Proof Verification in Constant Depth

Shafi Goldwasser
MIT and Weizmann Institute
shafi@theory.csail.mit.edu

Dan Gutfreund
SEAS, Harvard University
danny@eecs.harvard.edu

Alexander Healy
SEAS, Harvard University
ahealy@fas.harvard.edu

Tali Kaufman
CSAIL, MIT
kaufmant@mit.edu

Guy N. Rothblum
CSAIL, MIT
rothblum@csail.mit.edu

Abstract

We consider the task of proof verification by weak constant-depth verifiers. Working in the setting of interactive proofs, we show that (p-prover) interactive proofs with k rounds of interaction are equivalent to (p-prover) interactive proofs with $k+O(1)$ rounds, where the verifier is in $\mathcal{NC}^0$. That is, each round of the verifier's computation can be implemented in constant parallel time. As a corollary, we obtain interactive proof systems, with (optimally) constant soundness, for languages in $\mathcal{AM}$ and $\mathcal{NEXP}$, where the verifier runs in constant parallel-time.

The key idea we use is that of *delegating* some of the verifier's computation to the (potentially malicious) prover. The challenge is verifying that the delegated computation was performed correctly. This idea was introduced by Goldwasser *et al.* [STOC08] in the area of program checking, and also used in [STOC07] to design very efficiently decodable error-correcting codes.

Contents

1 Introduction

Proof verification is a central concept in theoretical computer science. In this setting, a computationally powerful and possibly malicious party, the *prover*, interacts with a weak and honest party, the *verifier*. The prover makes a claim (e.g., that a given graph is 3-colorable), and tries to convince the verifier that this claim is valid. The goal is to design a (possibly randomized and/or interactive) proof system such that the verifier is only convinced when the prover's claim is in fact correct, even when the prover behaves maliciously.

Seminal works in complexity theory have uncovered the striking power of interactive and randomized proof systems as introduced by Babai [Bab85] and Goldwasser, Micali and Rackoff [GMR89]. The works of Shamir [Sha92] (building on the work of Lund, Fortnow, Karloff and Nisan [LFKN92]), and Babai, Fortnow and Lund [BFL91], give efficient proof systems for every language that could conceivably have them. The works of Babai and Moran [BM88], Goldwasser and Sipser [GS86], Feige and Lovasz [FL92], and the PCP literature, show that interactive proof systems remain very powerful even when various limitations are placed on the verifier or on the protocol.

In this work we continue this line of research, exploring the power of proof systems with verifiers that have very limited computational resources. We show that proof systems remain powerful even when the verifier is severely weakened computationally to run in $\mathcal{NC}^0$ (constant parallel time). Beyond their complexity-theoretic interest, these proof systems could be used for delegating computation efficiently and reliably: a device that runs in constant parallel time can verify the outcomes of much more complex computations.

This is but one instance of setting in which a computationally powerful but possibly malicious *sender* interacts with a weak and honest *receiver*. Here the sender is the (potentially malicious) prover, and the receiver is the (weak) verifier. We use the approach of improving the receiver's efficiency by *delegating* most of its computational work to the sender. It is not immediately apparent how to do this, as the sender may be malicious, and thus we introduce tools for *reliably* delegating computation to a potentially malicious party. A similar approach was applied first in the setting of program checking by [GGH$^+$08], and in [GGH$^+$07] for obtaining efficiently decodable error-correcting codes.

Main Results We consider proof systems with extremely weak verifiers: i.e., verifiers in $\mathcal{NC}^0$ (or constant parallel time). By that we mean that the verifier's strategy at each interaction round can be computed in $\mathcal{NC}^0$ when given access to the input, the randomness, and the messages exchanged in the previous rounds. We show that such proof systems are surprisingly powerful: essentially, anything that is provable in k communication rounds with a polynomial-time verifier is also provable in $k + O(1)$ communication rounds with an $\mathcal{NC}^0$ verifier.[1] In particular, we obtain the following characterizations:

1. A language is in $\mathcal{AM}$ (the class of languages that have a proof system with a constant number of communication rounds) if and only if it has a single-prover, two-round, constant-soundness

[1]We observe in Section 4 that by adding $O(\log n)$ communication rounds it is not hard to transform any protocol into one with an $\mathcal{NC}^0$ verifier. However, we achieve this while only adding a *constant* number of communication rounds. this is what enables us to obtain constant parallel time verification for languages in $\mathcal{AM}$ and $\mathcal{NEXP}$.

interactive proof that can be verified in constant parallel time (Corollary 3.7).[2] In particular, every language in $\mathcal{NP}$ has such a proof system.

2. A language is in $\mathcal{NEXP}$ if and only if it has a two-prover, five-round interactive proof of constant soundness, that can be verified in constant parallel time (Theorem 3.10).

Previous proof systems for complete languages in $\mathcal{NP}$, $\mathcal{IP}$ and $\mathcal{NEXP}$ require, at the very least, the verification of an $\mathcal{NP}$ statement at the end of the protocol (even to achieve constant soundness). By the Cook-Levin reduction, this verification is very efficient (i.e. in $\mathcal{AC}^0$); indeed, a key point in the Cook-Levin theorem is that computation can be verified by making many local consistency checks and ensuring that they all hold. However, while the local checks are of constant size, the verifier still needs to verify that *all of them hold* by computing an AND of large fan-in, and therefore is not in $\mathcal{NC}^0$. We show that, perhaps surprisingly, a verifier can use interaction with the prover together with its (*private*) random coins to avoid performing a global test on its entire input and proof. This is done by replacing the global test with a test that the prover is not cheating. The size (fan-in) of this new test is only a function of the soundness, independent of the size of the input.[3]

Negative results We complement our positive results with two negative results. First, we show that constant-round proof systems with an $\mathcal{NC}^0$ verifier cannot have sub-constant soundness (unless the language itself is in $\mathcal{NC}^0$). Second, we show that there is no public-coin proof system with an $\mathcal{NC}^0$ verifier (again, unless the language itself is in $\mathcal{NC}^0$). This result sheds light on private vs. public coins proof systems and in particular on our protocols (which, naturally, use private coins). In particular, it shows that both *interaction* and *private randomness* are *provably essential* for non-trivial $\mathcal{NC}^0$ verification.

Our approach. We improve the receiver's (verifier's) efficiency by delegating some of its computation to the (possibly malicious) sender (the prover). The idea of delegating computation from the receiver to the untrusted sender seems dubious at first glance, as the receiver's computation is the only reliable part in the whole interaction; indeed, this seems to leave the receiver very vulnerable to malicious behavior of the sender. To give the receiver a better guarantee, we ask more from the sender: we ask the sender to convince the receiver that he has performed the computations correctly (in the case of proof verification), or to send the results of the computations with redundancy that will allow the receiver to easily recover the correct results even from a corrupted word (in the case of codes). This may seem to bring us back to square one; namely, the receiver again needs to verify a proof or to decode a code. So where do we gain in efficiency? The key point is that we are not trying now to verify an arbitrary claim, or to recover arbitrary information, but rather we are trying to make sure that a certain *computation* was conducted correctly. Here one

[2]Here we follow the standard convention that measures the complexity of the protocol only in terms of the resources used by the verifier, i.e. it is assumed that the prover's messages are generated instantly. Thus, our statement about constant parallel-time verification follows from the fact that the protocols we construct have a constant number of communication rounds and that each round the verifier's strategy can be implemented in constant parallel time.

[3]Although we emphasize the locality of the verifier, one should not confuse our verifiers with PCP verifiers. While the latter do look at a constant number of bits in the proof, they still need to check consistency with the whole input (e.g. when computing the PCP reduction) and therefore are not in $\mathcal{NC}^0$ (as a function of the input, the proof and the randomness). In fact, our lower bounds (see Section 4) show that non-trivial languages *cannot* have $\mathcal{NC}^0$ PCP verifiers (roughly speaking, this is because PCPs are not interactive).

can see a connection to program checking and correcting that we discuss below. Specifically, we show that if the receiver's computations have certain properties, which we discuss shortly, then the tasks of verifying their correctness or decoding the correct results of the computations can be done extremely efficiently – much more efficiently than the receiver's original computation. To develop our approach we look at functions that the receiver needs to compute and require them to have two properties:

1. (Random instance reduction) One can compute the function on any given instance by querying another function (say g) at a completely random location. This property allows us to "mask" the receiver's computation as a random instance and to correct the sender's computations.

2. (Solved instance generator) We can generate efficiently a random instance of the function g together with g's value on this instance. This property allows us to check the correctness of the sender's computations.

Combining these properties allows us to ensure (w.h.p.) that the computations that the sender conducts for the receiver are indeed correct. Of course, this approach would not give us much if we could not show that the above properties can be implemented more efficiently than the original computations. To that end we show, using techniques that were developed in the field of cryptography [Bab87, Kil88, FKN94, IK02, AIK06], that for functions computable in $\mathcal{NC}^1$ these properties can be implemented in probabilistic constant parallel time. Thus, we can take any $\mathcal{NC}^1$ receiver and transform it into one that runs in constant parallel time or constant depth. This reduces our task to finding a sender-receiver protocol for the required task in which the receiver is in $\mathcal{NC}^1$.

Related Work Our approach for improving the receiver's efficiency by delegating some of its computation to the (possibly malicious) sender, is based on a delegation methodology and tools developed in a recent work of Goldwasser et al. on improving the efficiency of program checkers [GGH$^+$08], and also inspired by the work of Applebaum, Ishai and Kushilevitz [AIK06] on improving the efficiency of cryptographic primitives. A discussion about the similarities and differences between these works and the results presented here follows.

[GGH$^+$08] develops a methodology of delegating computation as a way to increase the efficiency of program checkers (see [BK95]) and program testers/correctors (see [BLR93]). This idea plays a key role in their general approach of composing program checkers (and testers/correctors).

In fact, one can view program checking as an interactive proof setting where the prover is analogous to the program and the checker is analogous to a verifier. The prover in the program checking setting is fixed in advance and restricted to computing only the language being proved, as opposed to being computationally unbounded and dynamic (according to the messages exchanged in the protocol) in the usual proof verification setting. Moreover, a program checker for a language gives such a proof system both for the language and for its complement. These differences give rise to different challenges in the design of such protocols, and in particular in the implementation of the delegation methodology. The fact that the prover is restricted to answering queries about the language being proved, in the case of program checkers, requires careful design of such protocols that typically use very specific properties of the functions being proved (checked). In fact, it is not at all well understood which languages have such proof systems. [GGH$^+$08] gave both a methodology for constructing such systems with very efficient verification (checking) and a family of results for a wide variety of languages. While in this work we consider the (easier) setting of an unbounded prover, we (as opposed to [GGH$^+$08]) must deal with the challenge that the prover may change its answers according to the messages exchanged in the interaction. For example, in our setting one cannot design proof systems that first test the prover on random inputs, and then correct it (which is a common methodology for constructing program checkers).

The results in [GGH$^+$07] on constructing efficient error-correcting codes are also related to our results and the results in [GGH$^+$08] on building efficient program correctors. Again, in that setting they design decoders that delegate work to an encoder that can perform arbitrary (efficient) computations. the challenge is for the encoder to relay to the decoder computationally helpful information over a noisy channel in an error-robust way. There is a guarantee, though, that the channel is not arbitrarily malicious and the noise rate is bounded. The main challenge is then recovering from very large fractions of errors (especially in the list-decoding setting).

The work of [AIK06] can also be viewed as improving the efficiency of players participating in a protocol by pushing computation from one of the participants to another (e.g. improving the efficiency of encryption at the expense of adding to the complexity of decryption). The main difference between this approach and ours is that they consider protocols or objects in which the *goal* of the sender is to reveal the results of its computation to a receiver, so there is no issue of a malicious party. In our case, on the other hand, the sender is *untrusted* but the receiver still wants it to perform computations for him. Given these differences, the tools we use are somewhat different from those used in [AIK06]. Nonetheless, some of the techniques we employ are similar.

Organization. In Section 2 we give some general preliminaries and specific background about interactive proofs. In Section 2.4 we develop the tools that allow us to implement our approach. Section 3 shows how to transform general proof systems to ones with verifiers in $\mathcal{NC}^0$. In Section 4 we present our negative results.

2 Preliminaries and main tools

For a string $x \in \Sigma^*$ (where Σ is some finite alphabet) we denote by $|x|$ the length of the string, and by x_i or $x[i]$ the i'th symbol in the string. For a finite set S we denote by $y \in_R S$ that y is a uniformly distributed sample from S. For $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, 2, \ldots, n\}$. For a finite alphabet Γ we denote by Δ_Γ the relative (or fractional) Hamming distance between strings over Γ. That is, let $x, y \in \Gamma^n$ then $\Delta_\Gamma(x, y) = \Pr_{i \in_R [n]}[x[i] \neq y[i]]$, where $x[i], y[i] \in \Gamma$. Typically, Γ will be clear from the context, we will then drop it from the subscript.

2.1 Circuit and Complexity Classes

We assume that the reader is familiar with standard complexity classes such as $\mathcal{NP}$, $\mathcal{EXP}$ and $\mathcal{NEXP}$. For a positive integer $i \geq 0$, $\mathcal{AC}^{\mathrm{i}}$ circuits are Boolean circuits (with AND, OR and NOT gates) of size poly(n), depth $O(\log^i n)$, and unbounded fan-in AND and OR gates (where n is the length of the input). $\mathcal{NC}^{\mathrm{i}}$ circuits are boolean circuits of size poly(n) and depth $O(\log^i n)$ where the fan-in of AND and OR gates is 2. We use the same notations to denote the classes of functions computable by these circuit models. We denote by $\mathcal{AC}$ the class $\bigcup_{i \in \mathbb{N}} AC^i$, and by $\mathcal{NC}$ the class $\bigcup_{i \in \mathbb{N}} NC^i$. $\mathcal{RNC}^{\mathrm{i}}$, $\mathcal{RAC}^{\mathrm{i}}$, $\mathcal{RNC}$ and $\mathcal{RAC}$ are the (one-sided) randomized analogs of the above classes. In particular, $\mathcal{AC}^0$ circuits are boolean circuits (with AND, OR and NOT gates) of constant-depth, polynomial size, and unbounded fan-in AND and OR gates. $\mathcal{NC}^1$ circuits are boolean circuits of fan-in 2, polynomial size and logarithmic (in the input size) depth. $\mathcal{NC}^0$ circuits are similar to $\mathcal{NC}^1$, but have constant-depth. Note that in $\mathcal{NC}^0$ circuits, every output bit depends on a constant number of input bits. $\mathcal{AC}^0$, $\mathcal{NC}^1$ and $\mathcal{NC}^0$ are the classes of languages (or functions) computable (respectively) by $\mathcal{AC}^0/\mathcal{NC}^1/\mathcal{NC}^0$ circuits. $AC^i[q]$ (for a prime q) are similar to $\mathcal{AC}^{\mathrm{i}}$ circuits, but augmented with $\mod q$ gates.

Throughout, circuits may have many output bits (we specify the exact number when it is not clear from the context). Also, often we consider uniform circuit classes. Unless we explicitly note otherwise, circuit families are log-space uniform, i.e. each circuit in the family can be described by a Turing machine that uses a logarithmic amount of space in the size of the circuit (for non-polynomial size circuit families the default uniformity is using space logarithmic in the circuit family size). Thus $\mathcal{NC}^0$ (resp. $\mathcal{NC}^1$) computations in this work are equivalent to constant (resp. logarithmic) parallel time in the CREW PRAM model.

Finally, we extensively use oracle circuits: circuits that have (unit cost) access to an oracle computing some function. We sometimes interpret this function as a string, in which case the circuit queries and index and receives from the oracle the symbol in that location in the string.

2.2 Interactive Proofs

We give here standard definitions for interactive proof systems.

Definition 2.1. An *interactive proof system* for a language L with completeness c and soundness s, is a two party game between a probabilistic polynomial-time verifier V and a computationally unbounded prover P. The system has two stages: First, in the *interaction* stage, V and P are given a common input x and they exchange messages to produce a transcript $t = (V(r), P)(x)$ (the entire messages exchange) where r are the internal random coins of V. Then, in the *decision* stage, V decides according to x, t and r, whether to accept or reject. The following should hold:

1. (Completeness) There exist a (honest) prover strategy P, such that for every $x \in L$, $\Pr_r[V(x,t,r) = \text{accept}] \geq c$, where $t = (V(r), P)(x)$.
2. (Soundness) For every $x \notin L$ and every prover P^*, $\Pr_r[V(x,t,r) = \text{accept}] \leq s$, where $t = (V(r), P^*)(x)$.

If we do not specify c and s then their respective default values are $2/3$ and $1/3$.

A *multi-prover interactive proof system* with p provers for a language L is the same as an interactive proof system with the only difference that at the interaction stage V exchange messages with p different provers that have no communication between them. Thus the transcript t contains p separate sub-transcripts $t_1, \ldots, t_p$ each with a different prover.

A round of interaction is an exchange of p messages ($p \geq 1$) that are sent in parallel from the verifier to the p provers (one to each prover) and p messages that are sent in parallel back from the provers to the verifier (one from each prover). Note that the number of rounds may depend on the length of the input to the protocol. We denote by $\mathcal{AM}$ (i.e., Arthur-Merlin games) the class of languages that have protocols with one prover and a constant number of interaction rounds.

We denote by $IP_{c,s}(k)$ the class of languages that have an interactive proof system with completeness c, soundness s and k rounds of interaction. It is well known that $IP_{2/3,1/3}(k) = IP_{1-2^{-n},2^{-n}}(k)$ (see, e.g., [Gol99]). We denote by $MIP_{c,s}(p,k)$ the class of languages that have a multi-prover interactive proof system with completeness c, soundness s, p provers and k rounds of interaction.

We use the following characterization of $\mathcal{NEXP}$ by Feige and Lovasz:

Theorem 2.2. *[FL92]* $\mathcal{NEXP} = MIP_{1,2^{-n}}(2,1)$

2.3 Interactive Proofs: New Definitions

We proceed with new definitions and terminology that that will be useful for us later. We first formally define the notion of $\mathcal{NC}^0$ (or constant parallel time) verifiers.

Definition 2.3. We say that round i of a proof system (with one or more provers) can be computed in $\mathcal{NC}^0$ if the computation of the verifier in this round can be performed by an $\mathcal{NC}^0$ circuit (that may depend on the round i) that is given the input x to the protocol, the randomness r and the partial transcript from the previous $i-1$ rounds.

We assume that the circuit may depend on the number of the round because $\mathcal{NC}^0$ circuits cannot even increment an integer by 1.[4]

Definition 2.4. We say that a language can be verified (interactively) in constant parallel time, if it has an interactive proof system (with one or more provers) with a constant number of rounds, and the entire verifier's strategy can be implemented in $\mathcal{NC}^0$, i.e. every round of the interaction stage as well as the decision stage.

We define a special type of proof systems in which "most" of the verifier's computation is pushed to the decision stage, keeping the computation during the interaction stage extremely efficient.

[4]We could consider a model with the same $\mathcal{NC}^0$ verifier in all rounds. The models are equivalent for protocols with $O(1)$ communication rounds. For other protocols results carry through, except that we can only bound the *expected* number of communication rounds when interacting with a malicious prover.

Definition 2.5. We say that a proof system (with one or more provers) is *simple* if in every round of the interaction stage the verifier's computation can be performed by an $\mathcal{NC}^0$ circuit.

A special case of simple proof systems is public-coin proof systems.

Definition 2.6. An interactive proof system is called *public-coin*, if at each round of the interaction stage the verifier simply tosses new random coins and sends them to the prover (while preforming no other computation).

Theorem 2.7. *([GS86]) For every $k > 0$, every language in $IP(k)$ has a public-coin interactive proof system with $k + 2$ rounds.*

2.4 Randomized images

We start by defining the properties of functions and languages that we need for our approach. The first property says that we can easily generate a random instance together with the evaluation of the function on this input.

Definition 2.8 (Solved instance generator)**.** Let $f : \{0, 1\}^* \to \{0, 1\}^*$ be a function. We say that a randomized algorithm G is a *solved instance generator* for f if, given 1^n, it generates a pair (x, y), where x is a uniformly random element of $\{0, 1\}^n$ and $y = f(x)$.

The second property is a reduction from one function to another that says, roughly, that we can evaluate the first function on every instance by querying the second function in a random location.

Definition 2.9 (Random instance reduction)**.** Let $f : \{0, 1\}^* \to \{0, 1\}^*$ and $g : \{0, 1\}^* \to \{0, 1\}^*$ be two functions. We say that a pair of algorithms $(\mathcal{R}, \mathcal{E})$ is a *random instance reduction* from f to g if $\mathcal{R}$ is a randomized algorithm that given $x \in \{0, 1\}^n$, generates a pair (x', τ), where x' is a uniformly random element of $\{0, 1\}^{m(n)}$ and $\tau \in \{0, 1\}^*$ and it holds that $\mathcal{E}(g(x'), \tau) = f(x)$.

If $m(n) = n$ we say that the random instance reduction is *length-preserving.* If f and g are the same function, we say that it is a random instance self-reduction.[5] We call $\mathcal{R}$ the *Randomizer* and $\mathcal{E}$ the *Evaluator.*

The objects that we will be interested in are pairs of functions that have the above two properties.

Definition 2.10 (Randomized image)**.** Let $f : \{0, 1\}^* \to \{0, 1\}^*$ and $g : \{0, 1\}^* \to \{0, 1\}^*$ be two functions. We say that g is a *randomized image* of f, if there is a random instance reduction from f to g, and g has a solved instance generator.

We say that it is length-preserving if the random instance reduction is length-preserving, and that it is a randomized self-image if $f = g$. Finally, we say that the randomized image can be implemented in some complexity class $\mathcal{C}$, if the algorithms $G, \mathcal{R}$ and $\mathcal{E}$ (from Definitions 2.8 and 2.9) can be implemented in this class.

[5] Random instance self-reductions are a special form of what is called in the literature *random self-reductions.* The word instance in our terminology, should emphasize the fact that the reduction is from one instance to another (random) instance. General random self-reductions can make many *self*-queries to the function in order to compute its value on a given instance.

Next we present several languages that have extremely efficient randomized images. In the sequel we will abuse the term by saying that a language has a randomized image, meaning that its characteristic function has a randomized image. Our constructions use techniques that were developed in the field of cryptography, with some appropriate modifications.

One important family of functions that have randomized images are word problems over finite groups.

Definition 2.11. Let $(G, \odot)$ be a group. We define the word problem of G to be the function $L_G : G^* \to G$, where $L_G(a_1, \ldots, a_n) = a_1 \odot a_2 \odot \cdots \odot a_n$.

Claim 2.12. *Let $(G, \odot)$ be a finite group. Then L_G has a length-preserving randomized self-image that can be implemented by $\mathcal{NC}^0$ circuits given that they can sample random elements from G.*

Proof. Our constructions are based on a randomization technique from [Bab87, Kil88]. Details follow.

Solved instance generator: We would like to sample a random instance $x \in G^n$ together with $y = L_G(x)$. Given 1^n and a random string $\bar{a} = (a_1, \ldots, a_n) \in_R G^n$, define x to be $(a_1, a_1^{-1} \odot a_2, a_2^{-1} \odot a_3, \ldots, a_{n-1}^{-1} \odot a_n)$, and y to be a_n. Since $a_1, \ldots, a_n$ are independently and uniformly distributed, so are $a_1, a_1^{-1} \odot a_2, a_2^{-1} \odot a_3, \ldots, a_{n-1}^{-1} \odot a_n$. Clearly, $L_G(x) = a_1 \odot a_1^{-1} \odot a_2 \odot a_2^{-1} \odot a_3 \cdots a_{n-1}^{-1} \odot a_n = a_n$. Finally, note that every element in x' is a function of two elements in $\bar{a}$, therefore the procedure can be implemented by an $\mathcal{NC}^0$ circuit over the alphabet G.

Random instance self-reduction: The randomizer $\mathcal{R}$, given $x \in G^n$ and a random string $\bar{a} = (a_1, \ldots, a_n) \in_R G^n$, outputs $x' \in G^n$ which is $(x_1 \odot a_1, a_1^{-1} \odot x_2 \odot a_2, a_2^{-1} \odot x_3 \odot a_3, \ldots, a_{n-1}^{-1} \odot x_n \odot a_n)$, and τ which is a_n^{-1}. Define $\mathcal{E}(\sigma, \tau) = \sigma \odot \tau$. Since $a_1, \ldots, a_n$ are independently and uniformly distributed, so are $x_1 \odot a_1, a_1^{-1} \odot x_2 \odot a_2, a_2^{-1} \odot x_3 \odot a_3, \ldots, a_{n-1}^{-1} \odot x_n \odot a_n$. Clearly, $\mathcal{E}(f(x'), a_n^{-1}) = x_1 \odot a_1 \odot a_1^{-1} \odot x_2 \odot a_2 \odot a_2^{-1} \odot x_3 \odot a_3 \cdots a_{n-1}^{-1} \odot x_n \odot a_n \odot a_n^{-1} = x_1 \odot x_2 \odot \cdots \odot x_n = L_G(x)$. Finally, note that every element in y is a function of one element in x and two elements in $\bar{a}$, therefore $\mathcal{R}$ can be implemented by an $\mathcal{NC}^0$ circuit over the alphabet G ($\mathcal{E}$ is over a finite domain so it is clearly in $\mathcal{NC}^0$). ■

Corollary 2.13. *The parity function: $Parity(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i$ where $x_i \in \{0, 1\}$ and the sum is over $GF(2)$ has a length-preserving randomized self-image that can be implemented by $\mathcal{NC}^0$ circuits.*

Barrington [Bar89] has shown that L_{S_5} is complete for the class $\mathcal{NC}^1$ under $\mathcal{NC}^0$ reductions[6] (S_5 is the symmetric group over five elements). We conclude:

Corollary 2.14. *There is an $\mathcal{NC}^1$-complete function under $\mathcal{NC}^0$ reductions that has a length-preserving randomized self-image that can be implemented by $\mathcal{NC}^0$ circuits* that are given access to a source of random elements in S_5.

Having access to a source of random elements in S_5 is a non-standard requirement. The reason that we need it is that standard (Boolean) $\mathcal{NC}^0$ circuits cannot sample uniformly from a set of size $|S_5| = 120$ (given access to a source of random bits). This motivates (as in [AIK06]) the following construction, based on ideas from [IK02], which gives complete languages in the (higher) class $\oplus$-$\mathcal{L}$ that have efficient randomized images that can be implemented by *Boolean* $\mathcal{NC}^0$ circuits. Consider the following language:

[6] While [Bar89] only considers $\mathcal{AC}^0$ reductions, it is clear that the reduction is in fact $\mathcal{NC}^0$ as every element of S_5 in the resulting word problem depends on exactly one input bit of the original instance.

Definition 2.15. The language CMD (for Connectivity Matrix Determinant[7]) is defined as follows: an instance of the language is a $n \times n$ matrix A that has 0/1 entries on the main diagonal and above it, -1 on the second diagonal (one below the main), and 0 below this diagonal. A is represented by the list of $\frac{n(n+1)}{2}$ 0/1 entries on and above the main diagonal. Define the characteristic function of L to be $det(A)$ where the determinant is computed over $GF(2)$.

Claim 2.16. *[IK02] CMD is $\oplus$-$\mathcal{L}$-complete under $\mathcal{NC}^0$ reductions.*

Next, building on ideas from [Bab87, Kil88, FKN94, IK02, AIK06], we show that CMD has a a randomized self-image that can be implemented in $\mathcal{AC}^0[\oplus]$. We will then modify the language to obtain a randomized image (but not a self-image) for CMD that can be implemented in $\mathcal{NC}^0$.

Claim 2.17. *CMD has a length-preserving randomized self-image that can be implemented by $\mathcal{AC}^0[\oplus]$ circuits.*

Proof. For every integer n, Consider the family of $n \times n$ matrices $\mathcal{M}_1$ with 1's on the main diagonal, 0's below it and 0/1 above it. Similarly consider the family of $n \times n$ matrices $\mathcal{M}_2$ with 1's on the main diagonal, 0/1 in the last column (except for entry (n, n) which is 1), and 0's in all other entries. Notice that every matrix in $\mathcal{M}_1 \cup \mathcal{M}_2$ has determinant 1. It is shown in [IK02] that for every instance $A \in CMD$ (respectively $A \notin CMD$), the matrix $C_1 \times A \times C_2$ (represented by the entries on and above the main diagonal) is uniformly distributed over the elements in CMD (respectively not in CMD) of the same length, when C_1 and C_2 are uniformly distributed in $\mathcal{M}_1$ and $\mathcal{M}_2$ respectively. Furthermore, CMD halves the space, that is, for every input length, exactly half the instances are in CMD.

Given these properties, we can construct a solved instance generator and a random instance self-reduction for (the characteristic function of) CMD. The solved instance generator on 1^n chooses at random $b \in_R \{0, 1\}$ and constructs the matrix H_b that has -1 on the second diagonal, b in entry $(1, n)$ and 0's elsewhere. It then chooses $C_1 \in_R \mathcal{M}_1$ and $C_2 \in_R \mathcal{M}_2$ and outputs the pair $(C_1 \times H_b \times C_2, b)$. By the above properties $L(x) = det(C_1 \times H_b \times C_2) = b$, and $C_1 \times H_b \times C_2$ is uniformly distributed (over the choice of b, C_1, C_2).

Notice that,

$$(C_1 \times H_b \times C_2)_{ij} = \sum_{\ell=1}^{n} \sum_{k=1}^{n} ((C_1)_{ik} \cdot (H_b)_{k\ell} \cdot (C_2)_{\ell j}) \tag{1}$$

Thus every entry in the output of the solved instance generator is a degree 3 polynomial over $GF(2)$ of the bits representing C_1, C_2 and H_b and it is therefore computable by $\mathcal{AC}^0[\oplus]$ circuits.

For the random instance self-reduction, given an $n \times n$ matrix A as above, the randomizer $\mathcal{R}$ chooses $b \in_R \{0, 1\}$ and constructs the matrix A' which is equal to A everywhere except for entry $(1, n)$ defined to be $A'_{1,n} = A_{1,n} \oplus b$. Next $\mathcal{R}$ chooses $C_1 \in_R \mathcal{M}_1$ and $C_2 \in_R \mathcal{M}_2$ and outputs the pair $(C_1 \times A' \times C_2, b)$. Finally, for $\mathcal{E}$ to retrieve the value of $det(A)$ from $det(C_1 \times A' \times C_2)$ and b, we notice that $det(A) = det(A') \oplus b$ (when the determinants are computed over $GF(2)$). This is because the entry $(1, n)$ appears in exactly one summand in the determinant of A and A'. So $det(C_1 \times A' \times C_2) \oplus b = det(A') \oplus b = det(A)$. As in the case of the solved instance generator, the random instance self-reduction is computable in $\mathcal{AC}^0[\oplus]$. ■

[7]The origin of the name comes from the fact that the determinant of matrices we consider represents information about the number of of paths between two nodes in a directed acyclic graph.

As in [AIK06], the reason the above construction is not in $\mathcal{NC}^0$, is that every entry in the product matrix is a sum (over $GF(2)$) of many elements, each is a product (over $GF(2)$) of three elements. Thus it does not depend on a constant number of input and randomness bits. To overcome this, as in [AIK06], we randomize this sum using the randomizing technique for the word problem over the group Z_2^+ (Corollary 2.12).

Let us start by defining the following variant of CMD.

Definition 2.18. the language $DCMD$ (for Decomposed Connectivity Matrix Determinant) is defined as follows: for every integer $n > 0$, an instance (of length $\frac{n^3(n+1)}{2}$) of $DCMD$ is described by $\frac{n(n+1)}{2}$ n^2-tuples of bits. The n^2-tuples are indexed by $1 \leq j \leq i \leq n$. The (i,j)'th tuple is denoted $(b_{i,j}^{(1)}, \ldots, b_{i,j}^{(n^2)})$. We think of such an instance as a $n \times n$ matrix A with -1 on the second diagonal, 0's below the second diagonal, and the $\frac{n(n+1)}{2}$ entries above the second diagonal are given by

$$A_{i,j} = \bigoplus_{k=1}^{n^2} b_{i,j}^{(k)}$$

We say that such an instance is in $DCMD$ if and only if the determinant over $GF(2)$ of the matrix it represents is 1.

Claim 2.19. *$DCMD$ is a randomized image of CMD that can be implemented by $\mathcal{NC}^0$ circuits.*

Proof. We describe the solved instance generator, the random instance reduction is obtained similarly. The first step of the solved instance generator is as in the proof of Claim 2.17. That is, let $A = C_1 \times H_b \times C_2$ where $b \in_R \{0,1\}$, $C_1 \in_R \mathcal{M}_1$ and $C_2 \in_R \mathcal{M}_2$. Now consider the entries of A as given in Equation 1. For every entry, (i,j), the solved instance generator chooses n^2 random bits, $(b_{i,j}^{(1,1)}, \ldots, b_{i,j}^{(n,n)})$. It then generates the $DCMD$ instance whose (k,ℓ)'th entry (here we change to indexing in $[n] \times [n]$ instead of $[n^2]$) in the (i,j)'th n^2-tuple is

$$b_{k,\ell-1}^{i,j} \oplus (C_1)_{ik} \cdot (H_b)_{k\ell} \cdot (C_2)_{\ell j} \oplus b_{i,j}^{(k,\ell)}$$

Where we define $b_{i,j}^{(k,0)} = b_{i,j}^{(k-1,n)}$, and $b_{i,j}^{(0,n)} = b_{i,j}^{(n,n)} = 0$ (note that we do not use our random choice for $b_{i,j}^{(n,n)}$, it is included here only to simplify the indexing).

by applying the argument of Claim 2.12 w.r.t. the group Z_2^+ on each entry, we see that this instance is a $DCMD$ representation of the matrix A. Furthermore, For every entry on and above the main diagonal of A, the n^2-tuple associated with it is uniformly distributed among all tuples whose parity equals to that entry. Since the value of every entry on and above the main diagonal is uniformly and independently distributed (as argued in the proof of Claim 2.17), we obtain a uniformly distributed instance of $DCMD$ whose determinant is b. Finally, we note that every bit of the $DCMD$ instance is a function of five input/randmoness bits, and thus the solved instance generator can be implemented by an $\mathcal{NC}^0$ circuit. ∎

Now that we have randomized images for several complete problems, we observe that the property is preserved by reductions.

Claim 2.20. *If a language L is hard for some complexity class $\mathcal{C}_1$ under some class $\mathcal{C}_2$ of Karp-reductions and if L has a randomized image that can be implemented in $\mathcal{C}_2$, then every language $L' \in \mathcal{C}_1$ has a randomized image that can be implemented in $\mathcal{C}_2$.*

Proof. The randomized image of L' will be the one of L. To obtain a random instance reduction, given an instance x for the language L', reduce it to an instance y for L. Then apply on y the random instance reduction from L to its randomized image. ■

We conclude this section with the following lemma which says that there is an $\mathcal{NC}^1$-complete language that has all the properties that we need.

Lemma 2.21. *There is an $\mathcal{NC}^1$-complete language under $\mathcal{NC}^0$ reductions that has a randomized image that can be implemented by $\mathcal{NC}^0$ circuits.*

Proof. Barrington [Bar89] showed that the decisional version of L_{S_5} (in which one has to decide whether the resulting permutation in S_5 is the identity or not) is $\mathcal{NC}^1$ complete under $\mathcal{NC}^0$ reductions. Uniform $\mathcal{NC}^1$ is contained in $\oplus L$, so by Claims 2.16, 2.19 and 2.20, $DCMD$ is a randomized image of the decisional version of L_{S_5} that can be implemented by $\mathcal{NC}^0$ circuits. ■

3 Verification in Constant Parallel Time

We start by showing how to transform any simple proof system into one in which the entire strategy of the verifier (interaction and decision) is in $\mathcal{NC}^0$.

Lemma 3.1. *Every language that has a k-round simple interactive proof system with $p \geq 1$ provers, completeness c and soundness s, has an interactive proof system with p provers, $k+2$ rounds, completeness c and soundness $s+\delta$, where $\delta > 0$ is an arbitrarily small constant, and the verifier's entire strategy (both interaction and decision) is in $\mathcal{NC}^0$.*

Proof Intuition. We first notice that any simple proof system can be transformed into a simple proof system (with one more round) in which the decision stage can be implemented in $\mathcal{AC}^0$ (this essentially boils down to evaluating a CNF formula via the Cook-Levin reduction) and hence also in $\mathcal{NC}^1$. We now want to simulate this $\mathcal{NC}^1$ computation by an $\mathcal{NC}^0$ verifier with the help of the prover. Let L be the $\mathcal{NC}^1$-complete language given by Lemma 2.21 and let I be its randomized image. The verifier creates a *uniformly distributed* ℓ-tuple of instances $c_1, \ldots, c_\ell$ where for each i, c_i is either: (with probability 1/2) a solved instance for which the verifier knows $I(c_i)$ (this can be done by using the solved instance generator for I, see Definition 2.8), or (with probability 1/2) a randomized instance such that given $I(c_i)$ the verifier can compute the output of the $\mathcal{NC}^1$ computation of the original verifier (this can be done using the random instance reduction from L to I and the fact that L is $\mathcal{NC}^1$-complete).

The verifier asks the prover to provide the values $I(c_1), \ldots, I(c_\ell)$. It then checks for every solved instance c_i (for which it knows the value $I(c_i)$) whether the prover answered correctly. The verifier also checks that for the randomized instances the answers that the prover gave all evaluate (via the evaluator) to the same answer for the original instance. If both of these checks pass for all the c_i's, it is a good indication that the prover also provided the correct answer for all the randomized instances. Here we use the fact that the prover cannot distinguish between the two types of instances since they are all uniformly distributed. The verifier can then extract the value of the $\mathcal{NC}^1$ computation from any correct answer to a randomized instance, and use it as its output. ■

Full Proof of Lemma 3.1. We first consider the case of a single prover, i.e. $p = 1$. We proceed by showing that any simple proof system can be transformed into a simple proof system (with one more round) in which the decision stage can be implemented in $\mathcal{AC}^0$ (this essentially boils down to evaluating a CNF formula via the Cook-Levin reduction) and hence also in $\mathcal{NC}^1$. Then we add one more round to enable the verification to be performed in $\mathcal{NC}^0$. Details follow.

Let V' be the verifier and P' the honest prover in the original (simple) protocol and let V, P be these entities in the new protocol. By the definition of a simple proof system (Definition 2.5), the computations of V' during the interaction stage are in $\mathcal{NC}^0$. Thus, V and P run the first k rounds as in the original protocol, and we then add two rounds as follows.

Round $k+1$ In the original protocol, given the input x (of length n), the transcript t and the verifier's random coins r, V' can decide in polynomial time whether to accept or reject. Let $|(x,t,r)| = m(n) = poly(n)$. This round is as follows: V sends all its random coins to P and P sends back the tableau of the computation $V'(x,t,r)$.

If V were an $\mathcal{AC}^0$ circuit, it could at this stage verify that the tableau is correct and deduce the output of V' (accept/reject). This is because checking the validity of the tableau amounts to verifying that x, t, r is written in the first row, and that all the local transitions are legal. However V is not an $\mathcal{AC}^0$ circuit. So we proceed to the next round.

Round $k+2$ Consider the language L associated with the above $\mathcal{AC}^0$ computation. That is, an instance contains a tableau T and x, t, r as above, and it belongs to L if the tableau T is consistent with the computation $V'(x, t, r)$, and if this computation accepts. In particular $L \in NC^1$ and therefore, by Lemma 2.21, it has a randomized image I.

Let ℓ be an integer that will be determined later. Given an instance $a = (T, x, t, r)$, V does the following: for each $i \in [\ell]$ (in parallel and with independent random coins), choose uniformly $v_i \in_R \{0, 1\}$. If $v_i = 0$, then V runs the $\mathcal{NC}^0$ solved instance generator for I on input length $m'(n)$, to obtain a pair (c_i, y_i). If $v_i = 1$, then V runs on a the $\mathcal{NC}^0$ random instance reduction from L to I, to obtain a pair (c_i, τ_i). Here $|c_i| = m'(n) = \mathit{poly}(m(n)) = \mathit{poly}(n)$. V then sends to P the message $(c_1, \dots, c_\ell)$, and P sends back answers $(b_1, \dots, b_\ell) \in \{0, 1\}^\ell$.

Decision V accepts if and only if the following holds:

1. For every $i \in [\ell]$ for which $v_i = 0$, $y_i = b_i$.
2. For every $i \in [\ell]$ for which $v_i = 1$, $\mathcal{E}(b_i, \tau_i) = 1$ (recall that $\mathcal{E}$ is the evaluator in the random instance reduction from L to I).

Correctness Clearly, if ℓ is a constant (independent of n) the entire strategy of V can be implemented in $\mathcal{NC}^0$. We proceed to prove completeness and soundness.

Claim 3.2. *The protocol has completeness c.*

Proof. The honest prover P plays the first k rounds like the honest prover P' of the original protocol. It then sends the correct tableau, and the correct values $b_1, \dots, b_\ell$, which are the membership values (0/1) of the instances $c_1, \dots, c_\ell$ in I. By the definition of solved instance generator, this implies that with probability 1, the verifier passes the first test. By the definition of random instance reduction, for every $i \in [\ell]$ for which $v_i = 1$, $\mathcal{E}(b_i, \tau_i) = 1$ if and only if $(T, x, t, r) \in L$. This happens exactly when the original verifier V' accepts the original protocol, and the probability for that is at least c. ■

Claim 3.3. *The protocol has soundness* $s + 2^{-\ell}$.

Proof. Let x be an instance not in the language. Consider the event:

$$A : \mathcal{E}(b_i, \tau_i) = 1 \text{ for every } i \in [\ell] \text{ for which } v_i = 1$$

By the soundness of the original protocol and the definition of random instance reduction, $\Pr[A] \leq s$. If event A does not occur, the only way that a cheating prover, P^*, can convince V to accept is by cheating on c_i for every i for which $v_i = 1$. If the prover cheats on c_j where $v_j = 0$, then by the definition of solved instance generator, V will reject with probability 1. In other words, in order to cheat and not get caught, P^* must cheat on every i for which $v_i = 1$ and give the correct answer on every i for which $v_i = 0$. By the definitions of solved instance generator and random instance reduction, the v_i's are independent of $(c_1, \dots, c_\ell)$. Thus P^* has to guess exactly the value of ℓ independent unbiased coin tosses which he can do with probability at most $2^{-\ell}$. We conclude that the probability that P^* can convince V to accept is bounded by $s + 2^{-\ell}$. ■

Let $\delta > 0$ be an arbitrarily small constant. By setting $\ell = \log(1/\delta)$ we conclude the proof for single-prover systems. For multi-provers, the same arguments apply where the last two rounds $(k+1, k+2)$ are played only with the first prover P_1. That is, in round $k+1$, V sends to P_1 its random coins as well as the transcripts of messages exchanged with all the other provers, then P_1 and V proceed as above. ■

Remark 3.4. *In the proof above, the "hardest" computation that the verifier is performing is an AND of fan-in* $\log(1/\delta)$. *In terms of parallel computing time this amounts to* $\log\log(1/\delta)$. *Generalizing the argument to non-constant* δ *we can obtain proof systems with negligible soundness (e.g.* $n^{-\log n}$*) with a verifier that runs in* $O(\log\log n)$ *parallel time.*

Remark 3.5. *Vadhan [Vad06] has suggested an alternative implementation of round* $k+2$*: the prover wants to convince* V' *that* $V(x,t,r) = 1$. *Let* $b = V(x,t,r)$, *and for* $c \in \{0,1\}$ *denote* $I_c = \{z : I(z) = c\}$. *The verifier generates an instance* y *that is uniformly distributed in* I_b *restricted to the relevant input length. It also generates an instance* y' *that is uniformly distributed in* I_0 *restricted to the same input length. The ability to sample such* y, y' *follows directly from the techniques used to prove the results in Section 2.4. The verifier then chooses at random one of* y *and* y' *and the prover has to say whether it is from* I_1 *or* I_0. *Note that this is very similar to the protocol for Graph Non-Isomorphism [GMW91].*

3.1 General Proof Systems

Next we want to use Lemma 3.1 to obtain our results about general proof systems. For proof systems with a single prover, we can first apply the transformation of Goldwasser and Sipser [GS86] to obtain a public-coin protocol (which is clearly also a simple protocol). Then, by applying Lemma 3.1 to the resulting protocol we obtain a general transformation from any interactive single-prover proof system to a one in which the verifier is in $\mathcal{NC}^0$ with an addition of $O(1)$ rounds. In particular, we obtain the following theorem and corollary:

Theorem 3.6. *Every language in* $\mathcal{IP}(k)$ *has an interactive protocol with* $k+4$ *rounds, and soundness* δ, *where* $\delta > 0$ *is an arbitrarily small constant and the verifier is in* $\mathcal{NC}^0$.

Proof. Let $L \in \mathcal{IP}(k)$. By Theorem 2.7, L has a public-coin proof system with $k+2$ rounds. We can amplify the completeness and soundness of the protocol to be $1 - 2^{-n}$ and 2^n respectively, while still having $k+2$ rounds. By definition, a public-coin proof system is simple, so we can apply Theorem 3.1 to obtain a proof system with an $\mathcal{NC}^0$ verifier and the stated parameters. ■

For the class $\mathcal{AM}$ we get the following corollary:

Corollary 3.7. *A language is in* $\mathcal{AM}$ *if and only if it can be verified in constant parallel time with one prover, two rounds of interaction, and an arbitrarily small constant soundness.*

Next we want to apply similar arguments to obtain $\mathcal{NC}^0$ verifiers for multi-prover proof systems. Unfortunately, the concept of public-coins does not exist in this context, as clearly the queries to the different provers depend on a common random string that the verifier must hide from the provers. Thus we have to take a different approach.

Feige and Lovasz [FL92] showed how to transform any one-round proof system with many provers (and exponentially small soundness) to a one-round proof system with two provers (and exponentially small soundness). We show that a modification of their ideas can transform any one-round, two-prover protocol to a three-round, two-prover *simple* protocol (both having exponentially small soundness).

Lemma 3.8. *Every language in $\mathcal{MIP}_{c,2^{-n}}(2,1)$, has a simple multi-prover proof system with two provers, three rounds of interaction, completeness c and soundness 2^{-n}.*

Proof. We use a modification of the transformation of [FL92]. Let $L \in MIP_{c,2^{-n}}(2,1)$. Let V', P'_1, P'_2 be the players in the original protocol and V, P_1, P_2 be the players in the new protocol. In the new protocol there are three threads of exchanges that take place at the same time. For the convenience of the reader, we will state for each message which thread it belongs to.

In the new protocol, V plays with P_1 the original protocol, where P_1 plays the role of P'_1 and P'_2. Exchanges that are part of the simulation of the original protocol belong to thread I. Clearly, there is no reason that this protocol will be sound since P_1 can correlate the answers of the two provers. We therefore need to somehow force P_1 to run independent strategies and for that we will need P_2. That is, we will run certain correlation tests between P_1 and P_2. Roughly speaking, the idea is that P_1 will be so occupied with passing these tests, that it will be forced to play independent strategies for P'_1 and P'_2. Exchanges that are part of these tests belong to thread II. In addition, we will use P_1 to help V perform its computations in $\mathcal{NC}^0$ during the interaction stage. Exchanges that help achieve this goal belong to thread III. The last thread helps V perform the computations for the other threads in $\mathcal{NC}^0$, thus typically messages in this thread also have a role in either thread I or II.

Assume w.l.o.g. that every message exchanged in the original protocol is of the same length $\ell = \ell(n) \geq n$ (where n is the input length). Let $\mathbb{F}$ be a field of characteristic 2 of size at least $2^{9\ell}$ (This size of the field is as required by [FL92]). We can assume that V and the provers hold the same representation of this field. (One can use a canonic representation as in [HV06] or use P_1 to provide such a representation and at the decision stage prove that it is indeed an irreducible polynomial (of the relevant degree) over $GF(2)$.)

Let $g_j : \{0,1\}^\ell \to \mathbb{F}$ be a function that represents the optimal strategy for prover P'_j (for $j \in \{1,2\}$). Let $\hat{g}_j$ be the following unique multi-linear representation of g_j over $\mathbb{F}^\ell$: for $b \in \{0,1\}$ and a formal variable x let $s(b,x) = x$ if $b = 1$ and $s(b,x) = 1 - x$ if $b = 0$; then define the ℓ-variate multi-linear polynomial over $\mathbb{F}$:

$$\hat{g}_j(u_1, \ldots, u_\ell) = \sum_{b_1,\ldots,b_\ell \in \{0,1\}} g_j(b_1, \ldots, b_\ell)\Pi_{i=1}^{\ell} s(b_i, u_i)$$

We now describe the protocol on input $x \in \{0,1\}^n$:

Round 1:

V: Threads I & III - Toss coins r' for V' and sends them to P_1.

Thread II - choose uniformly and independently $h_1, h_2 \in_R \mathbb{F}^\ell$ and send them to P_1. (The $\mathcal{NC}^0$ verifier can choose random elements in the field since we assume it is of characteristic 2.)

P_1: Threads I & III - send $q_1, q_2 \in \{0,1\}^\ell$ (the queries V' generates on input x and randomness r'), and $a_1, a_2 \in \{0,1\}^\ell$ (the answers that P'_1, P'_2 provides on input x and queries q_1, q_2).

Thread II - send two univariate polynomials over $\mathbb{F}$ (represented by the list of coefficients), f_1, f_2, each of degree ℓ (these are the restrictions of $\hat{g}_1, \hat{g}_2$ to the lines $d_1 = \{q_1 + th_1 : t \in \mathbb{F}\}$ and $d_2 = \{q_2 + th_2 : t \in \mathbb{F}\}$ respectively, where we view q_1, q_2 as points in $\mathbb{F}^\ell$).
Round 2:
V: Threads I & III - choose uniformly $t_1, t_2 \in_R \mathbb{F}$ and send them to P_1 (indices of random points on the lines).
P_1: Threads I & III - send $v_1 = d_1(t_1) = q_1 + t_1 h_1$ and $v_2 = d_2(t_2) = q_2 + t_2 h_2$ (the names in $\mathbb{F}^\ell$ of the random points on the lines).
Round 3:
V: Thread II - send v_1, v_2 to P_2.
P_2: Thread II - send $y_1 = \hat{g}_1(v_1), y_2 = \hat{g}_2(v_2)$.

Decision: V checks that the following holds:

1. q_1, q_2 are the queries that V' asks P_1', P_2' given input x and randomness r' (threads I & III).
2. V' accepts given x, r' and the transcripts q_1, a_1, q_2, a_2 (thread I).
3. f_1, f_2 are univariate polynomials of degree ℓ over $\mathbb{F}$ (threads II & III).
4. $f_1(0) = q_1$ and $f_2(0) = q_2$ (threads II & III).
5. The values for $v_1, v_2 \in \mathbb{F}^\ell$ that P_1 sent are indeed $d_1(t_1) = q_1 + t_1 h_1$ and $d_2(t_2) = q_2 + t_2 h_2$ respectively (threads II & III).
6. $y_1 = f_1(t_1)$ and $y_2 = f_2(t_2)$ (thread II).

Clearly, the proof system is simple. Completeness follows easily from [FL92]. We now explain why it is sound. Note that all the above tests can be performed deterministically and in polynomial time given the view of V (which includes x, its randomness and the transcript). Thus, the tests themselves do not induce errors. The only errors can come from errors made by the simulation of the original protocol (thread I), or by the consistency tests between the restriction of the functions to the lines and the random points on the line (i.e. thread II). Thus the analysis reduces (with some modifications, see below) to the analysis of Feige and Lovasz [FL92], who prove that the new protocol has completeness and soundness as claimed.

There are two points in which our protocol differs from [FL92]. First, P_1 receives right at the beginning all the random coins of V'. To see why this is not a problem, consider the following mental experiment. Look at the 3-provers protocol, in which V' plays with P_1' and P_2' the original protocol, and with P_3' plays the following protocol: V' sends to P_3' its random coins r', and P_3' has to reply with q_1, q_2, which are the queries V' asks P_1', P_2' respectively on randomness r'. V' accepts if the original protocol with P_1', P_2' accepts and P_3' sends the correct queries. Clearly, this protocol has the same completeness and soundness because V' can check the correctness of the answer sent by P_3'. Now the proof of [FL92] actually shows how to transform a three (or in general polynomially many) provers protocol to a two provers protocol, by letting P_1 play all the provers and run the line vs. point test separately on the simulation of each prover. The idea is that this test forces P_1 to play an independent strategy for each prover in the simulation. This means that the new protocol must be sound because the original protocol is sound (against independent provers strategies). Now in our mental experiment P_3' has only one possible strategy that he can play without being caught.

So he is already forced to play this (independent) strategy and we therefore do not need to run the line vs. point test for P_3', resulting in the protocol that we give above.

The second point in which our protocol differs from [FL92] is that P_1 who provides the restriction of $\hat{g}_j$ to the line d_j, knows on which (random) point on the line we are going to query P_2. However this information is revealed to him only after he commits to some low-degree polynomial. This prevents him from correlating the polynomial with the point on the line. Given this, the analysis of [FL92] goes through. ■

By Lemmas 3.1 and 3.8 we obtain,

Theorem 3.9. *Every language in $\mathcal{MIP}(2,1)$ has a $\mathcal{MIP}(2,5)$ protocol with an $\mathcal{NC}^0$ verifier and arbitrarily small constant soundness.*

Combining this with Theorem 2.2 we get the following characterization of $\mathcal{NEXP}$.

Theorem 3.10. *A language is in $\mathcal{NEXP}$ if and only if it can be verified in constant parallel time with two provers, five rounds, perfect completeness and soundness δ, where $\delta > 0$ is an arbitrarily small constant.*

4 Negative Results

In this section we prove that the use of private coins in our protocol is inherent. We also show that constant soundness is the best one can hope for in proof systems that have a constant number of rounds and an $\mathcal{NC}^0$ verifier. These statements hold unless the language is already in $\mathcal{NC}^0$. We start with a more refined definition of $\mathcal{NC}^0$:

Definition 4.1. For $k \in \mathbb{N}$, $\mathcal{NC}^0_k$ is the class of $\mathcal{NC}^0$ circuits in which every output bit depends on at most k input bits. We say that a language belongs to the class $\mathcal{NC}^0_k$ if for every $n \in \mathbb{N}$, there is an $\mathcal{NC}^0_k$ circuit that decides $L^n = L \cap \{0,1\}^n$.

Note that if a language is in $\mathcal{NC}^0_k$ then its characteristic function (at every input length) is influenced by at most k variables.

Theorem 4.2. *Let $L \subseteq \{0,1\}^*$ be an arbitrary language, then L does not have a public-coin interactive protocol with an $\mathcal{NC}^0$ verifier, unless L is in $\mathcal{NC}^0$.*

Proof. Suppose L is not in $\mathcal{NC}^0_k$ for any constant k and yet it has a public-coin protocol with an $\mathcal{NC}^0$ verifier. In particular, this means that the verifier decides whether to accept its input using an $\mathcal{NC}^0$ circuit that runs on its input, randomness and the transcript. The number of input bits that influence the verifier's decision is constant. Let k be the overall number of input bits that influence the verifier's decision bit. Let n be an input length for which L^n does not have an $\mathcal{NC}^0_k$ circuit. Let $x_1, x_2 \in \{0,1\}^n$ be such that the k bits that the verifier reads are the same in x_1 and x_2, yet $x_1 \in L$ and $x_2 \notin L$. By the fact that L^n does not have an $\mathcal{NC}^0_k$ circuit (and hence its characteristic function is influenced by more than k variables), such a pair of instances exist. Consider the dishonest prover P^*, that on input x_2, for any verifier randomness, plays the strategy of the honest prover on input x_1. Because the protocol is public-coin, the verifier's view in both cases is exactly the same, i.e. for any verifier randomness, the prover's messages on inputs x_1 and x_2 are identical, and thus the bits that the verifier uses to make its decision are also identical. By the protocol's completeness on x_1, the soundness of the protocol on x_2 is violated and we get a contradiction. ∎

Next we state our negative result regarding sub-constant soundness.

Theorem 4.3. *Let $L \subseteq \{0,1\}^*$ be an arbitrary language, then L does not have a constant-round interactive protocol with sub-constant soundness and an $\mathcal{NC}^0$ verifier, unless L is in $\mathcal{NC}^0$.*

Proof. (sketch) Suppose L is not in $\mathcal{NC}^0_k$ for any constant k, and yet it has a constant-round interactive protocol with an $\mathcal{NC}^0$ verifier and soundness s. We consider two cases. First suppose that $s = 0$. Then the interactive protocol can be transformed into a public-coin protocol as follows: the prover sends a sequence of random coins on which the original verifier accepts, as well as the transcript of the entire original protocol, which deterministically depends on these bits. The verifier then checks that the original verifier accepts the protocol with these "random" bits. By the fact that $s = 0$, the prover cannot send a false proof. Moreover, this proof system is trivially public-coin, and Theorem 4.2 says that there is no public-coin proof system with $\mathcal{NC}^0$ verifiers for any language that is not already in $\mathcal{NC}^0_k$ for some constant k.

Next, suppose that $s > 0$. Let $x \notin L$ be an instance that together with a fixed dishonest prover strategy P^* exhibits $s > 0$. Fix some randomness $\bar{r}$ for the verifier that causes it to accept, and define a new *static* prover P^{**} that sends the same messages as $P^*(x)$ does when interacting with $V(x, \bar{r})$, regardless of the verifier's actual messages.

Now consider the verifier's output bit: it depends on only a constant number of bits from the input x, from the verifier's randomness and from the messages sent by P^{**}. Note that the decision bit may depend on messages that V itself sends in the protocol, but since V is $\mathcal{NC}^0$ in every round of the protocol, each of the bits sent by V depends in turn only on a constant number of bits from the input, the randomness, and messages sent in earlier rounds. Thus, since there is only a constant number of rounds, the total number of random bits that affect any V-message bit that affects the verifiers decision bit is constant (though we note it is exponential in the number of rounds).

Let d be the number of randomness bits that influence the decision bit, and denote their values in $\bar{r}$ by $r_{i_1}, \ldots, r_{i_d}$. Then for every random string for which the bits at positions $i_1, \ldots, i_d$ are $r_{i_1}, \ldots, r_{i_d}$, it holds that the verifier accepts the protocol on x. We conclude that the soundness on input x with this fixed strategy P^{**} is at least 2^{-d}, which is a constant. ∎

Discussion. At first glance, it may seem that the proof of Theorem 4.2 should also rule out protocols with private coins (at least for constant-round protocols). We want to explain why this is not the case. We believe that this explanation sheds some interesting light on public versus private coins in the context of $\mathcal{NC}^0$ verifiers, and specifically on our protocol (from Lemma 3.1). The idea in the proof of Theorem 4.2 is that we can let the prover choose its strategy regardless of the input. And then we can argue that since the verifier reads only a constant number of bits from the input before he makes his decision, we can change one input with another without the verifier noticing the change. This cannot be done when the verifier has private coins. Now the prover cannot decide on an arbitrary strategy, because it does not know the private coins of the verifier (i.e. different inputs with the same randomness will not give the same view anymore). This means that if we design our protocol properly, we can force the prover's bits to depend on the entire input. In this case, the decision bit also depends on the entire input via the prover's messages. I.e. even though the decision bit depends on a constant number of prover's bits, each one of them may depend on the entire input. We therefore cannot replace the input without the verifier's noticing the change.

To see how this works in practice, consider the protocol we give in the proof of Lemma 3.1. The last message of the prover contains only a constant number of bits. Let $i \in [\ell]$ be such that $v_i = 1$, and consider the prover's bit b_i. This bit depends on the entire input via the instance c_i that was generated by applying the random instance reduction on the instance $a = (T, x, t, r)$. The protocol, by using private coins, forces the prover to give the correct answer on c_i. The dependency of b_i on the input is then revealed to the verifier by computing $\mathcal{E}(b_i, \tau_i)$.

Moving to our result about sub-constant soundness, we want to point out that if we allow a non-constant number of rounds, we can achieve sub-constant soundness. In fact with an addition of $O(\log n)$ rounds we can achieve the soundness of the original protocol (which can be as small as 2^{-n}). This is because we can spread the $\mathcal{AC}^0$ computation at the decision stage of the simple proof system, over $O(\log n)$ rounds of the protocol. That is, consider the $\mathcal{NC}^1$ circuit that computes this $\mathcal{AC}^0$ computation. The $\mathcal{NC}^0$ verifier computes at each round another level of the $\mathcal{NC}^1$ circuit. it sends the prover the results of the computation. The prover sends a dummy message, and the verifier continues the computation by reading from the transcript the results from the previous layer of the circuit.

A key point in our results is that we only add a constant number of rounds to obtain an $\mathcal{NC}^0$ verifier. This is what allows us to obtain constant parallel time proof systems for $\mathcal{AM}$ and $\mathcal{NEXP}$.

5 Acknowledgements

We thank Salil Vadhan for many illuminating discussions, and for his skepticism which resulted in proofs of lower bounds. We also thank anonymous referees for many helpful comments.

References

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in nc^0. *SIAM J. Comput.*, 36(4):845–888, 2006.

[Bab85] Lászlό Babai. Trading group theory for randomness. In *STOC*, pages 421–429, 1985.

[Bab87] László Babai. Random oracles separate pspace from the polynomial-time hierarchy. *Inf. Process. Lett.*, 26(1):51–53, 1987.

[Bar89] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989.

[BFL91] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991.

[BK95] Manuel Blum and Sampath Kannan. Designing programs that check their work. *Journal of the ACM*, 42(1):269–291, 1995.

[BLR93] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, 1993.

[BM88] László Babai and Shlomo Moran. Arthur-merlin games: A randomized proof system, and a hierarchy of complexity classes. *J. Comput. Syst. Sci.*, 36(2):254–276, 1988.

[FKN94] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *STOC*, pages 554–563, 1994.

[FL92] Uriel Feige and László Lovász. Two-prover one-round proof systems, their power and their problems. In *STOC*, pages 733–744, 1992.

[GGH$^+$07] Shafi Goldwasser, Dan Gutfreund, Alexander Healy, Tali Kaufman, and Guy N. Rothblum. Verifying and decoding in constant depth. In *STOC*, pages 440–449, 2007.

[GGH$^+$08] Shafi Goldwasser, Dan Gutfreund, Alexander Healy, Tali Kaufman, and Guy N. Rothblum. A (de)constructive approach to program checking. In *STOC*, pages 143–152, 2008.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity, or all languages in np have zero-knowledge proof systems. *Journal of the ACM*, 38(1):691–729, 1991.

[Gol99] Oded Goldreich. *Modern cryptography, probabilistic proofs and pseudorandomness*, volume 17 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 1999.

[GS86] Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In *STOC*, pages 59–68, 1986.

[HV06] Alexander Healy and Emanuele Viola. Constant-depth circuits for arithmetic in finite fields of characteristic two. In *In Proceedings of STACS 2006*, pages 672 – 683, 2006.

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 2002.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

[Sha92] Adi Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, 1992.

[Vad06] S. Vadhan. Personal communication, 2006.