comput. complex. **10** (2001), 117 – 138
1016-3328/01/020117–22 \$ 1.50+0.20/0

computational complexity

REDUCING THE COMPLEXITY OF REDUCTIONS

Manindra Agrawal, Eric Allender, Russell Impagliazzo, Toniann Pitassi, and Steven Rudich

Abstract. We build on the recent progress regarding isomorphisms of complete sets that was reported in Agrawal *et al.* (1998). In that paper, it was shown that all sets that are complete under (non-uniform) AC^0 reductions are isomorphic under isomorphisms computable and invertible via (non-uniform) depth-three AC^0 circuits. One of the main tools in proving the isomorphism theorem in Agrawal *et al.* (1998) is a "Gap Theorem", showing that all sets complete under AC^0 reductions are in fact already complete under NC^0 reductions. The following questions were left open in that paper:

1. Does the "gap" between NC^0 and AC^0 extend further? In particular, is every set complete under polynomial-time reducibility already complete under NC^0 reductions?
2. Does a uniform version of the isomorphism theorem hold?
3. Is depth-three optimal, or are the complete sets isomorphic under isomorphisms computable by depth-two circuits?

We answer all of these questions. In particular, we prove that the Berman–Hartmanis isomorphism conjecture is true for P-uniform AC^0 reductions. More precisely, we show that for any class $\mathcal{C}$ closed under uniform TC^0-computable many-one reductions, the following three theorems hold:

1. If $\mathcal{C}$ contains sets that are complete under a notion of reduction at least as strong as Dlogtime-uniform AC^0[mod 2] reductions, then there are such sets that are not complete under (even non-uniform) AC^0 reductions.
2. The sets complete for $\mathcal{C}$ under P-uniform AC^0 reductions are all isomorphic under isomorphisms computable and invertible by P-uniform AC^0 circuits of depth-three.
3. There are sets complete for $\mathcal{C}$ under Dlogtime-uniform AC^0 reductions that are not isomorphic under any isomorphism computed by (even non-uniform) AC^0 circuits of depth two.

To prove the second theorem, 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.)

Keywords. Isomorphisms; completeness; constant-depth circuits; Berman–Hartmanis Conjecture; powering in finite fields.

Subject classification. 68Q17.

1. Introduction

Most of the computational problems that arise in practice turn out to be complete for one of a handful of complexity classes, even under *very* restrictive notions of reducibility. Indeed, it was noted in Berman & Hartmanis (1977) that the natural complete sets can even be shown to be *isomorphic* to each other under bijections computable and invertible in polynomial time, and thus they can be viewed as simple re-encodings of each other. This and other considerations led to the famous Berman–Hartmanis Conjecture (Berman & Hartmanis 1977) that *all* NP-complete sets are p-isomorphic.

It was shown in Agrawal *et al.* (1998) that a version of this conjecture is true. More precisely, it was shown that in NP (and in most other complexity classes of interest), all of the sets that are complete under AC^0 reductions are isomorphic to each other under bijections computable and invertible by (non-uniform) depth-three AC^0 circuits. This is a very natural re-statement of the Berman–Hartmanis Conjecture, since (a) AC^0 reductions are the most natural notion of reducibility to consider when presenting complete sets for small classes such as NC^1 or $\mathrm{DSPACE}(\log n)$, and (b) all of the well-known complete sets for NP and other complexity classes are complete even under AC^0 reductions.

The work mentioned above leads us to ask whether, in fact, *all* sets complete for a well-known complexity class (e.g., NP) under polynomial-time reductions are already complete under AC^0 reductions. (In regard to this question, it is interesting to note that Veith (1998) shows that all "succinctly represented" problems that are complete under polynomial-time reductions *are* complete under AC^0 reductions. In fact, these problems are all complete under projections, which are an even more restrictive notion of reducibility.) This possibility may seem unlikely, especially in light of the fact that there are many functions computable in polynomial time that are not computable in AC^0. However, it was shown in Agrawal *et al.* (1998) that all sets complete under AC^0 reductions are complete under NC^0 reductions, in spite of the fact that there are many functions computable in AC^0 that are not computable in NC^0. In this paper, we give a negative answer to this question by showing:

THEOREM 1.1 ("Stop Gap Theorem"). *There exists a set that is complete for NP under Dlogtime-uniform* $\mathrm{AC}^0[\mathrm{mod}\ 2]$ *reductions but not under non-uniform* AC^0 *reductions.*

Also, by derandomizing the version of the Switching Lemma used in Agrawal *et al.* (1998) we extend the isomorphism theorem of Agrawal *et al.* (1998) to P-uniform AC^0 reductions:

Theorem 1.2. *All sets complete for NP under P-uniform* AC^0 *reductions are isomorphic to each other via isomorphisms computable and invertible by depth-three P-uniform* AC^0 *circuits.*

Finally, we show that the above result cannot be improved to depth two:

Theorem 1.3. *There exist two sets—both complete for NP under Dlogtime-uniform* AC^0 *reductions—such that no isomorphism between the two sets can be computed by depth-two non-uniform* AC^0 *circuits.*

This result implies that the isomorphisms *cannot* be computed by NC^0 circuits since any NC^0 circuit can be converted to a depth-two AC^0 circuit. This observation, coupled with the fact that the two sets above are also complete under u-uniform NC^0 reductions for any reasonable notion of uniformity u, implies that the Berman–Hartmanis Conjecture is *false* for NC^0 reductions for any reasonable notion of uniformity.

As in Agrawal *et al.* (1998), all our results hold not just for NP, but for any class closed under Dlogtime-uniform TC^0-computable many-one reductions.

The paper is organized as follows. Section 2 presents definitions for the classes of reductions considered in this paper. In Sections 3, 4, and 5 we prove Theorems 1.1, 1.2, and 1.3 respectively. And finally, Section 6 contains a discussion on the results obtained and future directions for research.

2. Basic definitions and preliminaries

We assume familiarity with the basic notions of many-one reducibility as presented, for example, in Balcázar *et al.* (1995, 1990). In this paper, only many-one reductions will be considered.

A *circuit family* is a set $\{C_n : n \in \mathbb{N}\}$ 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 (Barrington *et al.* 1990) and P-uniformity (Allender 1989) (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 logic with $\{+, \times, \leq\}$ (Barrington *et al.* 1990; Lindell 1992)[1], the logspace-rudimentary reductions of Jones (Allender & Gore 1991; Jones 1975), logarithmic-time alternating Turing machines with $O(1)$ alternations (Barrington *et al.* 1990) 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 Håstad (1987).

For a complexity class $\mathcal{C}$, a $\mathcal{C}$-*isomorphism* is a bijection f such that both f and f^{-1} are in $\mathcal{C}$. (To eliminate unnecessary notation, we follow standard practice in ignoring the distinction between the set of decision problems $\mathcal{C}$ and the closely-related set of functions. Thus, for instance, AC^0 can be viewed as

[1]Lindell (1992) shows only that this coincides with first-order expressibility in first order logic with $\{+, \times, \leq, \exp\}$, where "exp" denotes exponentiation. However, personal communication from K. Regan and S. Lindell shows that exponentiation can be eliminated. For details, see Immerman (1998).

either a set of languages or as a set of functions, with no confusion.) 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 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., have this closure property.

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* (resp. *length-increasing*, *length-squaring*) if, for all x, $|x| \leq |f(x)|$ (resp. $|x| < |f(x)|$, $|x|^2 \leq |f(x)|$); 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. Proof of Theorem 1.1

Let SAT be the set of strings coding satisfiable Boolean formulas (under some standard coding scheme). SAT is complete for NP under Dlogtime-uniform AC^0 reductions (and, in fact, even under projections). Let PARITY be the set of all binary strings with an odd number of ones. PARITY is in NP.

The idea behind the proof is as follows. We first define a function f that is computable by $\mathrm{AC}^0[\mathrm{mod}\ 2]$ circuits, and is an error-correcting code capable of correcting a "large" fraction of errors. Since f is 1-1, and computable in Dlogtime-uniform $\mathrm{AC}^0[\mathrm{mod}\ 2]$, SAT is $\mathrm{AC}^0[\mathrm{mod}\ 2]$ reducible to $f(\mathrm{SAT})$. Thus, $f(\mathrm{SAT})$ is complete for NP under $\mathrm{AC}^0[\mathrm{mod}\ 2]$ reductions. Assuming $f(\mathrm{SAT})$ is also complete under AC^0 reductions (in that case, it is also complete under NC^0 reductions by the Gap Theorem (Agrawal *et al.* 1998)), we consider an NC^0 reduction of PARITY to $f(\mathrm{SAT})$. For any input length n, most of the input bits of the reduction circuit can influence very few output bits whereas the strings in $f(\mathrm{SAT})$ are very far apart. Thus, this circuit must map all inputs in PARITY to the same output. This gives an AC^0 circuit for PARITY, a contradiction.

It turns out that the standard Reed–Solomon code can be used to define function f. For the purpose of self-containment, we provide a description of function f:

> Input x, $|x| = n$. Let $y = x10^k$ with k being the smallest number such that $|y|$ is divisible by t ($t = O(\log n)$ to be fixed later). Let $y = a_0 a_1 \cdots a_s$ where each a_i has t bits. Let polynomial $Y(z) = \sum_{i=0}^{s} a_i \cdot z^i$ where each a_i is treated as an element of $\mathbb{F}_{2^t}$—the finite field of 2^t elements. Let $b_1, \ldots, b_{2^t}$ be an enumeration of all elements of $\mathbb{F}_{2^t}$. Output the string
>
> $$Y(b_1) \cdots Y(b_{2^t}),$$
>
> where $Y(b_i)$ is evaluated over $\mathbb{F}_{2^t}$.

Note that if $x \neq x'$ are two strings of length n, then $f(x)$ and $f(x')$ differ in at least $2^t - s$ bits (and thus the fraction of the bit positions in which they differ is at least $(2^t - s)/(t2^t)$). To see this, let Y and Y' be the associated polynomials as defined above. Clearly $f(x)$ and $f(x')$ agree in block j if and only if $(Y - Y')(b_j) = 0$. Since $Y - Y'$ is a non-zero polynomial of degree s, this can happen only for s distinct b_j's, and hence $f(x)$ and $f(x')$ differ in at least $2^t - s$ blocks of t bits, which establishes the claim.

Number t should be $O(\log n)$ to ensure that f is polynomially bounded. It should also be greater than $\log s = \Theta(\log n)$, since otherwise the code is meaningless. To facilitate computing in $\mathbb{F}_{2^t}$, we choose $t = 2 \cdot 3^\ell$ for the smallest ℓ such that $2^t \geq 2n$. For this choice of t, the polynomial $z^t + z^{t/2} + 1$ is irreducible in $\mathbb{F}_2[z]$ and thus $\mathbb{F}_2[z]/(z^t + z^{t/2} + 1) = \mathbb{F}_{2^t}$.

With this value of t, the fraction of bits in which two codewords differ is

$$\delta = \frac{2^t - s}{t \cdot 2^t} \geq \frac{2^t - 2^{t-1}}{t \cdot 2^t} = \frac{1}{2t} \geq \frac{1}{24 \log n}$$

for large enough n.

Let us now consider the complexity of computing $f(x)$. Consider any particular output bit. This bit of the output is one of the bits of $Y(b_j)$ for some $j \leq 2^t$, where $Y(z)$ is the polynomial $\sum_i a_i \cdot z^i$, where each a_i consists of $t = O(\log n)$ bits of x.

Note that b_j^i is a constant, not depending on the input x. It follows from recent progress on the circuit complexity of division (Hesse 2001) that b_j^i can actually be made available as a constant in Dlogtime-uniform AC^0; details can be found in Theorem 3.2 below. Thus we can view the constants b_j^i as being "hardwired" into the circuit computing f. Next, consider how to compute $a_i \cdot w$, where w is any t-bit constant (such as $w = b_j^i$). Of course, since a_i is the bitwise sum of its components, $a_i \cdot w$ can be expressed as a sum of $t = O(\log n)$ terms of the form $0 \ldots 0 x_j 0 \ldots 0 \cdot z$, where x_j is one of the bits of x. The term $0 \ldots 0 x_j 0 \ldots 0 \cdot z$ can be computed by (1) obtaining a $2t$-bit vector representing the polynomial q of degree $\leq 2(t-1)$ that one obtains by multiplying z by $0 \ldots 0 x_j 0 \ldots 0 \cdot z$ (this is just a left shift of z, unless x_j is 0), and then (2) finding (by table look-up) the t-bit vector v_j such that q is equivalent to v_j mod $z^t + z^{t/2} + 1$. Note that v_j can be found in Dlogtime-uniform AC^0, by checking if there exists a vector v' such that $q - v_j = v' \cdot (z^t + z^{t/2} + 1)$. That is, $a_i \cdot w$ can be expressed a sum of $O(\log n)$ terms of the form v_j, where each component of v_j is either 0 or x_j. Thus $a_i \cdot z$ can be computed by t PARITY gates, each of which is computing the sum in $\mathbb{F}_2$ of a component of the v_j's.

The final output $\sum_i a_i \cdot b_j^i$ can thus be computed by t PARITY gates, each connected to some of the PARITY gates computing $a_i \cdot b_j^i$. This establishes that f can be computed in Dlogtime-uniform $\mathrm{AC}^0[\mathrm{mod}\ 2]$.

A closer examination of the foregoing algorithm shows that the AND and OR gates are used only in computing certain constants that do not depend on the input, and that each bit of the output is simply the PARITY of some of the input bits (and these connections can be computed in Dlogtime-uniform AC^0). This establishes that f is computed by very uniform depth-one circuits consisting only of PARITY gates.

Let $S = \{f(x) : x \in \mathrm{SAT}\}$. Since f is 1-1, it is a reduction from SAT to S. Since f is computable in (Dlogtime-uniform) $\mathrm{AC}^0[\mathrm{mod}\ 2]$, and SAT is NP-complete under AC^0 reductions, S is NP-complete under (Dlogtime-uniform) $\mathrm{AC}^0[\mathrm{mod}\ 2]$ reductions. Suppose, for a contradiction, that S is NP-complete under non-uniform AC^0 reductions. In particular, there must be an AC^0 reduction from PARITY to S. By invoking the Gap Theorem of Agrawal *et al.* (1998), we see that there must be an NC^0 reduction from PARITY to S. In fact, it is shown in Agrawal *et al.* (1998) that this NC^0 reduction has the property that the length of the output depends only on the length of the input. Let this reduction be computed by circuit family C_n.

Fix an integer n, and consider the circuit C_n that defines the reduction on strings of length n. Let C_n have m output bits. There is a constant b such that each output bit of the circuit C_n depends on at most b input bits. Let o_i be the number of output bits that depend on variable x_i. The sum of o_i's is therefore bounded by mb. Hence at most $2b/\delta = O(\log n)$ o_i's can be greater than $\delta m/2$. In other words, at most $2b/\delta$ input variables influence (i.e., are inputs to) $\geq \delta m/2$ output bits. Thus we can set these $O(\log n)$ additional input variables and obtain an NC^0 circuit family on $n' \geq n - O(\log n)$ input variables that also reduces PARITY to S, and has the property that every input variable influences fewer than $\delta m/2$ output bits. Call this new circuit $D_{n'}$, and let g be the function computed by it.

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 $f(\mathrm{SAT})$), 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 $10^{n'-1}$ by a sequence $10^{n'-1} = x_1, x_2, \ldots, x_r = x$ 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 $\{x : g(x) = g(10^{n'-1})\}$. Thus, we can construct a circuit that first computes $g(x)$ using $D_{n'}$ and then checks equality with $g(10^{n'-1})$ using a single "AND" gate of fan-in m. This constant-depth circuit has size $O(|C_n|) = n^{O(1)} = {n'}^{O(1)}$, and since n' gets arbitrarily large as n does, this contradicts the lower bounds for computing parity via constant-depth circuits (Ajtai 1983; Furst *et al.* 1984).

Examining the proof, we used only the facts that PARITY $\in$ NP, NP has a complete set for AC^0[mod 2] reductions, and that NP is closed under AC^0[mod 2] reductions. Generalizing, we get:

THEOREM 3.1. *Let $\mathcal{R}$ be a class of functions closed under composition and containing Dlogtime-uniform* AC^0[mod 2]. *Let $\mathcal{C}$ be a complexity class closed under both Dlogtime-uniform* TC^0 *reductions and $\mathcal{R}$-reductions, and having a set that is complete under $\mathcal{R}$-reductions. Then $\mathcal{C}$ contains an $\mathcal{R}$-complete set that is not complete under non-uniform* AC^0 *reductions.*

3.1. Powering in finite fields. To complete the proof of Theorem 1.1, we need only provide the proof that powering in small finite fields can be performed in Dlogtime-uniform AC^0.

THEOREM 3.2. *Let $t = 2 * 3^\ell$ for some ℓ, where $t = O(\log n)$. Then, given input (a, i, b) of length $O(\log n)$, it can be determined in Dlogtime-uniform* AC^0 *if $a^i = b$, where a and b are elements of $\mathbb{F}_{2^t}$.*

We remark that the restriction on t is merely so that we can be explicit about our choice of an irreducible polynomial. In Dlogtime-uniform AC^0, it is possible to locate the lexicographically-first irreducible polynomial of degree $t = O(\log n)$, and use it to represent $\mathbb{F}_{2^t}$, for any choice of t.

Proof. By Hesse (2001), it is sufficient to show that a Dlogtime-uniform AC^0 circuit family exists that takes as input the binary representations of r elements of $\mathbb{F}_{2^t}$ (where $r = O(\log n)$; call these elements $(a_1, \ldots, a_r)$) and computes $\prod_i a_i$.

Each of the a_i's can be viewed as a polynomial over $\mathbb{F}_2$ (taken modulo the irreducible polynomial $z^t + z^{t/2} + 1$). Our approach, modeled on Frandsen *et al.* (1994), will be to use Chinese Remaindering over the ring of polynomials over $\mathbb{F}_2$.

Let $h(z)$ be the polynomial $z^{2^b} - z$, where b is the smallest number such that $2^b > rt = O(\log^2 n)$. As pointed out in Frandsen *et al.* (1994), h has 2^b distinct factors in $\mathbb{F}_{2^b}$, and thus each of the irreducible factors of h (call them $h_1, \ldots, h_k$) has degree bounded by $b = O(\log t) = O(\log \log n)$.

By Lemma 3.3 below, in uniform AC^0 we can test, for each short bit string representing a small polynomial h', if h' divides h. Also, it is simple to check, for such a polynomial h', that no other polynomial divides h'. Thus we can find the irreducible factors of h in uniform AC^0.

Let A be the polynomial of degree $O(\log^2 n)$ that results by taking the product of the polynomials representing $(a_1, \ldots, a_i)$ in $\mathbb{F}_2[z]$. By Chinese Remaindering, A can be represented by the sequence $(\hat{A}_1, \ldots, \hat{A}_k)$, where $\hat{A}_j$ is the remainder obtained when dividing the polynomial A by h_j. Similarly, if we let $\widehat{a_{i,j}}$ be the remainder of dividing a_i by h_j, then we have $\hat{A}_j = \prod_i \widehat{a_{i,j}}$, where the product is taken in $\mathbb{F}_2[z]/h_j$. Since the multiplicative group of $\mathbb{F}_2[z]/h_j$ is cyclic and of size $\log^{O(1)} n$, it is easy in Dlogtime-uniform AC^0 to find a generator of $\mathbb{F}_2[z]/h_j$, and compute a table of discrete logs relative to this generator. (To see this, for each potential generator g, build a graph with nodes for group elements and edges representing multiplication by g. Each node can be represented with $O(\log \log n)$ bits; thus a path of length $\log n/\log\log n$ can be represented with $O(\log n)$ bits. Hence it is easy in Dlogtime-uniform AC^0 to determine if there is a path of length $i < \log n/\log\log n$ between two nodes. Iterating this construction $O(1)$ times allows one to look for paths of length $\log^{O(1)} n$. The node g is a generator if and only if the graph is connected.) Now the product $\prod_i \widehat{a_{i,j}}$ can be computed by adding the discrete logs. Since addition of $\log^{O(1)} n$ numbers can be performed in Dlogtime-uniform AC^0, it is clear that we can compute the Chinese Remainder representation of A, $(\hat{A}_1, \ldots, \hat{A}_k)$, in Dlogtime-uniform AC^0.

To complete the proof, we need only show how we can obtain the coefficients of A from the Chinese Remainder representation, and then divide A by $z^t + z^{t/2} + 1$ to obtain the representative element of $\mathbb{F}_2[z]/(z^t + z^{t/2} + 1)$. (This latter division can be accomplished, by appeal to Lemma 3.3.)

By the Chinese Remainder Theorem, A is equivalent to $\sum_{i=1}^{k} \hat{A}_i c_i d_i$ modulo h, where $c_i = h/h_i$, and d_i is an element of $\mathbb{F}_2[z]/h_i$ such that $c_i d_i = 1 \bmod h_i$. By Lemma 3.3, the representation of c_i can be computed in Dlogtime-uniform AC^0. Since $\mathbb{F}_2[z]/h_i$ is so small, d_i can be found by brute force. Thus we can compute each term of the sum. Since there are only $\log n$ terms in the sum, and each component of the term can be computed by taking the parity of $O(\log n)$ elements, we can compute the coefficients of A, as required. □

LEMMA 3.3. *Let $k \in \mathbb{N}$. Then there is a Dlogtime-uniform AC^0 circuit family that takes as input a sequence of coefficients defining two polynomials h_1 and $h_2 \in \mathbb{F}_2[z]$ of degree $\log^k n$, and outputs the sequence of coefficients for polynomials q and r such that the degree of r is less than the degree of h_2, and such that $h_1 = h_2 \cdot q + r$. That is, division with polynomials of degree $\log^{O(1)} n$, and finding remainders, can be performed in Dlogtime-uniform AC^0.*

PROOF. Let $m = \log^k n$. Eberly (1989) shows that division of polynomials of degree m is reducible to the problem of computing the product of $m^{O(1)}$ integers, each having $m^{O(1)}$ bits. Eberly claims only an NC^1-Turing reduction, but an examination of his proof shows that it can be implemented as a Dlogtime-uniform AC^0-Turing reduction. However, it is shown in Hesse (2001) that computing the product of $\log^{O(1)} n$ integers, each of length $\log^{O(1)} n$, can be computed in Dlogtime-uniform AC^0. □

4. Proof of Theorem 1.2

Our proof of Theorem 1.2 results from a technical improvement of one of the steps from the proof of the Isomorphism Theorem in Agrawal *et al.* (1998). We will not repeat the construction here, but we will provide a very high-level sketch of the approach taken there. There are three main parts of the argument in Agrawal *et al.* (1998):

- a *gap theorem* (to which we have already referred in this paper), stating that sets complete under AC^0 reductions are also complete under NC^0 reductions,
- a technical theorem, showing that all sets complete under NC^0 reductions are complete under easily invertible reductions called *superprojections*, and
- an *isomorphism theorem*, showing that all sets complete under superprojections are isomorphic under depth-three AC^0 isomorphisms.

The technical theorem and the isomorphism theorem of Agrawal *et al.* (1998) also hold in the P-uniform setting. Thus, in order to prove Theorem 1.2, it suffices to prove a P-uniform version of the Gap Theorem:

THEOREM 4.1. *All sets hard for NP under P-uniform* AC^0 *reductions are hard for NP under P-uniform* NC^0 *reductions.*

We now outline the proof of the Gap Theorem of Agrawal *et al.* (1998) in order to identify the non-uniform step. Let A be a hard set for NP under AC^0 reductions. We need to show that any set B in NP has an NC^0 reduction to A. For this, first a version B' of B is defined with a lot of redundancy: corresponding to a string x in B, B' has many strings with each such string y having $|x|$ *blocks* of bits such that the i^{th} bit of x equals the parity of the i^{th} block of bits of y (this is not the exact definition of B' as in Agrawal *et al.* (1998) but captures the essential idea). The set B' also is in NP and therefore there is an AC^0 reduction, given by circuit family $\{C_m\}$, of B' to A. It is implicit in Furst *et al.* (1984) and Ajtai (1983) (and a detailed proof is provided in Agrawal *et al.* (1998)) that a random restriction[2] of the input variables of circuit C_m transforms it (with high probability) to an NC^0 circuit on at least m^ϵ bits for some $\epsilon > 0$. If we divide the input into n blocks of equal length with $n = m^{\epsilon/2}$, a simple probability calculation shows that in the random restriction, with high probability, each of these blocks would have at least three unset bits. Fix a restriction τ_m that has both the above properties. Modify this restriction as follows: in each block, set all but one of the unset bits in such a way that parity of all the set bits in the block becomes zero. (Since each block starts with at least three unset bits, we can always do this—actually two unset bits suffice for this but three unset bits are needed for technical reasons in Agrawal *et al.* (1998).) Let this modified restriction be τ'_m (clearly, τ'_m transforms C_m to an NC^0 circuit on n bits). We now define a reduction of B to B' as: given x, $|x| = n$, output $\tau'_m(x)$ which is the string constructed from τ'_m by filling in the i^{th} bit of x into the unset bit of the i^{th} block of τ'_m. By the definition of B', $x \in B$ iff $\tau'_m(x) \in B'$. Also, this reduction has trivial circuit complexity—it is just a projection. A composition of this circuit with C_m yields an NC^0 circuit which reduces B to A, completing the proof.

In the above construction, although the circuit computing the reduction of B to B' is trivial, it is non-uniform since it requires a "good" random restriction τ_m. In the lemma below, we show how to compute such a restriction

[2] A random restriction here leaves each bit unset with probability $1/m^{1-2\epsilon}$, and sets it to 1 or 0 with probability $\frac{1}{2}(1 - 1/m^{1-2\epsilon})$ each.

in polynomial-time given the circuit C_m. Now if the circuit family $\{C_m\}$ is P-uniform, the entire construction becomes P-uniform, proving the uniform version of the Gap Theorem.

LEMMA 4.2. *For any* AC^0 *reduction computed by a family* $\{C_m\}$ *of circuits, there exists an* $a \in \mathbb{N}$ *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.*

The remainder of this section is devoted to proving Lemma 4.2.

4.1. Derandomizing the switching lemma. In this section we provide 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 each output bit of $C\lceil_\rho$ depends only 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. It will be convenient for us to consider a modified class of circuits, consisting of usual AND and OR gates, but at each "input" gate to the circuit we instead attach a decision tree; the circuit receives as input the value (0 or 1 or x_i) that is reached by querying the input bits specified by the decision tree and proceeding to a leaf of the decision tree. Thus an ordinary circuit corresponds to the case where we use decision trees of height zero. We will assume without loss of generality that C is arranged into d alternating levels of AND and ORs, and at the leaves of the circuit are constant-depth decision trees of height $\leq c_1$. The constant c_1 will be chosen to be sufficiently large as a function of k, where $n^k = S$ is the size of the original circuit. The proof will proceed in d steps. At step one, we will apply c_1 successive restrictions in order to replace the bottom levels of (ANDs of constant-depth-c_1 decision trees) by (constant-depth-c_2 decision trees), or similarly, in order to replace the bottom levels of (ORs of constant-depth-c_1 decision trees) by (constant-depth-c_2 decision trees). In general in step i, we will be applying c_i restrictions in order to replace the bottom levels of ANDs and ORs of constant-depth-c_i decision trees by depth-c_{i+1} decision trees. Thus after d steps, the total number of restrictions applied will be $c_1 + \cdots + c_d$, with c_i restrictions at step i, for d steps. The underlying variables will always be grouped into r blocks, where the block size will be $m = m_1$ at the start. After applying one restriction, we will still have r blocks, and exactly $m_1^{1/4}$ variables will remain unset within each block. (If a restriction is applied to r blocks, each of size m, then the restriction will consist of a first part where $rm^{1/2}$ variables are chosen uniformly to be set to $*$, and with the condition that no block will have size less than $m^{1/4}$, and then a second clean-up part where we set additional variables so that each block will have uniform size $m^{1/4}$.) Thus after one step, there will be r blocks, each of size $m_2 = m_1^{1/4^{c_1}}$, and finally after d steps, there will be r blocks, each of size $m_{d+1} = m_1^{1/4^{c_1+\cdots+c_d}}$.

We will now describe one step. Assume that the bottom level subcircuits have the form: AND of depth-c_1 decision trees. Then each such subcircuit can be expressed as an AND of size-c_1 ORs. Let $S_1, \ldots, S_q$ be the set of (polynomially many) ANDs of size-c_1 ORs. The first step proceeds in c_1 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'_1, and where each leaf is labeled by either a constant, or by an AND of size-$(c_1 - 1)$ 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 non-constant formulas labeling 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 labeled either by constants or by ANDs of size-$(c_1 - j)$ ORs. For each stage, we use Algorithm A to find the restriction. Finally after c_1 stages, we have decision trees for the original S_i's where all leaves are labeled by constants.

After one step, we have gone from a depth-d size-S circuit with rm_1 underlying variables, arranged into r blocks, where each block has size m_1, to a depth-$(d-1)$ size-S circuit, where now the number of underlying variables is rm_2, again arranged into r blocks, and where each block has size m_2. It is easy to see that now the bottom level consists of decision trees of depth $c'_1 c_1$, which will be chosen to be at most c_2. After repeating this for d steps, each output gate of the original circuit will be represented by a depth-c_{d+1} decision tree on the remaining variables. At the end, there will be rm_{d+1} remaining variables, again consisting of r blocks, each containing m_{d+1} variables.

We will now define the relationships between the various parameters. First, $c_1 = O(k)$, and for all $i \geq 2$, $c_i = 8^{c_{i-1}}$. For all $i \geq 1$, $c'_i = 6^{c_i}$. Thus, for each i, we have $c'_i c_i \leq c_{i+1}$ as required. Initially there are rm_1 variables. One restriction ρ will set all but $m_1^{1/4}$ variables per block. Thus after one restriction, there are $rm_1^{1/4}$ variables remaining, and after one step, there are rm_2 variables remaining (m_2 variables per block), where $m_2 = m_1^{1/4^{c_1}}$.

The number of variables remaining after d steps is $m_{d+1} = m_1^{1/4^{c_1+\cdots+c_d}} \geq m_1^{1/5^{c_d}}$. Recall that initially, there are r blocks, each of size $m_1 = r^a$ for some a, and we want it to be the case that after all restrictions are successfully applied to reduce the circuit, the final block size, m_{d+1}, is at least r^2. It should be clear that a can be chosen to be sufficiently large (depending on k and d) such that this holds.

We will now describe Algorithm A.

4.2. Algorithm A. The input to this algorithm is a collection of polynomially many formulas $Q_1, \ldots, Q_q$, 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 $b_1, \ldots, b_r$, 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 labeled by either a constant, or by an AND of size-$(c-1)$ ORs.

We follow the usual convention and refer to an OR of literals as a *clause*. Given a Q_i, we define a set Maxset(Q_i) of clauses 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 $f \log m$ (for a suitably chosen constant f whose value will be fixed later), then let Maxset(Q_i) be the lexicographically first $f \log m$ of these clauses. (So $|\text{Maxset}(Q_i)| \leq f \log m$.) We divide the Q_i's into two disjoint sets: First, $\{n_1, \ldots, n_s\}$, the *narrow* formulas, are those Q_i's such that $|\text{Maxset}(Q_i)| < f \log m$. Secondly, $\{w_1, \ldots, w_t\}$, the *wide* formulas, are those Q_i's such that $|\text{Maxset}(Q_i)| = f \log m$. We will find a restriction ρ setting all but $rm^{1/2}$ variables such that:

(1) ρ assigns at least $m^{1/4}$ $*$'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 all but exactly $m^{1/4}$ variables per block. Secondly, for each w_j, we can create the trivial decision tree for $w_j\lceil_\rho$ labeled 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 $\mathrm{Maxset}(n_i)\lceil_\rho$. 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 any other clause intersects at least one variable of $\mathrm{Maxset}(n_i)$, and all variables in $\mathrm{Maxset}(n_i)$ have been set.

The following three lemmas show that for suitable choices of the parameters, such a restriction ρ exists.

LEMMA 4.3. *Let $\{b_1, \ldots, b_r\}$ 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 $\sum_i \Pr[B_i] \le 1/4$, 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,

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

Summing up over all B_j shows that the total probability is at most $1/4$. □

We will apply the above lemma repeatedly, for smaller and smaller values of m. However, for each application, m will be equal to m_1^ϵ for some very tiny ϵ, which will be equal to r^δ for $\delta = l\epsilon$, and thus the above probability will always be less than $1/4$.

LEMMA 4.4. *Consider the set $\{s_1, \ldots, s_S\}$ where each s_i is a collection of at most $cf \log m$ literals, where S is a polynomial in m, and where the s_i's are pairwise disjoint. (For a given narrow formula n_i, s_i is the set of variables that underly the clauses in $\mathrm{Maxset}(n_i)$; since there are fewer than $f \log m$ clauses in $\mathrm{Maxset}(n_i)$, the total number of variables in s_i is at most $cf \log m$.) 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 have property (2) above.) Then as long as $S < m^{c'/4}$, $\sum_i \Pr[N_i] \le 1/4$.*

PROOF. Let rm be the original number of variables, and let $m' = r\sqrt{m}$ be the number of $*$'d variables in ρ. Then $p = m'/m = 1/\sqrt{m}$ 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 to $*$ is $|s_i|p \le O(\log m)p$.

The probability that there are more than c' $*$'s in s_i is at most

$$\binom{|s_i|}{c'} p^{c'} \le \left(\frac{e|s_i|p}{c'}\right)^{c'} = \left(\frac{ecf\log m}{c'\sqrt{m}}\right)^{c'} < (f\log m/\sqrt{m})^{c'}.$$

Since there are S many s_i's, the total probability of failure is at most

$$(f\log m/\sqrt{m})^{c'} S \le m^{-c'/4} S/4$$

for sufficiently large m. Thus, as long as $S < m^{c'/4}$, the overall probability is at most $1/4$. □

We will apply the above lemma repeatedly. At the start of each step i, $S \le n^k$, and at the end of stage j in step i, S will be bounded by $2^{jc'_i} n^k$. Thus in all cases where we apply the lemma, S will be bounded by $2^{c_{i+1}} n^k < 2^{c_{i+1}} m_1^{2k}$. At step i, m will be equal to m_i, c will be equal to c_i, and c' will be equal to c'_i. For our choices of parameters ($c'_i = 6^{c_i}$ and $m_i > m_1^{1/5^{c_i}}$), our condition that S be less than $m^{c'/4}$ thus holds if $2^{c_{i+1}} m_1^{2k} < m_1^{6/5^{c_i}/4}$, which holds for all large m_1.

Lemma 4.5. *Let* $\{w_1, \ldots, w_S\}$ *be ANDs of size-c ORs, where for each* w_i, $|\mathrm{Maxset}(w_i)| = f\log m$. *(The* w_i*'s are the wide formulas.) Let the underlying universe be of size* rm. *For a given* w_i, *let* W_i *be the bad event that no clause in* $\mathrm{Maxset}(w_i)$ *is set to zero. Then if* $S < e^{(f\log m)/4^c}/4$, *then* $\sum_i \Pr[W_i] \le 1/4$. *(That is, 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* $\mathrm{Maxset}(w_i)$ *is set to* 0 *by* ρ*.)*

Proof. For a given w_i, let $s_1, \ldots, s_{f\log m}$ denote the underlying (disjoint) clauses in $\mathrm{Maxset}(w_i)$. We will first show that for a given polynomial $p(m)$ there exists an f (depending on $p(m)$ and c) such that $\Pr[W_i] < 1/(4p(m))$:

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

Since the total number of w_i's is S, the total probability that some w_i does not have a clause in $\mathrm{Maxset}(w_i)$ that is set to 0, is at most $1/4$, by our choice of parameters. □

Once again, we will be applying the above lemma repeatedly for various values of c and m. At step i, m is equal to $m_i > m_1^{1/5^{c_i}} \geq m_1^{1/5^{c_d}}$, and in all cases $c \leq c_d$. In all applications, we will pick the constant f to be equal to c_{d+2}. As in our analysis of Lemma 4.4, in all applications S will be bounded by $2^{c_{d+1}} m_1^{2k}$. Hence,

$$S < 2^{c_{d+1}} m_1^{2k} < 2^{c_{d+1}} e^{2k \log m_1} < e^{(c_{d+2}/20^{c_d}) \log m_1} = e^{(f \log m_1^{1/5^{c_d}})/4^{c_d}} < e^{f \log m / 4^c}.$$

We now want to obtain a good ρ using the method of conditional probabilities (Alon & Spencer 1992). We will obtain ρ by choosing one element at a time to be set. That is, we first choose one of the rm variables and set it to 1 or 0; equivalently we 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 remains 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 every variable in H has already been set to 0 or 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 $s_1, \ldots, s_t$, $t = f \log m$, denote the $f \log m$ disjoint clauses in $\mathrm{Maxset}(w_k)$, each consisting of the OR of at most c disjoint literals. Recall that W_k 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 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 $I(x_1, y_1, \ldots, x_{f\log m}, y_{f\log m}, H)$ 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 $b = |H \cap (s_1 \cup \ldots \cup s_t)|$. 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 $x_1, y_1, \ldots, x_t, y_t$ of the following quantity, where the x_i's and y_i's satisfy: $x_1 \subset s_1$, $|x_1| \leq |s_1| - 1$, $y_1 \subset s_1$, $x_1 \cap y_1 = \emptyset$, $\ldots$, $x_t \subset s_t$, $|x_t| \leq |s_t| - 1$, $y_t \subset s_t$, $x_t \cap y_t = \emptyset$:

$$\begin{aligned} & I(x_1, y_1, \ldots, x_t, y_t, H) \\ & \times \binom{rm - |s_1| - \cdots - |s_t| - h + b}{rm - rm' - |x_1| - |y_1| - \cdots - |x_t| - |y_t| - h + b} \\ & \times 2^{rm-rm'-|x_1|-|y_1|-\cdots-|x_t|-|y_t|-h+b}. \end{aligned}$$

Since $|s_i| \leq c$, there are at most 2^c values for the variables x_i and y_i. Thus the total number of terms in the above summation is bounded by $2^{ct} = 2^{cf \log m}$, 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 stages, and at each stage the number of formulas under consideration is also polynomial.

5. Proof of Theorem 1.3

Let A be any AC^0-complete set for NP. Define two sets based on A:

$$E = \{x : (x = 10y \text{ and } y \in A) \text{ or } (10 \text{ is not a prefix of } x)\},$$
$$F = \{x : x = yz, \text{ and } z = \bar{y}, \text{ and } y \in A\}.$$

(Here, $\bar{y}$ denotes the binary string that is the bitwise complement of y.)

It is obvious that both of these sets are AC^0-complete for NP. Now suppose that these two sets are isomorphic to each other under isomorphism h, computed by a depth-two AC^0 circuit family. Let $\{D_n\}$ be the family of depth-two AC^0 circuits computing h that reduces F to E.

We first observe that $01\Sigma^* \subseteq E$. Therefore, $h^{-1}(01\Sigma^*) \subseteq F$. Since h^{-1} can blow up the size only polynomially, there exists a polynomial p such that for any n there is a number $m \leq p(n)$ such that the set $h^{-1}(01\Sigma^n) \cap \Sigma^{2m}$ contains at least $2^n/p(n)$ strings.

Choose a large enough n, and the corresponding m as above. Consider the circuit D_{2m}. Assume that D_{2m} has OR gates at the bottom level, and AND gates at the top. Observe that if, on an input x, the first (i.e., leftmost) output bit of D_{2m} is zero, then $h(x) \in E$, and therefore $x \in F$, which, in turn, means that $x = y\bar{y}$ for some string y of length m. Let this first output bit be denoted by ℓ. The subcircuit computing ℓ is an AND of ORs. Thus, ℓ can be written as

$$\ell = c_1 \wedge \cdots \wedge c_r$$

where each c_i is a disjunction of literals. We now claim that each c_i must contain all the $2m$ input variables. Suppose not. Let c_j be a disjunction not containing all the variables. Set all the variables occurring in c_j to make it evaluate to false. Therefore, $\ell = 0$. This implies that $x = y\bar{y}$ for some y as noted above. However, since not all bits of x are set, we can assign the unset bits a value so as to have $x \notin F$. Contradiction. Therefore, each of c_i must contain all the variables.

Now, ℓ would be zero for exactly r of the input strings where r is bounded by a polynomial in n. However, at least $2^n/p(n)$ strings must be mapped by D_{2m} to strings beginning with a zero. Since n was chosen to be large enough, this is a contradiction.

A similar argument can be given for the case when D_{2m} is an OR of ANDs using the second bit of the output of D_{2m} (whenever this bit is 1, the input must belong to F). Therefore, there is no depth-two AC^0 circuit family that computes an isomorphism between E and F.

6. Conclusions

Although Theorem 1.1 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-one 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 Agrawal *et al.* (1998) 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.

We especially call attention to the following problems:

1. Does the Berman–Hartmanis Conjecture hold for AC^0[mod 2] reductions? That is, are all of the sets that are complete under AC^0[mod 2] reductions isomorphic under AC^0[mod 2] isomorphisms?

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?

Very recently, Agrawal has provided a very strong affirmative answer to question 3: Every class $\mathcal{C}$ closed under Dlogtime-uniform TC^0 reductions has this property. More precisely, Agrawal (2001b) improves our Theorem 1.2 to replace the P-uniformity condition by L-uniformity, and then this is improved further in Agrawal (2001a) to achieve Dlogtime-uniformity.

Acknowledgements

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

A preliminary version of this work appeared in Proc. 29th ACM Symposium on Theory of Computing (STOC 1997).

Part of the first author's research was done while visiting the University of Ulm under an Alexander von Humboldt Fellowship. The research of the second author was supported in part by NSF grants CCR-9509603, CCR-9734918, and CCR-0104823. The research of the third author was supported by NSF Awards CCR-92-570979 and CCR-0098197, by Sloan Research Fellowship BR-3311, and USA-Israel BSF Grant 97-00188. The research of the third author was supported by NSF Award CCR-94-57782, and USA-Israel BSF Grant 95-00238.

References

M. Agrawal (2001a). The first-order isomorphism theorem. In *Proc. 21st Foundations of Software Technology and Theoretical Computer Science Conference (FST&TCS)*, Lecture Notes in Comput. Sci., Springer, to appear.

M. Agrawal (2001b). Towards uniform AC^0-isomorphisms. In *Proc. 16th IEEE Conference on Computational Complexity*, 13–20.

M. Agrawal, E. Allender & S. Rudich (1998). Reductions in circuit complexity: An isomorphism theorem and a gap theorem. *J. Comput. System Sci.* **57**, 127–143.

M. Ajtai (1983). Σ_1^1 formulae on finite structures. *Ann. Pure Appl. Logic* **24**, 1–48.

E. Allender (1989). P-uniform circuit complexity. *J. Assoc. Comput. Mach.* **36**, 912–928.

E. Allender & V. Gore (1991). Rudimentary reductions revisited. *Inform. Process. Lett.* **40**, 89–95.

N. Alon & J. Spencer (1992). *The Probabilistic Method.* Wiley.

J. Balcázar, J. Díaz & J. Gabarró (1995, 1990). *Structural Complexity Theory I and II.* Springer.

D. A. M. Barrington, N. Immerman & H. Straubing (1990). On uniformity within NC^1. *J. Comput. System Sci.* **41**, 274–306.

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

W. Eberly (1989). Very fast parallel polynomial arithmetic. *SIAM J. Comput.* **18**, 955–976.

G. Frandsen, M. Valence & D. M. Barrington (1994). Some results on uniform arithmetic circuit complexity. *Math. Systems Theory* **27**, 105–124.

M. Furst, J. B. Saxe & M. Sipser (1984). Parity, circuits, and the polynomial-time hierarchy. *Math. Systems Theory* **17**, 13–27.

J. Håstad (1987). One-way permutations in NC^0. *Inform. Process. Lett.* **26**, 153–155.

W. Hesse (2001). Division is in Uniform TC^0. In *Proc. Twenty-Eighth International Colloquium on Automata, Languages and Programming (ICALP)*, Lecture Notes in Comput. Sci. 2076, Springer, 104–114.

N. Immerman (1998). *Descriptive Complexity*. Graduate Texts in Computer Sci., Springer.

N. D. Jones (1975). Space-bounded reducibility among combinatorial problems. *J. Comput. System Sci.* **11**, 68–85.

S. Lindell (1992). A purely logical characterization of circuit uniformity. In *Proc. 7th IEEE Conference on Structure in Complexity Theory*, 185–192.

H. Veith (1998). Succinct representation, leaf languages, and projection reductions. *Inform. and Comput.* **142**, 207–236.

Manuscript received 30 September 1999

Manindra Agrawal
Department of Computer Science
Indian Institute of Technology
Kanpur, India
manindra@iitk.ac.in

Eric Allender
Department of Computer Science
Rutgers University
Piscataway, NJ, USA
allender@cs.rutgers.edu

Russell Impagliazzo
Department of Computer Science
University of California
San Diego, CA, USA
russell@cs.ucsd.edu

Toniann Pitassi
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
toni@cs.toronto.edu

Steven Rudich
Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA, USA
rudich@cs.cmu.edu