

The First-Order Isomorphism Theorem

Manindra Agrawal

Department of Computer Science
IIT Kanpur
Kanpur 208016, India
manindra@iitk.ac.in

Abstract. For any class \mathcal{C} und closed under NC^1 reductions, it is shown that all sets complete for \mathcal{C} under first-order (equivalently, Dlogtime-uniform AC^0) reductions are isomorphic under first-order computable isomorphisms.

1 Introduction

One of the long-standing conjecture about the structure of complete sets is the isomorphism conjecture (proposed in [BH77]) stating that all sets complete for NP under polynomial-time reductions are polynomial time isomorphic. As the conjecture cannot be resolved either way unless we discover non-relativizable techniques (see [KMR88,KMR89,FFK92] for more details), efforts have been made to prove the conjecture in restricted settings by restricting the power of reductions (see for example [Agr96,AAR98]). One of the most natural definition of restricted reductions is that of functions computed by *uniform constant-depth* (or AC^0) circuits (first studied in [CSV84]). These reductions provide the right notion of completeness for small complexity classes (logspace and below). Also, it has been observed that *natural* complete problems for various complexity classes remain complete under such reductions [IL95,Imm87]. Although the class of AC^0 functions is much smaller than the class of polynomial-time functions, it is interesting to note that, till recently, there was *no* known example of an NP-complete set that is not complete under uniform AC^0 reductions [AAI⁺97].

The notion of uniformity to be used with AC^0 circuits is widely accepted to be that of *Dlogtime-uniformity* (see Section 3 for definition). Under this uniformity condition, these circuits admit a number of different characterizations [BIS90,AG91]: functions computed by first-order logic formulae [Lin92], $O(1)$ -alternating log-time TMs [Sip83], logspace rudimentary predicates [Jon75] etc.

The isomorphism conjecture for complete sets for NP under AC^0 reductions has been studied before. Allender et. al. [ABI93] showed that all sets complete under *first-order projections* (these are very simple functions computed by uniform circuits with no gates [IL95]) are Dlogtime-uniform AC^0 -isomorphic (i.e., the isomorphism between any two such sets is computable in both directions by Dlogtime-uniform AC^0 circuits). This was improved in [AAR98] who showed that all sets complete under u -uniform (for any u) AC^0 reductions are non-uniform

AC^0 -isomorphic. Notice that this result proves the isomorphism conjecture for non-uniform AC^0 reductions but not for Dlogtime-uniform reductions. The uniformity condition for isomorphisms was improved first in [AAI⁺97] to P-uniform and then in [Agr01] to logspace-uniform thus proving the isomorphism conjecture for P-uniform and logspace-uniform AC^0 reductions respectively. More specifically, for all sets complete under u -uniform AC^0 reductions, [AAI⁺97] shows that they are $(u+P)$ -uniform AC^0 -isomorphic, while [Agr01] shows that they are $(u+\text{logspace})$ -uniform AC^0 -isomorphic. However, the conjecture remains open for Dlogtime-uniform AC^0 reductions, which is, in many ways, the correct formulation of the isomorphism conjecture for constant depth reductions.

In this paper, we prove that all complete sets for NP under u -uniform AC^0 reductions are $(u+\text{Dlogtime})$ -uniform AC^0 -isomorphic thus proving the isomorphism conjecture for constant depth reductions. Since there are a number of alternative characterizations of Dlogtime-uniform AC^0 circuits, this theorem can be viewed in many interesting ways, e.g., *all sets complete under first-order reductions are first-order isomorphic* (first-order functions are computed by first-order formulae). The above in fact holds for any class closed under TC^0 reductions.

The next section provides an outline of our proof. Section 3 contains definitions, and the subsequent sections are devoted to proving the result.

2 Proof Outline

The overall structure of the proof remains as given in [AAR98]. The proof in [AAR98] is a three stage one:

Stage 1 (Gap Theorem): This shows that all complete sets under u -uniform AC^0 reductions are also complete under *non-uniform* NC^0 reductions. This step is non-uniform.

Stage 2 (Superprojection Theorem): This proves that all complete sets under u -uniform NC^0 reductions are also complete $(u+P)$ -uniform *superprojections*, where superprojections are functions similar to projections. This step is P-uniform.

Stage 3 (Isomorphism Construction): This proves that all complete sets under u -uniform superprojections are isomorphic under $(u+\text{Dlogtime})$ -uniform AC^0 isomorphisms. This step is Dlogtime-uniform: starting with Dlogtime-uniform superprojections, one gets Dlogtime-uniform AC^0 isomorphisms.

The proof of Gap Theorem uses the Switching Lemma of [FSS84] in the construction of NC^0 reductions and is the reason for its non-uniformity. In [AAI⁺97] the lemma was derandomized using method of conditional probabilities making the stage P-uniform. Improving upon this, in [Agr01], the lemma was derandomized by constructing an appropriate *pseudo-random generator*. This made the stage logspace-uniform.

The Superprojection Theorem of [AAR98] uses the Sunflower Lemma of [ER60] which is P-uniform. This construction was replaced in [Agr01] by

a probabilistic construction that could be derandomized via an appropriate pseudo-random generator. This again resulted in a logspace-uniform construction.

Clearly, the uniformity of both these stages needs to be improved to obtain Dlogtime-uniformity. It is useful to note here that we need to make both the stages AC^0 -uniform only as that makes the isomorphism constructed by Stage 3 also AC^0 -uniform and then the AC^0 circuit used in uniformity can be incorporated in the AC^0 circuit for the isomorphism making the resulting AC^0 circuit Dlogtime-uniform. In fact this is the best that we can hope to do as it is known that the Gap Theorem *cannot* be made Dlogtime-uniform [AAR98].

We preserve the idea of [Agr01] of first giving a probabilistic construction and then derandomizing it via an appropriate pseudo-random generator in both the stages. The improvement in the uniformity condition of first stage is achieved by a careful construction of the pseudo-random generator needed that allows it to become AC^0 -uniform. The second stage presents a bigger problem. We replace the probabilistic construction used in [Agr01] by a more involved probabilistic construction and then derandomize it to obtain AC^0 -uniformity.

Combining the above constructions together with the Isomorphism Construction, we get Dlogtime-uniform AC^0 -isomorphisms.

3 Basic Definitions and Preliminaries

We assume familiarity with the basic notions of many-one reducibility as presented, for example, in [BDG88].

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, \dots, x_n (as well as the constants 0 and 1 allowed as inputs) and some number of output gates y_1, \dots, 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 $d(n)$.

For a circuit family $\{C_n\}$, the *connection set* of the family is defined as:

$$\text{Conn}_C = \{\langle n, t, i, j \rangle \mid \text{gate } i \text{ in } C_n \text{ is of type } t \text{ and takes input from gate } j\}.$$

A family $\{C_n\}$ is *u-uniform* if the connection set can be computed by a machine (or circuit) with a resource bound of u . In this paper, we will consider two notions of uniformity: Dlogtime-uniformity [BIS90] and AC^0 -uniformity. In the first, the connection set is computed by a TM with random access tapes working in $O(\log n)$ time (which is linear time as a function of input size), and in the second, the connection set is computed by an AC^0 circuit of polynomial size (which is exponential size in terms of input size). We will follow the standard convention that whenever the connection set is computed by a circuit family, the circuit family is assumed to be Dlogtime-uniform. So, for example, AC^0 -uniform means that the set can be computed by a Dlogtime-uniform AC^0 family of circuits.

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 $f(x)$. We will adopt the following specific convention for interpreting the output of such a circuit: 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 binary string contained in the first r output bits. It is easy to verify that this convention is AC^0 -equivalent to any other reasonable convention that allows for variable sized output, and for us it has the advantage that only $O(\log n)$ output bits are used to encode the length.

With this definition, the class of Dlogtime-uniform AC^0 -computable functions admits many alternative characterizations, including expressibility in first-order with $\{+, \times, \leq\}$, [Lin92, BIS90] the logspace-rudimentary reductions of Jones [Jon75, AG91], logarithmic-time alternating Turing machines with $O(1)$ alternations [BIS90] and others. This lends additional weight to our choice of this definition.

NC^0 is the class of functions computed in this way by circuit families of size $n^{O(1)}$ and depth $O(1)$, 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. An NC^0 function is a *projection* if its circuit family contains no AND or OR gates. For the sake of simplicity, we assume that NC^0 and projection functions *do not have variable sized output*. This may seem restrictive at a first glance, however, as we show later, that at least for complete sets we can ensure this property.

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} .

(A *language* is in a complexity class \mathcal{C} if its characteristic function is in \mathcal{C} . This convention allows us to avoid introducing additional notation such as FAC^0 , FNC^1 , etc. to distinguish between classes of languages and classes of functions.)

4 AC^0 -Uniform Gap Theorem

In this section, we prove the AC^0 -uniform version of the Gap Theorem of [AAR98]:

Theorem 1. *For any class \mathcal{C} closed under NC^1 reductions, all complete sets for \mathcal{C} under u -uniform AC^0 reductions are also complete under $(u + \text{AC}^0)$ -uniform NC^0 reductions.*

Proof. We begin by outlining the proof in [AAR98] and improvements of [Agr01] as we make use of both of them.

Fix a set A in \mathcal{C} that is complete under u -uniform AC^0 reductions and let $B \in \mathcal{C}$ be an arbitrary set. We need to show that B reduces to A via a $(u + \text{AC}^0)$ -uniform NC^0 reduction. We first define a set \hat{B} , which is a highly redundant version of B , as accepted by the following procedure:

On input y , let $y = 1^k 0z$. Reject if k does not divide $|z|$. Otherwise, break z into blocks of k consecutive bits each. Let these be $u_1 u_2 u_3 \cdots u_q$.

For each i , $1 \leq i \leq q$, let v_i be the parity of bits in u_i . Accept iff $v_1 v_2 \cdots v_q \in B$.

As one can readily observe, corresponding to a string in B there are infinitely many strings in \hat{B} . Also, \hat{B} reduces to B via an NC^1 reduction and so $\hat{B} \in \mathcal{C}$. Fix a reduction of \hat{B} to A given by u -uniform AC^0 circuit family $\{C_n\}$, say. Now define a reduction of B to \hat{B} as follows (it would be useful to keep the above definition of \hat{B} in mind while reading this definition):

Given an input x , $|x| = n$, let $m = n^t$ for an appropriate constant t to be fixed later. Consider the circuit $C_{m/n+1+m}$ with the first $m/n+1$ bits set to $1^{m/n}0$ resulting in circuit C'_m , say. Apply the Switching Lemma of [FSS84] on C'_m to obtain a setting of all but $\Omega(n \cdot (\log n)^2)$ input bits such that the circuit reduces to an NC^0 circuit and in addition, all the n blocks of $m/n = n^{t-1}$ consecutive bits in the input have at least $(\log n)^2$ unset bits (it was shown in [AAR98] that this can be ensured and this is what governs the choice of constant t). Now set all those unset bits to zero that influence at least one of the last $k \cdot \log n$ bits of the output (remember that these bits encode the length of the output as per our convention). This sets $O(\log n)$ additional unset bits. Since each block had $(\log n)^2$ unset bits to begin with, each block would still have at least two unset bits. Now for each of the n blocks, set all but one bits of the block to ensure that the number of ones in the block is 0 modulo 2 (this can also always be done as there is at least one unset bits available for setting). This sets all the m bits of input to C'_m except for n bits and on these n unset bits the circuit C'_m becomes an NC^0 circuit. Now map x to a string of length $m/n+1+m$ whose first $m/n+1$ bits are set to $1^{m/n}0$ and the remaining bits are set according to the above procedure and the i^{th} remaining unset bit is given the value of i^{th} bit of x .

It is easy to verify that the mapping constructed above is indeed a reduction of B to \hat{B} . Notice that this reduction is simply a *projection*: each input bit is mapped to some output bit directly and there are no gates in the circuit computing the reduction. It is also clear that a composition of this reduction with the reduction of \hat{B} to A is a reduction of B to A that can be computed by an NC^0 circuit family. The uniformity machine (or circuit) for this NC^0 circuit family is required to do the following tasks, apart from generating the circuit C'_m itself:

1. identify the settings of input bits to circuit C'_m that make the circuit an NC^0 circuit,
2. given such a setting, transform the circuit C'_m to the equivalent NC^0 circuit, and
3. set some of the unset bits as outlined above to leave only one unset bit in each block (in which string x would be placed).

The second task can be done by a Dlogtime -uniform AC^0 circuit that, for each output bit of the circuit C'_m , guesses the $O(1)$ input bits influencing the corresponding NC^0 circuit and then verifies this guess by evaluating C'_m on all

possible settings of these bits and noting if the chosen output bit of C'_m becomes constant for each setting or not.

For the third task, a Dlogtime-uniform AC^0 circuit can identify which unset bits influence the output bits coding length of the output, however, to set bits in a block appropriately (so that number of ones is 0 modulo 2), one requires a parity gate making the overall circuit an NC^1 circuit.

For the first task, we first note that according to the Switching Lemma of [FSS84] *most of the settings work*. In [AAI⁺97], a polynomial-time algorithm was given to identify one such setting given the circuit C'_m thus making the NC^0 circuit P-uniform. In [Agr01], a pseudo-random generator was constructed that stretches a seed of $O(\log n)$ bits to m bits such that on most of the strings output by the generator the circuit C'_m reduces to an NC^0 circuit. Using this, a uniformity machine can be constructed that first generates all possible $n^{O(1)}$ outputs of the generator and then, in parallel, checks which one of these is “good” by attempting to transform C'_m to an NC^0 circuit as outlined above. The power of the machine is decided by the difficulty of computing the generator. In [Agr01], the generator designed can be computed in logspace making the entire construction logspace-uniform.

To obtain AC^0 -uniformity, we need to improve upon both the first and third tasks. For the first task, one can try to obtain a generator that is computable by a Dlogtime-uniform AC^0 circuit. However, improving the third one seems impossible at the first glance as it is well known that computing parity of n bits cannot be done by even non-uniform AC^0 circuits [FSS84]. We solve this problem by a clever design of the generator: the generator would be such that it associates a sign (0 or 1) with each unset bit and the parity of all the set bits and signs of unset bits in a block is always zero! This trivializes the third task. The reduction of B to \hat{B} has to be changed slightly to make this work: map the i^{th} bit of x to the i^{th} unset bit if its sign is 0, else map it to the i^{th} unset bit by first complementing it.

We now give the generator construction of [Agr01] and then show how to improve it so that both the first and third tasks are completed. The generator is a combination of two types of primitive generators: (1) generators that produce bits that are $\frac{1}{n^{O(1)}}$ -biased, $O(\log n)$ -wise independent [NN90], and (2) generators, based on Nisan-Wigderson designs [NW94]. The generator consists of ℓ primitive generators of each type where ℓ is a constant dependent on the depth and size of the circuit C'_m . Also, each one of these primitive generators requires seed of length $O(\log n)$, and therefore, the seed length of the generator is $O(\log n)$. The generator is constructed as follows:

Let $G_{IND}^1, \dots, G_{IND}^\ell$ be ℓ primitive generators producing bits that are $\frac{1}{n^{O(1)}}$ -biased, $O(\log n)$ -wise independent, and $G_{NW}^1, \dots, G_{NW}^\ell$ be primitive generators based on NW-designs (their construction will be discussed later). The i^{th} bit of the generator G of [Agr01] is computed as: write i in binary and let $i = i_1 i_2 \dots i_{\ell+1}$ where $|i_j| = \frac{|i|}{2^j}$ for $1 \leq j \leq \ell$ (we assume $|i|$ to be a power of two to avoid complications). Compute bit $G_{NW}^j[i_j i_{j+1} \dots i_{\ell+1}]$ (we use $G[k]$ to denote the k^{th} bit of function G)

for $1 \leq j \leq \ell$. Let j_0 be the first j for which $G_{NW}^j[i_j i_{j+1} \cdots i_{\ell+1}]$ is zero. If there exists such a j_0 then let $G[i] = \bigoplus_{1 \leq j \leq j_0} G_{IND}^j[i_j i_{j+1} \cdots i_{\ell+1}]$. If there is no such j_0 then leave $G[i]$ unset and compute its sign as $\bigoplus_{1 \leq j \leq \ell} G_{IND}^j[i_j i_{j+1} \cdots i_{\ell+1}]$.

The following lemma was proved for this generator in [Agr01]¹:

Lemma 1. [Agr01] *Let C be any AC^0 circuit of a depth and size bounded by C'_m having m input bits. Then on at least half of the outputs of G , C reduces to an NC^0 circuit.*

It is clear from the construction of G above, that the computational resources required for G depend on the resources required to compute the two types of primitive generators. In particular, if both these types of primitive generators can be computed by Dlogtime-uniform AC^0 circuits, the generator G can also be computed by such circuits.

Let us now see constructions for these primitive generators. Three simple constructions of $\frac{1}{n^{O(1)}}$ -biased, $O(\log n)$ -wise independent generators are given in [AGHP90]. We choose one based on quadratic residues in a small field (in [Agr01] a different generator is used): i^{th} bit of $G_{IND}^j(s^j)$ is 1 iff the number $s^j + i$ is a quadratic non-residue in the field F_p with prime $p = n^{O(1)}$ (here s^j is the seed). This can be done by a Dlogtime-uniform AC^0 circuit: first an appropriate prime p is computed (fixing the field F_p), then s^j is added to i (addition is modulo p), and finally it is checked if there exists an x such that $x^2 = s^j + i$. All these computations can be done by Dlogtime-uniform AC^0 circuits as the field size is small (as shown in [BIS90]).

The generator G_{NW}^j is defined as: let $i = i_1 i_2$ with $|i_1| = |i_2| = k$; let seed $s^j = s_1^j s_2^j \cdots s_c^j$ with $|s_1^j| = |s_2^j| = \cdots = |s_c^j| = k$ for appropriate constant c ; compute $i' = \sum_{e=1}^c s_e^j \cdot (i_2)^{e-1}$ where all the operations are over field F_{2^k} , and set the i^{th} bit to 1 iff $i' = i_1$. Again, all the computations here can be done by a Dlogtime-uniform AC^0 circuit (shown in [BIS90]).

Thus the generator G can be computed by a Dlogtime-uniform AC^0 circuit. The primitive generator G_{NW}^1 sets exactly one bit to 1 in consecutive blocks of $m^{\frac{1}{2}}$ bits, the generator G_{NW}^2 sets exactly one bit to 1 in consecutive blocks of $m^{\frac{1}{4}}$ unset bits remaining, etc. Thus the generator G leaves exactly unset one bit in consecutive blocks of $m^{\frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^t}} = m^{1 - \frac{1}{2^t}}$ bits which makes a total of $m^{\frac{1}{2^t}}$ unset bits. Now choosing $t = 2^{\ell+1}$ (recall that $m = n^t$) ensures that each of the n blocks contains exactly $n \geq (\log n)^2$ unset bits as required.

However, this generator *does not* guarantee that the parity of all the set bits and signs of the unset bits in a block is zero (the length of a block is $m/n = n^{2^{\ell+1} - 1}$). To achieve this, we change the definition of generators G_{IND}^j in such a way that parity of all the bits that it contributes to setting of bits and signs

¹ Actually, in [Agr01] the generator construction is slightly different: $G[i]$ is simply set to $G_{IND}^{j_0}[i_{j_0} \cdots i_{\ell+1}]$ when j_0 exists. However, the lemma holds for this modification too, and this modification makes our current construction simpler.

in a block is zero. Notice that the generator G_{IND}^j contributes exactly $n^{2^{\ell+2-j}-1}$ bits to a block. Change the generator G_{IND}^j by replacing every $(f \cdot \log n)^{th}$ bit by the parity of previous $f \cdot \log n - 1$ bits for a large enough constant f (f should be chosen so that it is power of two and the generator is required to be $f' \cdot \log n$ -wise independent for $f' < f$). Without loss of generality, we can assume that n is of the form 2^{2^u} and then, since f is a power of two, $f \cdot \log n$ would divide n . This ensures that the parity of all the bits contributed by the modified generator to a block is zero. However, we now need to show that the modified generator is still $\frac{1}{n^{O(1)}}$ -biased, $O(\log n)$ -wise independent. This follows immediately from the fact that any set of $< f \cdot \log n$ bits of the modified generator is still $\frac{1}{n^{O(1)}}$ -biased (follows from [AGHP90]), and therefore these bits are independent with a similar bias (shown in [NN90]). Thus with these modified primitive generators, the generator G satisfies all the required conditions proving the theorem. \square

5 AC⁰-Uniform Superprojection Theorem

We start with the definition of a superprojection [AAR98].

Definition 1. *An NC⁰ 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.*

Now we prove the AC⁰-uniform Superprojection Theorem:

Theorem 2. *For any class \mathcal{C} closed under NC¹ reductions, all complete sets for \mathcal{C} under u -uniform NC⁰ reductions are also complete under $(u + AC^0)$ -uniform superprojections.*

Proof. Fix a set A in \mathcal{C} that is complete under u -uniform NC⁰ reductions and let $B \in \mathcal{C}$ be an arbitrary set. We need to show that B reduces to A via a $(u + AC^0)$ -uniform superprojection. We first define, as before, a set \hat{B} as accepted by the following procedure:

On input y let $y = z'11z$ such that $z' \in \{00, 01, 10\}^*$. Break z' into pairs of bits. Ignoring all the 00 pairs, consider the first $\log |z|$ pairs. Define number k by setting i^{th} bit of k to 1 if the i^{th} of the above $\log |z|$ pairs is 10, to 0 otherwise. Reject if k does not divide $|z|$. Else, break z into blocks of k consecutive bits each. Reject if the number of blocks is not a multiple of four. Else, let $z = u_1u_2u_3 \cdots u_{4q}$ with $|u_i| = k$. Let v_i be the parity of bits in u_i . Let $w_i = v_{4i-3}v_{4i-2}v_{4i-1}v_{4i}$ for $1 \leq i \leq q$ (so each w_i is a four bit string). If $w_i = 1111$ for any $1 \leq i \leq q$, accept. Else if some w_i has exactly three ones, reject. Else, for each i , $1 \leq i \leq q$, let $b_i = 1$ if w_i has exactly two ones, $b_i = 0$ if w_i has exactly one one, $b_i = \epsilon$ otherwise. Accept iff $b_1b_2 \cdots b_q \in B$.

The definition of set \hat{B} is more complicated than the previous one. Even the block size ($= k$) is coded in the string in a non-straightforward way. We refer to

the bits of z' of any instance y of \hat{B} as *length encoder bits* and to the bits of z as *string encoder bits*. It is easy to see that \hat{B} reduces to B via an NC^1 reduction and so $\hat{B} \in \mathcal{C}$. Fix a reduction of \hat{B} to A given by u -uniform NC^0 circuit family $\{C_n\}$, say. Let each output bit of any circuit C_n depend on at most c input bits.

As before, we now define a reduction of B to \hat{B} . The idea is same: for an appropriate m and ℓ , consider the circuit $C_{\ell+2+m}$. Set some of the input bits of $C_{\ell+2+m}$ so that the circuit on remaining unset bits is a superprojection. Now set some more bits (including all of length encoder bits) to satisfy all the conditions in the definition of set \hat{B} and finally map string x to remaining unset bit positions. In [Agr01], a simple random construction for this was given and then the construction was derandomized using an appropriate generator. However, the random construction did not guarantee that in every block at least one unset bit would be present after all the settings (that is why we need to have “empty” blocks for which $b_i = \epsilon$). This makes the mapping of bits of x difficult as we need to use threshold gates to find the i^{th} unset bit. This was not the case in the previous proof as every block there had an unset bit and so the i^{th} unset bit can be identified by using an AC^0 circuit on the bits of the i^{th} block.

We give a different construction to solve this problem. Interestingly, our construction uses the central idea of the Switching Lemma proof of [FSS84] which is also used in the construction in Gap Theorem.

We first discuss a simple idea (one that is used in [Agr01]) and see why it does not work.

Consider circuit $C_{\ell+2+m}$. Randomly set every input bit of the circuit to 0 or 1 with probability $\frac{1}{4}$ each leaving it unset with probability $\frac{1}{2}$. Say that an input bit in string encoder part is *good* if it remains unset and there is at least one output bit that now depends *only on this bit*. For any input bit that influences some output bit in $C_{\ell+2+m}$, the probability that this bit is good is at least $\frac{1}{2} \cdot (\frac{1}{4})^{c-1} > \frac{1}{4^c}$. Therefore, the expected number of good input bits is $\Omega(m')$ where m' is the number of input bits in the string encoder part of $C_{\ell+2+m}$ that influence at least one output bit. Identify all the good bits and set all the other unset input bits appropriately. This makes the circuit $C_{\ell+2+m}$ on the remaining unset bits a superprojection.

The above construction yields $\Omega(m)$ good bits provided we can ensure that nearly all the input bits influence the output (part of the complexity in definition of \hat{B} is due to this requirement). The construction can easily be derandomized by using a $2c$ -wise independent generator for selecting unset bits and setting remaining bits. However, the problem pointed out earlier—it cannot be ensured that every block has at least one unset bit—remains.

In our construction, we *do* use the above construction, but only after making sure that every block is guaranteed to have at least one unset bit. For this, we successively shrink the block size simultaneously making “bad” blocks (i.e., those that do not have unset bits) “empty.” This is why we cannot fix the block size in the beginning of the construction unlike the previous proof.

Let x , $|x| = n$, be an instance of B . Let $m = (4n^2)^c$. Consider the circuit $C_{4^{4c} \log m + 2 + m}$. To begin with, set the bit numbers $4^{4c} \log m + 1$ and $4^{4c} \log m + 2$ of the input to $C_{4^{4c} \log m + 2 + m}$ to 1 identifying the first $4^{4c} \log m$ bits as length encoder and the last m bits as string encoder bits. Let C be the resulting circuit.

Split the string encoder bits of the input to C into $4n$ block of equal size ($= n \cdot (4n^2)^{c-1}$). Firstly, we notice that *every bit in every block* must influence some output bit. Suppose not. Let such a bit belong to $(4i + j)^{th}$ block. Set all the bits in all the blocks except for block numbers $4i + 1$ through $4i + 4$ so that the parity of bits in every block is zero. Set bits in blocks $4i + 1$ through $4i + 4$ except those in block $4i + j$ such that parity of bits in these blocks is one. Set all the bits in the block $4i + j$ except the the bit that does not influence any output bit so that the parity of set bits is zero. This fixes the output of circuit C . However, the value of the lone unset bit decides whether the input string belongs to the set \hat{B} or not, contradicting the fact that family $\{C_n\}$ computes a reduction of \hat{B} to A .

Apply a random restriction to input bits of C as outlined above using a $\frac{1}{n^2}$ -biased, $4^{3c} \log n$ -wise independent source to generate the restriction (instead of a $2c$ -wise independent generator—the reason for this would be clear soon). There are two cases that can arise now:

Case 1. For every seed value of the generator, there is at least one block with no good bit.

Case 2. There is a seed of the generator that leaves at least one good bit in every block.

We tackle Case 1 first. Undo the above random restriction. Divide each block into n sub-blocks of equal size ($= (4n^2)^{c-1}$). For each sub-block, do the following experiment: set all other bits in all the other sub-blocks and blocks to zero, and then see if by setting an additional $4^{3c} \log n$ length encoder bits to zero all the output bits of the circuit C now depend only on at most $c - 1$ unset bits. We show later that there *must* exist such a sub-block. Fix any such sub-block. Set all bits in all other sub-blocks to zero and also those length encoder bits identified for this sub-block resulting in a circuit whose every output bit depends only on at most $c - 1$ input bits. For each length encoder bit set to zero, set its paired bit also to zero (rendering these pairs ineffectual for encoding length). We now are left with exactly $(4n^2)^{c-1}$ unset string encoder bits and at least $4^{4c} \log n - 2 \cdot 4^{3c} \log n$ unset length encoder bits.

Apply the random restriction on these bits and repeat the same process. If Case 1 keeps occurring, after $c - 1$ iterations, we would be left with exactly $4n^2$ unset string encoder bits and at least $(4^{4c} - 2(c-1) \cdot 4^{3c}) \cdot \log n \geq 2c \cdot \log 2n = \log m$ length encoder bits. And the circuit C is simply a projection on these unset bits! We can now fix the length encoder bits to code the block length as 1, set all the remaining length encoder bits to zero, set all but first $4n$ of unset string encoder bits to zero, set last three bits in every group of 4 unset bits to 100 and map the i^{th} bit of x to the first bit of i^{th} group. This defines a projection reduction of B to \hat{B} and on outputs of this reduction, circuit C is also a projection. Therefore,

their composition is a projection (we shall see later that this composition can be computed by an AC^0 -uniform circuit).

The other possibility is that after some iterations, Case 2 occurs. In that case, identify the seed on which the generator output leaves at least one good bit in each block. Set the length encoder bits to code the current block length (we argue later that this can always be done). In every block, set all the bits except one good bit to zero. Now map the string x to these good bits as above. Use the modified generator G_{IND}^1 of previous proof so that the parity of all the set bits plus the signs in each block is zero (to incorporate the sign, we just need to xor it with the settings of all the unset bits). This defines a projection reduction of B to \hat{B} whose composition with C is a superprojection.

There are two things remaining to be done: (1) we need to show that when Case 1 occurs then there exists a sub-block with the desired properties and when Case 2 occurs then there are enough unset length encoder bit pairs are available, (2) we need to show that the above construction can be done by a Dlogtime-uniform AC^0 circuit. The second is easy to show: we have already seen that the generator output can be computed by Dlogtime-uniform AC^0 circuit. The remaining tasks can easily be done by a Dlogtime-uniform AC^0 circuit.

To show the first, we make use of the central idea in [FSS84]. For j^{th} block, let o_1, \dots, o_p be *all* the output bits of C that depend on some bit in the block. For output bit o_i , let I_i be the set of input bits that influence o_i . Clearly, $|I_i| \leq c$. On a random restriction as defined above, the probability that a bit in j^{th} block belonging to I_i becomes good due to I_i is at least $\frac{1}{4^c}$. Let MaxSet be any *maximal* set of disjoint I_i s. If $|\text{MaxSet}| \geq 4^{2c} \log n$ then drop some I_i s from it to make $|\text{MaxSet}| = 4^{2c} \log n$. Since the restriction bits are $4^{3c} \log n$ -wise independent (with a small bias, of course) and MaxSet contains at most $c \cdot 4^{2c} \log n < 4^{3c} \log n$ bits, the probability that at least one of the bits in the j^{th} block belonging to *some* I_i in MaxSet becomes good is at least $1 - (1 - \frac{1}{4^c})^{4^{2c} \log n} - \frac{1}{n^2} > 1 - \frac{1}{8n}$. If each one of $4n$ blocks has this property, then the probability that each one of them has at least one good bit is at least $\frac{1}{2}$. The same calculation works when we drop from the set MaxSet those I_i s that contain a bit from the first $\log m$ unset pairs of bits of length encoder bits (we drop at most $4c \log 2n$ I_i s). This is the Case 2: we can keep sufficient number of length encoder bit pairs unset and still have every block having at least one good bit.

Now consider the other possibility: there is a block with $|\text{MaxSet}| < 4^{2c} \log n$. Divide this block into n sub-blocks of equal size as described above. Clearly, one of these sub-blocks will contain *no* bit that belongs to MaxSet, since MaxSet has less than $c \cdot 4^{2c} \log n$ bits. Fix a sub-block that does not intersect with MaxSet. Now if we set all the bits of all the other blocks and sub-blocks and also at most $c \cdot 4^{2c} \log n < 4^{3c} \log n$ length encoder bits (the ones that belong to MaxSet), all the bits in MaxSet would be set and this would mean that each I_i contains at most $c - 1$ unset bits. This is Case 1! \square

References

- AAI⁺97. M. Agrawal, E. Allender, R. Impagliazzio, T. Pitassi, and S. Rudich. Reducing the complexity of reductions. In *Proceedings of Annual ACM Symposium on the Theory of Computing*, pages 730–738, 1997.
- AAR98. M. Agrawal, E. Allender, and S. Rudich. Reductions in circuit complexity: An isomorphism theorem and a gap theorem. *J. Comput. Sys. Sci.*, 57:127–143, 1998.
- ABI93. E. Allender, J. Balcázar, and N. Immerman. A first-order isomorphism theorem. In *Proceedings of the Symposium on Theoretical Aspects of Computer Science*, 1993.
- AG91. E. Allender and V. Gore. Rudimentary reductions revisited. *Information Processing Letters*, 40:89–95, 1991.
- AGHP90. N. Alon, O. Goldreich, J. Hastad, and R. Peralta. Simple constructions of almost k -wise independent random variables. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science*, pages 544–553, 1990.
- Agr96. M. Agrawal. On the isomorphism problem for weak reducibilities. *J. Comput. Sys. Sci.*, 53(2):267–282, 1996.
- Agr01. M. Agrawal. Towards uniform AC^0 isomorphisms. In *Proceedings of the Conference on Computational Complexity*, 2001. to be presented.
- BDG88. J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1988.
- BH77. L. Berman and J. Hartmanis. On isomorphism and density of NP and other complete sets. *SIAM Journal on Computing*, 1:305–322, 1977.
- BIS90. D. Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1 . *J. Comput. Sys. Sci.*, 74:274–306, 1990.
- CSV84. A. Chandra, L. Stockmeyer, and U. Vishkin. Constant depth reducibility. *SIAM Journal on Computing*, 13:423–439, 1984.
- ER60. P. Erdős and R. Rado. Intersection theorems for systems of sets. *J. London Math. Soc.*, 35:85–90, 1960.
- FFK92. S. Fenner, L. Fortnow, and S. Kurtz. The isomorphism conjecture holds relative to an oracle. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science*, pages 30–39, 1992. To appear in SIAM J. Comput.
- FSS84. M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial hierarchy. *Mathematical Systems Theory*, 17:13–27, 1984.
- IL95. N. Immerman and S. Landau. The complexity of iterated multiplication. *Information and Computation*, 116:103–116, 1995.
- Imm87. N. Immerman. Languages that capture complexity classes. *SIAM Journal on Computing*, 16:760–778, 1987.
- Jon75. N. Jones. Space-bounded reducibility among combinatorial problems. *J. Comput. Sys. Sci.*, 11:68–85, 1975.
- KMR88. S. Kurtz, S. Mahaney, and J. Royer. The structure of complete degrees. In A. Selman, editor, *Complexity Theory Retrospective*, pages 108–146. Springer-Verlag, 1988.
- KMR89. S. Kurtz, S. Mahaney, and J. Royer. The isomorphism conjecture fails relative to a random oracle. In *Proceedings of Annual ACM Symposium on the Theory of Computing*, pages 157–166, 1989.
- Lin92. S. Lindell. A purely logical characterization of circuit complexity. In *Proceedings of the Structure in Complexity Theory Conference*, pages 185–192, 1992.

- NN90. J. Naor and M. Naor. Small-bias probability spaces: Efficient constructions and applications. In *Proceedings of Annual ACM Symposium on the Theory of Computing*, pages 213–223, 1990.
- NW94. N. Nisan and A. Wigderson. Hardness vs. randomness. *J. Comput. Sys. Sci.*, 49(2):149–167, 1994.
- Sip83. M. Sipser. Borel sets and circuit complexity. In *Proceedings of Annual ACM Symposium on the Theory of Computing*, pages 61–69, 1983.