \sum_j Pr[N_j \mid l] + \sum_k Pr[W_k \mid l]$ is minimized. By the three lemmas above, $\sum_i Pr[B_i] + \sum_j Pr[N_j] + \sum_k Pr[W_k]$ is at most $3/4$. Thus it follows that for some variable l we do at least as well as $3/4$. (There are two arguments to see that this follows: (1) you can view the sample space of possible ρ's as larger than the original one, where the $rm - rm'$ set variables are ordered, and then do the following calculations relative to this enlarged sample space. In this case, the conditions l are independent so when we do the above sum over all $2rm$ conditions l we get exactly the same number as the original unconditional sum. Or (2) work in the original sample space of possible ρ's. In this case, the conditional spaces given l are not independent, but they are completely symmetric so the averaging argument is still valid.)

We repeat this argument $rm - rm^{1/2}$ times, at each point conditioning upon the set H of variables set thus far. At the end, we are guaranteed to have obtained a good restriction since we maintain that the conditional probability is always no greater than the original probability which is less than 1.

It is left to show how to exactly calculate the quantities $Pr[B_i \mid H]$, $Pr[N_j \mid H]$, and $Pr[W_k \mid H]$, where H is a collection of at most $rm - rm^{1/2}$ variables that have been set.

Let A be a set of size a; let H be a set of h variables that have been set; let $|A \cap H| = d$; let rm be the original universe size, and let rm' be the number of *'s after ρ has been applied. Then the probability that $A\lceil_\rho$ has more than l *'s, given that H has already been set to 0,1 is:

$$
\sum_{i=l+1}^{a} \frac{\binom{a-d}{i}\binom{rm-a-h+d}{rm'-i}}{\binom{rm-h}{rm'}}.
$$

This quantity is used to calculate exactly $Pr[N_j \mid H]$, and a very similar formula can be used to calculate $Pr[B_i \mid H]$. Calculating $Pr[W_k \mid H]$ exactly is more work. Consider a particular w_k, and let $s_1, ..., s_r$, $r = f \log m$, denote the $f \log m$ disjoint clauses in $Maxset(w_k)$, each consisting of the OR of at most c disjoint literals. Recall that W_i is the event that none of the s_i's are set to 0 by ρ. In order for this to happen, each s_i must have at most $|s_i| - 1$ of its literals set to 0, and the remaining literals in s_i can be set to either * or to 1. We calculate this quantity straightforwardly by considering all possible subsets, x_i and y_i of s_i, where x_i is the set of at most $|s_i| - 1$ literals in s_i set to 0, and y_i is the subset of remaining literals in s_i set to 1. While doing the calculation, we have to keep track of which of these possibilities are actually not valid due to the fact that H has already been set. Let $I(x_1, y_1, ..., x_{f \log m}, y_{f \log m}, H)$ be an indicator random variable that outputs 1 if the assignment given by setting all literals in the x_i's to zero, and setting all literals in the y_i's to one, is consistent with the assignment H. Also let $b = |H \cap (s_1 \cup s_2 ... \cup s_r)|$. We can compute $Pr[W_k \mid H]$ as $A/(\binom{rm}{rm-rm'}2^{rm-rm'})$, where A is given by the sum over all $x_1, y_1, x_2, y_2, ..., x_r, y_r$ of the following quantity, where the x_i's and y_i's satisfy: $x_1 \subset s_1$, $|x_1| \leq |s_1| - 1|$, $y_1 \subset s_1$, $x_1 \cap y_1 = \emptyset$, $\ldots$, $x_r \subset s_r$, $|x_r| \leq |s_r| - 1$, $y_r \subset s_r$, $x_r \cap y_r = \emptyset$.

$$
I(x_1, y_1, .., x_r, y_r, H) \times \binom{rm - |s_1| - .. - |s_r| - h + b}{rm - rm' - |x_1| - |y_1| - ... - |x_r| - |y_r| - h + b} \times 2^{rm - rm' - |x_1| - |y_1| - .. - |x_r| - |y_r| - h + b}
$$

The total number of sets x_i, and y_i in the above sum at most 2^c. Thus the total number of terms in the above summation is bounded by $2^{cr} = 2^{cf \log m}$ which is polynomial in m.

To see that the entire algorithm is polynomial time, note that the number of iterations of the above algorithm is $rm - rm'$, and each iteration takes time polynomial in m. Furthermore, the entire procedure for finding ρ is polynomial time, since we apply the above algorithm for a constant number of stage, and at each stage the number of formulas under consideration is also polynomially-many.