Efficient Interactive Proofs for Non-Deterministic Bounded Space

Joshua Cook
Department of Computer Science, University of Texas Austin, TX, USA

Ron D. Rothblum
Faculty of Computer Science, Technion, Haifa, Israel

Abstract

The celebrated **IP** = **PSPACE** Theorem gives an efficient interactive proof for any bounded-space algorithm. In this work we study interactive proofs for *non-deterministic* bounded space computations. While Savitch's Theorem shows that nondeterministic bounded-space algorithms can be simulated by deterministic bounded-space algorithms, this simulation has a quadratic overhead. We give interactive protocols for nondeterministic algorithms directly to get faster verifiers.

More specifically, for any non-deterministic space S algorithm, we construct an interactive proof in which the verifier runs in time $\tilde{O}(n+S^2)$. This improves on the best previous bound of $\tilde{O}(n+S^3)$ and matches the result for *deterministic* space bounded algorithms, up to polylog(S) factors.

We further generalize to *alternating* bounded space algorithms. For any language L decided by a time T, space S algorithm that uses d alternations, we construct an interactive proof in which the verifier runs in time $\tilde{O}(n+S\log(T)+Sd)$ and the prover runs in time $2^{O(S)}$. For $d=O(\log(T))$, this matches the best known interactive proofs for deterministic algorithms, up to polylog(S) factors, and improves on the previous best verifier time for nondeterministic algorithms by a factor of $\log(T)$. We also improve the best prior verifier time for *unbounded* alternations by a factor of S.

Using known connections of bounded alternation algorithms to bounded depth circuits, we also obtain faster verifiers for bounded depth circuits with *unbounded* fan-in.

2012 ACM Subject Classification Theory of computation → Interactive proof systems

Keywords and phrases Interactive Proofs, Alternating Algorithms, AC0[2], Doubly Efficient Proofs

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2023.47

Category RANDOM

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2023/097/`

Funding *Joshua Cook*: Supported by the National Science Foundation under grant number 2200956.
Ron D. Rothblum: Funded by the European Union (ERC, FASTPROOF, 101041208). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

Acknowledgements Thanks to Dana Moshkovitz for introducing the authors of this paper, and Anna Gál for finding the references relating low depth circuits to alternating algorithms.

1 Introduction

Interactive proofs, introduced by Goldwasser Micali and Rackoff [22], are proof systems that enable a prover to convince a verifier of the truth of a given statement. The interaction proceeds in rounds where in each round the prover sends a message and the verifier responds. Crucially, in every round the verifier uses randomness that the prover cannot predict. At the end of the interaction the verifier either accepts or rejects the statement. We require that the honest prover convinces the verifier to accept true statements with high probability (and in fact, in most[1] protocols with probability 1) and that no prover, even a computationally unbounded one, can convince the verifier to accept a false statement other than with some small probability.

One of the most celebrated results in complexity theory is that **IP** = **PSPACE** [26, 34]. That is, the set of languages with polynomial space algorithms is exactly the set of languages with interactive protocols whose verifiers run in polynomial time. Interactive proofs have been prolific throughout other areas of complexity theory, including circuit lower bounds [33, 28], pseudorandomness from uniform assumptions [42], and has also been very influential in other proof systems, such as MIPs [5], PCPs [6, 14, 4, 3], and IOPs [7, 32].

The **IP** = **PSPACE** result can be generalized to any deterministic bounded space computation. For a space S deterministic algorithm, the interactive protocols with the fastest verifiers [9, 40] have a time $\tilde{O}(n + S^2)$ verifier and time $2^{O(S)}$ prover, where $\tilde{O}$ hides polylog(S) factors.[2]

In this work we study interactive proofs for more general forms of bounded space computations: *non-deterministic bounded space* and *alternating bounded space*. Recall that a non-deterministic space S algorithm is a space S Turing machine that gets in addition *read-once* access to a witness (which can be as long as 2^S). For example, the complexity class **NL** refers to non-deterministic logarithmic-space algorithms. Alternating algorithms are a generalization of nondeterministic algorithms that can "alternate" quantifiers. The prior best protocols [9] for space S nondeterministic algorithms have verifier time $\Omega(n + S^3)$, which is an S factor slower than the best verifiers for deterministic algorithms. See Table 1 for a more complete comparison with prior works.

1.1 Our Results

Our main result is an improved interactive proof for alternating algorithms. We start by highlighting a special-case of this result for nondeterministic bounded-space algorithms. We construct interactive proofs for space S nondeterministic algorithms whose verifier runs in time $\tilde{O}(n + S^2)$, matching the time bound for deterministic verifiers (up to polylog(S) factors). Broadly, our techniques combine the recent verifier efficient interactive proofs for bounded space by Cook [9], with an efficient interactive proof for $\mathbf{AC}^0_\oplus$ circuits of Goldreich and Rothblum [20], and an improved derandomization through random walks on expander graphs.

The new interactive proof for non-deterministic bounded space is a special case of a more general result that we show for alternating bounded space algorithms. To state the result precisely, we first set up the notation. Let $\mathbf{ATISP}_d[T, S]$ be the set of languages decided by a simultaneous time T, space S and d alternation algorithm. Alternating algorithms have 3 tapes, a read only input tape containing the input, a read once input containing a witness, and a work tape. Only the work tape is limited to have space S. The input tape is read only, but can be read many times. The witness tape can have T symbols on it, but must be read sequentially and each symbol can only be read once. The witness can be thought of as being separated into d segments, each with a different quantifier. The change of quantifier is called an alternation. For example, nondeterministic algorithms have $d = 1$ since they only use existential quantifiers.

[1] By [17], probability 1 can always be achieved, but that reduction has a significant cost to the prover's runtime.

[2] Throughout this work we mostly optimize for verification time and leave the proving time as a secondary consideration. This is in contrast to *doubly efficient interactive proofs* (see [19]) in which we insist on a polynomial-time prover. In this "doubly-efficient" regime, interactive proofs with a polynomial-time prover and almost linear time verifier are known for linear depth, poly-size, uniform circuits [21] and poly-time and bounded-poly space computation [32].

Let $\mathbf{ITIME}[T_V, T_P]$ be the set of languages with an interactive proof whose verifier runs in time T_V and whose prover runs in time T_P. If T_P is omitted, we assume it is the trivial bound of $T_P = O(2^{T_V})$. In this paper, all our protocols are public-coin and have perfect completeness.

Our most general result is an interactive protocol for alternating bounded-space algorithms.

▶ **Theorem 1** (Interactive Proof For Alternating Space). *For any T, S, and d constructible in time $O(S \log(T))$ and space $O(S)$:*

$$\mathbf{ATISP}_d[T, S] \subseteq \mathbf{ITIME}\left[\tilde{O}\left(n + S\log(T) + Sd\right), 2^{O(S)}\right].$$

Further, the verifier runs in space $O(S\log(d+S))$, the protocol is public coin, has $O(S\log(S)(\log(T)+d))$ rounds, $O(S\log(S)(\log(T)+d)\log(d+S))$ bits of communication, and perfect completeness.

For $d = O(\log(T))$, up to small polylog(S) factors, our protocol has the same verifier time and prover time as the best known protocol for deterministic bounded space algorithms [9]: verifier time $\tilde{O}(n + S\log(T))$ and prover time $2^{O(S)}$. As a special case for nondeterministic algorithms, this gives an interactive protocol with verifier time $\tilde{O}(n + S\log(T))$, improving upon the nondeterministic algorithms in [9], whose verifiers required time $\tilde{O}(n + S\log(T)^2)$, by a factor of $\log(T)$. We note $\log(T)$ may be as large as S.

In a limited sense, these results could be seen as tight, as they match, up to polylog(S) factors, the best known results for simulating deterministic algorithms by alternating ones. Chandra, Kozen, and Stockmeyer [8] show that any deterministic algorithm running in time T and space S has an alternating algorithm running in time $S\log(T)$. Specifically, $\mathbf{TISP}[T, S] \subseteq \mathbf{ATISP}_{\log(T)}[O(S\log(T)), O(S)]$. If we improved our verifier time dependence on $S\log(T)$ or Sd, this would improve the time of alternating algorithms simulating deterministic ones.

For $d = T$, Theorem 1 improves over the best known interactive proofs for alternating algorithms, with unbounded alternations, by Fortnow and Lund [16], which have verifier time $\tilde{O}(n + S^2T)$ and verifier space $O(S\log(T))$. Our protocol's verifier is at least a factor S faster (when $ST = \Omega(n)$).

See Table 1 for a comparison of how our protocol compares to prior protocols for nondeterministic algorithms, and Table 2 for a comparison of how our protocol compares to prior protocols for alternating algorithms.

The best verifiers [9, 40] for deterministic algorithms have verifier time $\tilde{O}(S\log(T) + n)$, verifier space $O(S\log(S))$, and provers with time $2^{O(S)}$. The best provers [32] for deterministic algorithms have prover time $T^{1+o(1)}\text{poly}(S)$, but require verifier time $T^{o(1)}\text{poly}(S) + n\text{polylog}(T)$. These protocols are incomparable for T much larger than S, but much smaller than 2^S. For a more comprehensive summary, see the full version [11].

Table 1 Comparison of different protocols for $\mathbf{NTISP}[T, S]$ with polylog(S) factors omitted.

$\mathbf{NTISP}[T,S]$	Verifier Time	Verifier Space	Prover Time
[34]	$(n+S)\log(T)^2$	$(n+S)\log(n+T)$	$2^{\text{poly}(S,n)}$
[16]	$n+S^3\log(T)$	$S\log(S)$	$2^{\text{poly}(S,n)}$
[21]	$n+S^2\log(T)$	$S\log(S)$	$2^{O(S)}$
[9]	$n+S\log(T)^2$	$S\log(T)$	$2^{O(S)}$
This Work	$n+S\log(T)$	$S\log(S)$	$2^{O(S)}$

Table 2 Comparison of different protocols for $\mathbf{ATISP}_d[T,S]$ with polylog(S) factors omitted.

$\mathbf{ATISP}_d[T,S]$	Verifier Time	Verifier Space	Prover Time
[34]	$(n + S(\log(T) + d))S(\log(T) + d)$	$n + S\log(T) + Sd$	$2^{\text{poly}(S,n)}$
[16]	$n + S^2T$	$S\log(T)$	$2^{\text{poly}(S,n)}$
[21]	$n + S^2\log(T) + S^2d$	$S\log(S + d)$	$2^{O(S)}$
[9]	$n + (S\log(T) + Sd)^2$	$S\log(T) + Sd$	$2^{O(S\log(T)+Sd)}$
This Work	$n + S\log(T) + Sd$	$S\log(S + d)$	$2^{O(S)}$

When $S = O(\log(n))$, our prover runs in polynomial-time. This gives us doubly efficient proofs for alternating algorithms with few alternations and logarithmic space. As a special case, we give doubly efficient interactive proofs for **NL** where the number of bits communicated is $\tilde{O}(\log(n)^2)$. This improves on the amount of communication achieved by GKR (specialized for **NL**), which uses $\tilde{\Omega}(\log(n)^3)$ bits of communication.

▶ **Corollary 2** (Doubly Efficient Interactive Proofs for **NL**). **NL** *has interactive protocols whose provers run in polynomial time, verifiers run in quasilinear time, verifiers use* $\tilde{O}(\log(n))$ *space, the protocol uses* $\tilde{O}(\log(n)^2)$ *rounds,* $\tilde{O}(\log(n)^2)$ *bits of communication, is public coin and has perfect completeness.*

More generally, our protocols for nondeterministic algorithms use a factor $\log(T)$ less communication then the previous best protocols by Cook, and match the best prior protocols for deterministic algorithms, up to polylogarithmic factors.

1.1.1 Unbounded Fan in Circuit Results

Let $\mathbf{SIZE}-\mathbf{DEPTH}[2^S,d]$ be the set of space $O(S)$ uniform circuits of size 2^S and depth d with *unbounded* fan in AND and OR gates. Let T-uniform $\mathbf{SIZE}-\mathbf{DEPTH}[2^S,d]$ be the set of time T uniform, space S circuits of size 2^S and depth d with *unbounded* fan in AND and OR gates. Then due to a close relationship between alternating circuits and low depth circuits by Ruzzo and Tompa [37] (see the full version [11]), we have

▶ **Theorem 3** (Uniform Shallow Circuits Have Fast Interactive Proofs). *For any d,T,S constructible in time $O(S\log(T))$ and space $O(S)$, we have*

$$T\text{-uniform } \mathbf{SIZE}-\mathbf{DEPTH}\left[2^S,d\right] \subseteq \mathbf{ITIME}\left[\tilde{O}(n+S\log(T)+Sd), 2^{O(S)}\right]$$
$$\mathbf{SIZE}-\mathbf{DEPTH}\left[2^S,d\right] \subseteq \mathbf{ITIME}\left[\tilde{O}(n+S^2+Sd), 2^{O(S)}\right].$$

Further, the verifier runs in space $O(S\log(d+S))$ and the protocol is public coin and has perfect completeness.

For fan in 2 circuits, this matches the verifier time of GKR [21], while the prover time remains polynomial in the circuit size[3]. For unbounded fan in circuits, or for alternating algorithms, our verifier is a factor of S faster than GKR.

[3] Note however that recent improvements [12, 39, 44, 45] of GKR have a (close to) linear prover, whereas our prover is only polynomial in the circuit size.

1.2 Proof Overview

We start by reviewing our efficient interactive proofs for deterministic algorithms. Then we explain the difficulty of extending this to nondeterministic algorithms, and how to overcome these problems. Finally we show how to extend this technique to alternating algorithms. We assume familiarity with the sumcheck protocol [26]. For a more detailed explanation of our interactive proofs for deterministic algorithms, see [9] or the nearly identical protocol by Thaler [41, Section 4.5.5] (see also [23, 40]).

1.2.1 Deterministic Algorithms

For a deterministic algorithm A, we first reduce the problem to repeated matrix squaring, then give an interactive protocol for that. Suppose A runs in time T on some input x and has unique start state a and accept state b. Let M be the adjacency matrix of A's computation graph on input x. Then A accepts x if and only if $(M^T)_{a,b} = 1$ (where M^T is M raised to the Tth power, *not* M transposed). For notation, we write $M_{a,b}$ as $M(a,b)$. At a high level, the idea is that if we have an interactive protocol that can reduce a claim that $M^{2i}(u,v) = \alpha$ to the claim that $M^i(u',v') = \alpha'$, then by applying this protocol $\log(T)$ times, we can verify the value of $M^T(a,b)$. We give such a reduction, but on the multilinear extensions of M^{2i} and M^i.

Like [9, 40], we reduce to matrix exponentiation and give an interactive protocol for that, instead of reducing to a quantified Boolean formula [34], or to a uniform circuit [21]. This both simplifies the protocol somewhat and makes it more efficient to compute the prover. The idea is to arithmetize these adjacency matrices, and then use a sum check [26] to reduce the statement about M^{2i} to the statement about M^i. In particular, we use the sum check for matrix exponentiation given by Thaler [39], details follow.

For a finite field $\mathbb{F}$, for any i define $\widehat{M^i} : \mathbb{F}^S \times \mathbb{F}^S \to \mathbb{F}$ as the multilinear extension of M^i. That is, $\widehat{M^i}$ is multilinear and for each $u, v \in \{0,1\}^S$ we have $\widehat{M^i}(u,v) = M^i(u,v)$. Then observe that for any i, and $u, v \in \mathbb{F}^S$ we have

$$\widehat{M^{2i}}(u,v) = \sum_{w \in \{0,1\}^S} \widehat{M^i}(u,w)\widehat{M^i}(w,v).$$

To see that this formula is correct, first observe that it is correct for *Boolean* values as it precisely corresponds to the definition of matrix multiplication. So the formula is correct on Boolean values. Since both sides of the equation are multi-linear[4], it follows that the formula holds for all values.

Then, we can use the sum check of [26] to reduce this to a claim that for some $w' \in \mathbb{F}^S$ and some $\beta \in \mathbb{F}$ we have $\beta = \widehat{M^i}(u,w')\widehat{M^i}(w',v)$. Then using a multi-point reduction, as was done in GKR [21], we reduce this to a claim that for some $u', v' \in \mathbb{F}^S$ and $\alpha' \in \mathbb{F}$ we have that $\alpha' = \widehat{M^i}(u',v')$.

Finally running this $\log(T)$ times gives the interactive protocol for deterministic algorithms, since the verifier can efficiently calculate $\widehat{M}$ itself.

[4] To show it is multilinear, we take any variable, say u_i, and show the formula is linear in u_i. For any w see that $\widehat{M^i}(u,w)$ is linear in u_i since $\widehat{M^i}$ is multilinear, and $\widehat{M^i}(w,v)$ is constant in u_i. Thus $\sum_{w \in \{0,1\}^S} \widehat{M^i}(u,w)\widehat{M^i}(w,v)$ is linear in u_i.

We remark that, using linearization type ideas (as in [35]), the above can be extended from the task of deciding whether a deterministic algorithm accepts, to verifying the multilinear extension of a function that the algorithm computes. This will be important for us later on when we use the above interactive proof as a subroutine in the protocol for nondeterministic algorithms.

1.2.2 Nondeterministic Algorithms and Changing Arithmetization

To try to apply this technique to a nondeterministic algorithm, A, we immediately encounter an issue with how to formulate the problem. Namely, if we are doing arithmetic over $\mathbb{Z}$, if the underlying matrix M corresponds to a non-deterministic computation, then the matrix $M^T_{a,b}$ is no longer 1 if and only if A accepts x. Rather, $M^T_{a,b}$ specifies the number of length T paths from a to b. This might be as large as $2^{\Omega(T)}$. If we do arithmetic over a field of characteristic q, then $M^T_{a,b}$ is the number of paths mod q. If the number of paths is some adversarial product of many small primes, we may need $q = \Omega(T)$ for the number of accepting paths to be non zero, mod q. This gave the less efficient verifier time for nondeterministic algorithms in [9].

We will still solve this problem by arithmetization, but we need to change our matrix multiplication from a field matrix multiplication to a binary multiplication, then arithmetize that. We define the matrix multiplication with binary operations where multiplication is AND and addition is OR. So let $M^{(2)} : \{0,1\}^S \times \{0,1\}^S \to \{0,1\}$ denote this binary matrix multiplication, squaring, so that for any $u, v \in \{0,1\}^S$ we have

$$M^{(2i)}(u,v) = \bigvee_{w\in\{0,1\}^S} M^{(i)}(u,w)M^{(i)}(w,v).$$

With this form of matrix exponentiation, it suffices to check if $M^{(T)}_{a,b} = 1$. To do so, we convert binary matrix multiplication into an algebraic circuit. The obvious approach is to use a formula like

$$\tilde{M^{(2i)}}(u,v) = 1 - \prod_{w\in\{0,1\}^S} \Big(1 - \widehat{M^{(i)}}(u,w) \cdot \widehat{M^{(i)}}(w,v)\Big).$$

Unfortunately, this has too high of individual degree: 2^S. One can insert some linearization operations between the multiplications to reduce the degree, like those used by Shen [35]. But then for each of the S variables in w, one would need to add $O(S)$ linearization operations, giving a size $O(S^2)$ algebraic circuit, which we cannot afford.

Instead, we use an idea of Goldreich and Rothblum [20] to probabilistically reduce the degree of these large conjunctions by leveraging the Razborov-Smolensky [31, 36] approximation of large disjunctions as low degree polynomials. Razborov-Smolensky give a reduction from a large disjunction to a random parity check that succeeds with high probability:

$$\forall g \in \{0,1\}^n : \Pr_{r\in\{0,1\}^n}\left[\bigvee_{i\in[n]} g_i = \sum_{i\in[n]} g_i r_i \pmod 2\right] \geq \frac{1}{2}.$$

We note that if $g = 0^n$, then for any r, we have $\sum_{i\in[n]} g_i r_i \pmod 2 = 0$. That is, the error is one sided. The formula $\sum_{i\in[n]} g_i r_i \pmod 2$ is a linear polynomial in a field of characteristic 2. As this is useful for us, we shall only work with fields of characteristic 2 in this paper.

Then, taking an OR of k independent choices of randomness, we get an individual degree k polynomial that succeeds with probability $1 - \frac{1}{2^k}$. If $n = 2^S$ and $k = S$, this gives us a degree $\log(n)$ polynomial for the disjunction that is only wrong with probability $\frac{1}{n}$.

The idea is to replace our boolean formula with a low degree polynomial through Razborov-Smolensky. So let $D_r : \{0,1\}^\ell \times \{0,1\}^S \to \{0,1\}$ be a function outputting our random bits. Here $2^\ell = k = O(S)$ is the number of choices of random bits. Then our new approximation for $M^{(2i)}$ is

$$\widetilde{M^{(2i)}}(u,v) = 1 - \prod_{j \in \{0,1\}^\ell} \left(1 - \sum_{w \in \{0,1\}^S} D_r(j,w) \widehat{M^{(i)}}(u,w) \widehat{M^{(i)}}(w,v) \right). \tag{1}$$

Now we only need to insert $\ell = \log(S)$ levels of linearizations. In the technical details of the paper, we will not actually use algebraic circuits with linearization operations, but will work with these polynomials directly with an "unlinearization" procedure, to avoid discussing circuit uniformity.

1.2.3 Efficient Randomness

At this point we encounter a problem - Equation (1) calls for sampling $2^{\ell+S}$ random bits for D_r, which we cannot afford (since we want our verifier to run in time $\tilde{O}(S)$). So as in GR, we need to sample these using an ϵ biased set. For our ϵ biased set, we use the same one as GR, described in [1] (which is based on a Reed-Solomon code concatenated with a Hadamard code).

Thus, for every value of $j \in \{0,1\}^\ell$, we would like to set $D_r(j,\cdot)$ to be an ϵ-biased set. As $\ell = \log(S)$, if we were to sample these independently, as in GR (i.e., the protocol given in [20]), our verifier would require $O(S^2)$ bits of randomness. Instead, we sample these small bias sets in a correlated manner - via a random walk on an expander (each node in the expander specifies a seed for a small bias set). We use the Margulis [27] expander since it is a constant degree, constant spectral expander with extremely simple edge descriptions: simple additions and subtractions. This makes it very easy to take a start vertex and a (specification of a) random walk and compute any given step on that walk in both small space and small time.

Thus, we only require $R = O(S)$ truly random bits to describe a length $O(S)$ random walk on the ϵ biased sets described by a Reed-Solomon code concatenated with a Hadamard code. So let $D : \{0,1\}^R \times \{0,1\}^\ell \times \{0,1\}^S \to \{0,1\}$ be a function that generates our pseudorandomness, given R bits of true randomness. The verifier first chooses that randomness r, and then $D_r(j,w) = D(r,j,w)$.

Since D is both space and time efficient, we can have the prover compute its value for the verifier, and then have the verifier run the *deterministic* interactive protocol to confirm its value. In contrast, the GR verifier must calculate some low degree extension of D_r directly to use a constant number of rounds. This saves us time over GR.

Finally, as in GR, there is a chance that our pseudorandom bits give a polynomial that fails to compute the disjunctions correctly. In this case, to get perfect completeness we need to prove that the pseudorandom bits are incorrect. To do this, the prover just finds a disjunction closest to the input where the low degree approximation fails and tells the verifier where it fails. This would be a gate where its value in the low degree polynomial is one thing, but one of its input gates should force it to be something else. For instance, an OR gate with a value of 0, and an input to it with a value of 1. Then the verifier can run the interactive protocol to confirm that the low degree polynomial indeed says the gate's value conflicts with its input gate value, showing the pseudorandom bits were bad.

1.2.4 Alternating Algorithms In Terms Of Nondeterministic Algorithms

To use our protocol with alternating algorithms, we want to reduce the alternating algorithm to one with a few large disjunctions or conjunctions over a nondeterministic algorithm. This is similar to what is done when converting alternating algorithms to alternating circuits. Once we have few conjunctions and disjunctions over a nondeterministic algorithm, we can do the same low degree approximations again.

The idea is to, instead of quantifying over the symbols in the read once proof, quantify over the potential states the algorithm could be in when the quantifier changes. Then a nondeterministic algorithm describes if a proof could cause the state to change from one intermediate state to the next when the quantifier changes.

For example, suppose A is an algorithm with $d = 2$ alternations and running in space S recognizing language L. Think of A as a deterministic algorithm taking a proof and outputting true or false. Then since A is an alternating algorithm, $x \in L$ if and only if

$$\forall \text{BigProof1} : \exists \text{BigProof2} : A(x, (\text{BigProof1}, \text{BigProof2})).$$

We can instead be more fine grain with A and talk about its states. Let a be the start state of A and b be its unique accept state. Let B be the algorithm which takes an initial state u a final state v and a proof p, then checks if A starting at u is at state v when given the proof p after time $|p|$. Then our algorithm accepts x if and only if

$$\forall w \in \{0,1\}^S : ((\exists \text{Proof1} : B(x, a, w, \text{Proof1})) \implies (\exists \text{Proof2} : B(x, w, b, \text{Proof2}))).$$

If we know how long Proof1 is supposed to be, we can replace

$$\exists \text{Proof1} : B(x, a, w, \text{Proof1})$$

with a nondeterministic algorithm C. Then our alternating algorithm becomes

$$\forall w \in \{0,1\}^S : C(x, a, w) \implies C(x, w, b).$$

Now, beside our nondeterministic algorithm, we are only quantifying over a variable of size $O(S)$, whereas Proof1 has size $O(T)$.

For a more general example, we can replace

$$\forall \pi_1 : \exists \pi_2 : \forall \pi_3 : \exists \pi_4 : A(x, (\pi_1, \pi_2, \pi_3, \pi_4))$$

with

$$\forall w_1 : C(x, a, w_1) \implies (\exists w_2 : C(x, w_1, w_2) \land (\forall w_3 : C(x, w_2, w_3) \implies C(x, w_3, b))).$$

1.2.5 Protocols for Alternating Algorithms

At this point, each quantification is now only over S variables, so we can use the same trick as before to replace these quantifications with low degree polynomials. Each of the universal quantifiers gets replaced with a large conjunction, and each of the existential quantifiers gets replaced with a large disjunction. Then we use Razborov-Smolensky to replace these conjunctions and disjunctions with low degree polynomials and use an interactive proof to remove the quantifiers one by one.

A few subtleties show up when doing this. One subtlety of this process is that in a straightforward reduction, a d alternation algorithm would give our verifier a claim about C at d different places. Running an interactive protocol d times to confirm each of these d

claims independently would require time $dS\log(T)$, which is too much for us. Instead, we need to use a multi point reduction again to reduce this to a claim about C at one location before running an interactive protocol to confirm that value.

Another subtlety is that it is not convenient to represent C as a nondeterministic algorithm taking two states as an input and checking if there is a computation path from one to the other. It is more convenient to describe C directly with the computation graph of A (now viewing A as a nondeterministic algorithm). For this to work, we need to make sure each alternation takes the same amount of time, say T. Then we write $C(x,u,v) = \widehat{M_x^{(T)}}(u,v)$.

So for example, consider the simple case of a 2 alternation algorithm. That is, suppose we want to verify that

$$\forall w \in \{0,1\}^S : C(x,a,w) \implies C(x,w,b).$$

As described before, we replace C with $\widehat{M^{(T)}}$. So we want to verify

$$\forall w \in \{0,1\}^S : \widehat{M_x^{(T)}}(a,w) \implies \widehat{M_x^{(T)}}(w,b).$$

Now we need to arithmetize the formula being quantified. So let

$$\widetilde{E}(w) = 1 - \widehat{M_x^{(T)}}(a,w)(1 - \widehat{M_x^{(T)}}(w,b)).$$

See that E is low degree and agrees with the predicate $\widehat{M_x^{(T)}}(a,w) \implies \widehat{M_x^{(T)}}(w,b)$ on binary inputs. Of course, $\tilde{E}$ is not multilinear, it has individual degree 2. Luckily, if we let $\widehat{E}$ be the multilinear function consistent with D on binary inputs, then one can use an unlinearization operation (similar to those used by Shen [35]) to reduce from a statement about $\widehat{E}$ to a statement about $\widetilde{E}$. So we need to verify that

$$\forall w \in \{0,1\}^S : \widehat{E}(w).$$

Using our low degree approximation, our verifier first chooses D_r, then wants to check if

$$1 = \prod_{j\in\{0,1\}^\ell} \left(1 - \sum_{w\in\{0,1\}^S} D_r(j,w)(1-\widehat{E}(w))\right). \tag{2}$$

Then we can reduce this to a statement about $\widehat{D_r}$ at a random location, and $\widehat{E}$ at a random location by using $\ell = O(\log(S))$ product reductions. We can unlinearize the statement about $\widehat{E}$ to get a claim about $\widetilde{E}$, or equivalently, about $\widehat{M_x^{(T)}}$ at two locations. Now we can verify the value of $\widehat{M_x^{(T)}}$ by using our protocol for nondeterministic algorithms. But to avoid doing this twice, we first run a multi-point reduction to reduce this to a statement about $\widehat{M_x^{(T)}}$ at one location first.

We can do a similar thing d times for an alternating algorithm. One more subtlety is that for $d > 2$, we need to make $\widehat{E}$ a function of a and b. This is so that we can view the formula in Equation (2) as a function of a and b so we can properly linearize and unlinearize it with respect to a and b. See the full proof for details.

▶ Remark 4 (Proof For Unbounded Fan in Depth Circuits Directly). We could have made an interactive protocol for unbounded fan-in circuits directly. After all, we start with a formula that is essentially the low depth, unbounded fan in circuit for an alternating algorithm, if we view C as a low depth circuit. We can think of our alternating algorithms as a particular kind of very uniform circuit. We don't give an interactive proof for circuits directly to avoid handling uniformity.

One reason we chose not to just provide an interactive protocol for circuits directly is that we need a faster interactive protocol for deterministic algorithms as a subroutine to verify our pseudorandom bits. Since we view this interactive protocol as a problem for bounded space, we find it natural to present the rest of the results in this framework.

An overview of the protocol for alternating algorithms are in Appendix A, but a full proof is given in the full version [11].

1.2.6 Extensions

We note that our paper focuses on verifier time, so we have not optimized other parameters, like verifier space. There are also some other straightforward extensions to further generalizations of alternations we don't prove here.

▶ Remark 5 (Parity Gates and Parity Quantifiers). Like GR, our techniques can also be used on alternating circuits with parity gates, or bounded space algorithms with parity quantifiers. This is clear since a parity gate is an addition gate over fields of characteristic 2, so is of low degree already.

We emphasize that our protocol is different from GR since we need more randomness efficient pseudorandom bits, efficient computation of those pseudorandom bits, more rounds of interaction to keep our degree (and thus verifier time) lower, use of an interactive protocol for deterministic algorithms as a subroutine, and by using connections between low space algorithms, low alternation algorithms, and uniform, low depth circuits.

▶ Remark 6 (Space Efficiency of Our Verifier). While we only achieve a verifier running in space $O(S\log(S))$, for any $\epsilon > 0$ we should be able to get verifier space $O(S/\epsilon)$ using standard techniques, at the cost of increasing verifier time by a factor of S^ϵ. Specifically, instead of using multilinear polynomials, we would use individual degree S^ϵ polynomials. Since we are focused on improving verifier time, we do not prove this result.

This technique was used by Shamir, Fortnow and Lund, and GKR [34, 16, 21] to give the space efficiency claimed in those papers. We state the special case where $\epsilon = \frac{1}{\log(S)}$ in our results since we want to compare verifier time.

2 Preliminaries

We assume the reader is familiar with basic complexity concepts like circuits, Turing machines, and big O notation. See [2] for a reference. For notation, we define $\tilde{O}$ to hide polylogarithmic factors in whatever is inside it in general, not specifically polylog(T) or polylog(n). That is:

▶ **Definition 7** (Big Tilde O). *For functions $f, g : \mathbb{N} \to \mathbb{N}$, we define $f(n) = \tilde{O}(g(n))$ if and only if there exists some constant c such that $f(n) = O(g(n)\log(g(n))^c)$.*

2.1 Bounded Space and Alternating Algorithms

We denote by $\mathbf{TISP}[T, S]$ languages that are computable by a Turing Machine running in time T and space S.

▶ **Definition 8** (TISP). *For functions $T, S : \mathbb{N} \to \mathbb{N}$, we say language L is in $\mathbf{TISP}[T, S]$ if there is an Turing Machine, A, running in time T and space S that decides L.*

We want interactive proofs for a generalization of nondeterministic algorithms called alternating algorithms, as was formally defined in [8]. Like how a nondeterministic algorithms have existential states where the algorithm accepts if any transition from that state accepts, alternating algorithms get both existential and accept states.

We further parameterize our alternating algorithms by the number of alternations. The number of alternations is the number of times it switches between existential and universal states, plus one. For instance, nondeterministic algorithms can be viewed as having one alternation, the second level of the polynomial hierarchy has two, and so on.

▶ **Definition 9 (ATISP).** *For functions $T, S, d : \mathbb{N} \to \mathbb{N}$, we say a language L is in $\mathbf{ATISP}_d[T, S]$ if there is an alternating Turing Machine, A, running in time T and space S that recognizes L such that on any input x, our algorithm A only changes from no quantification to one, from existential to universal quantifiers, or from universal to existential quantifiers d times.*

So for instance, nondeterministic time T and space S algorithms would be contained in $\mathbf{ATISP}_1[T, S]$.

2.2 Interactive Proofs

An interactive proof informally is a proof system where a verifier with access to unpredictable randomness can verify a result such that if the statement is true, an honest prover can convince the verifier with high probability. And if the statement is false, no prover, no matter how powerful, can convince the verifier the statement is true above some constant probability.

For our definition of interactive time, let $\mathrm{Int}(V, P', x)$ denote the random variable that V outputs on input x when interacting with prover P'. See the full version for a more detailed definition [11]. Now we define interactive time. We note that in all our protocols, we achieve perfect completeness. That is, $c = 1$.

▶ **Definition 10** (Interactive Time ($\mathbf{ITIME}$)). *If for any language L, soundness $s \in [0, 1]$, completeness $c \in [0, 1]$, verifier V and prover P we have*
Completeness: *If $x \in L$, then $\Pr[\mathrm{Int}(V, P, x) = 1] \geq c$, and*
Soundness: *if $x \notin L$, then for any function P' we have $\Pr[\mathrm{Int}(V, P', x) = 1] \leq s$,*
then we say V and P are an interactive protocol for L with soundness s and completeness c.

If in addition verifier V runs in time T_V, soundness $s < \frac{1}{3}$, and completeness $c > \frac{2}{3}$, then

$$L \in \mathbf{ITIME}[T_V].$$

If P is also computable by an algorithm running in time T_P, we say

$$L \in \mathbf{ITIME}[T_V, T_P].$$

2.3 Expander Graphs

We will use expander graphs to create hitting samplers through the hitting properties of expanders [24]. See [43] for a more detailed review of expander graphs. We will assume some basic familiarity with graphs here.

We use an expander graph given by Margulis [27] proven by Gabber and Galil [18]. We use this expander because it's simple structure makes it very clear that we can compute random walks on it in very little time and linear space.

▶ **Lemma 11** (Efficient Expander Graphs). *For any square $n = m^2$, there exists an expander graph G with constant degree d and constant spectral expansion $\lambda < 1$.*

Let V be the vertex set of G, and $E : V \times [d] \to V$ be the edge function taking in a vertex, v, and the index of an edge, e, out of v, and outputting the other vertex incident to e. Then E can be computed in space $O(\log(n))$ and time $O(\log(n))$.

2.4 Arithmetization

A core technique of standard interactive proofs is called “arithmetization”. Arithmetization is the process of converting some Boolean function, f, to a low degree formula over a larger field, $\mathbb{F}$, which agrees with f on Boolean inputs. The main function we will be arithmetizing is the state transition of Turing Machines. So for a given algorithm, on a length n input x and two states s_0 and s_1, let the state transition function $M_n(x, s_0, s_1)$ be one if and only if the algorithm on input x starting in state s_0 can transition to state s_1 in one step. See the full version for a more detailed definition [11].

We use [10, Lemma 36] for our arithmetization of the Turing Machine State transition.

▶ **Theorem 12** (Arithmetization of State Transition). *Suppose A is a space S nondeterministic algorithm with transition matrix $M_n : \{0,1\}^n \times \{0,1\}^S \times \{0,1\}^S \to \{0,1\}$ as above.*

Then we can compute the multilinear extension of M_n, denoted $\widehat{M_n} : \mathbb{F}^n \times \mathbb{F}^S \times \mathbb{F}^S \to \mathbb{F}$, in time $(n+S)\tilde{O}(\log(|\mathbb{F}|))$ and space $O((\log(S) + \log(n))\log(|\mathbb{F}|))$.

2.5 Standard Algebraic, Interactive Proof Tools

We use a few standard tools in interactive proofs. Like [21, 9, 40, 26, 34, 35], perhaps the most important tool is the original sum check from [26]. We also use an unlinearization protocol, like the one used by Shen [35]. And a multi-point reduction, like those used in [21, 9]. Similar query reductions have a long history in PCP literature [3, 15, 13, 30, 25]. To see statements of these lemmas, see the full version [11].

3 Interactive Proof For Deterministic Algorithms

Internally, our proof will need interactive proofs for deterministic algorithms. We use a variation of the deterministic protocols from [9, 40]. A full proof can be found in our full version [11], here is just an overview.

The idea of the algorithm is that for a time T algorithm A, if on an input x algorithm A has computation graph G with adjacency matrix M, then for unique start state a and end state b, algorithm A accepts A if and only if $M^T_{a,b} = 1$. Then by using a matrix square reduction repeatedly, this can be reduced to a statement about the value of $\widehat{M}$, the multilinear extension of M, at a random point. And $\widehat{M}$ can be calculated quickly using Theorem 12.

Our matrix square reduction is very similar to the matrix reduction by Thaler [39], except generalized to the case where the matrix is also a multilinear extension of a third input. The main difference with Thaler’s is that we need to perform a few unlinearizations, similar to Shen’s [35].

▶ **Lemma 13** (Matrix Square To Matrix Reduction). *Given a function $M : \{0,1\}^n \times \{0,1\}^S \times \{0,1\}^S \to \mathbb{F}$, denote for any $x \in \{0,1\}^n$ the matrix M_x such that $(M_x)_{u,v} = M(x,u,v)$. Then M_x^2 is defined in the usual way: $(M_x^2)_{u,v} = \sum_{w\in\{0,1\}^S} M_x(u,w)M_x(w,v)$. Now define $M^2 : \{0,1\}^n \times \{0,1\}^S \times \{0,1\}^S \to \mathbb{F}$ by $M_x^2(u,v) = (M_x^2)_{u,v}$. Let $\widehat{M}$ be the multilinear extension of M and $\widehat{M^2}$ be the multilinear extension of M^2.*

Then there is an $S+n+2$ round interactive protocol with $O((S+n)\log(|\mathbb{F}|))$ bits of communication, a verifier V that runs in time $(S+n)\tilde{O}(\log(|\mathbb{F}|))$ and space $O((S+n)\log(|\mathbb{F}|))$, and a prover P that runs in time $2^{S+n}\tilde{O}(\log(|\mathbb{F}|))$ with $O(2^{S+n})$ oracle queries to $\widehat{M}$. The protocol takes as input $\alpha \in \mathbb{F}$, $u, v \in \mathbb{F}^S$, and $x \in \mathbb{F}^n$ and acts such that

Completeness: *If $\alpha = \widehat{M^2}(x, u, v)$, then when V interacts with P, V outputs a $u', v' \in \mathbb{F}^S$, $x' \in \mathbb{F}^n$, and $\alpha' \in \mathbb{F}$ such that $\alpha' = \widehat{M}(x', u', v')$.*

Soundness: *If $\alpha \neq \widehat{M^2}(x, u, v)$, then for any prover P' with probability at most $\frac{4S+3n}{|\mathbb{F}|}$ will V output a $u', v' \in \mathbb{F}^S$, $x' \in \mathbb{F}^n$, and $\alpha' \in \mathbb{F}$ such that $\alpha' = \widehat{M}(x', u', v')$.*

Now applying this square reduction $\log(T)$ times gives our interactive proof for the multilinear extension of a space efficient function. The proof is essentially many calls to Lemma 13, along with a final check by computing $\hat{M}_n$ directly at the final point using Theorem 12.

▶ **Theorem 14** (Interactive Proof For Multilinear Extension of Bounded Space)**.** *For any function $D : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$, for any $x \in \{0,1\}^n$, denote $D_x : \{0,1\}^m \to \{0,1\}$ by $D_x(y) = D(x, y)$. Let $\widehat{D_x}$ be the multilinear extension of D_x. If D is computed by a space S time T deterministic algorithm, then there is a $(m + S + 2)\log(T)$ round interactive protocol with $O((m + S)\log(T)\log(|\mathbb{F}|))$ bits of communication, a verifier V that runs in time $(n + (m + S)\log(T))\tilde{O}(\log(|\mathbb{F}|))$ and space $O((\log(n) + m + S)\log(|\mathbb{F}|))$, and a prover P that runs in time $2^{2m+2S}\log(T)\tilde{O}(\log(|\mathbb{F}|))$ which takes as input an $x \in \{0,1\}^n$, $w \in \mathbb{F}^S$ and $\alpha \in \mathbb{F}$ such that*

Completeness: *If $\widehat{D_x}(w) = \alpha$, then when V interacts with P, V accepts.*

Soundness: *If $\widehat{D_x}(w) \neq \alpha$, then for any prover P' with probability at most $\frac{(4S+3m)\log(T)}{|\mathbb{F}|}$ will V accept.*

4 Interactive Proofs For Nondeterministic Algorithms

Now we describe an interactive proof for nondeterministic algorithms because it is an interesting special case in its own right, it develops the tools needed for the more general alternating algorithm, and gives a good warm up for the general case. Here we give an outline of the proof, a full proof is in the full version [11].

But before we start, we quickly make a detour to explain that the matrix "multiplication" used here for nondeterministic algorithms is different than the one used for deterministic algorithms. For deterministic algorithms, we used standard multiplication and addition in some field. But for nondeterministic algorithms, we do binary matrix multiplication, with multiplication replaced with AND, and addition replaced with OR. To emphasize the difference, we use parentheses around the exponent to indicate we are performing binary matrix multiplication.

▶ **Definition 15** (Binary Matrix Multiplication)**.** *Let $M : \{0,1\}^S \times \{0,1\}^S \to \{0,1\}$ be any function. Then by induction, define $M^{(1)} = M$ and for any i, define*

$$M^{(i+1)}(u, v) = \bigvee_{w} M^{(i)}(u, w)M(w, v).$$

See that if M is an adjacency matrix of a graph, then $M^{(i)}(s, t) = 1$ if and only if there is a path from s to t of length i.

▶ Remark 16. M^i is different from $M^{(i)}$ in that $M^{(i+1)}$ uses an OR function, whereas M^{i+1} uses a plus function. These are equivalent for the adjacency matrix of a deterministic algorithm, but crucially differ for a nondeterministic algorithm.

We emphasize that these binary matrix multiplications algebraically act very similarly to integer matrix multiplication. Specifically, $M^{(T)}$ can still be calculated with $\log(T)$ repeated binary squaring.

Now our goal is to replace the matrix sum check used for deterministic algorithms, with a new efficient reduction for nondeterministic algorithms. It is not clear how to do this directly, so we use a Razborov-Smolensky style low degree approximation, and give a reduction for that instead.

4.1 Extended Product Reduction

The main tool for this new reduction is this extended product reduction. This reduces a statement about the *multilinear extension* of a large product of terms to a statement about the *multilinear extension* of one term. This product reduction could be used to give a square reduction for nondeterministic algorithms directly, but is much more efficient if the number of multiplications is smaller. This is why we use Razborov-Smolensky.

The idea is to just apply many unlinearizations and product reductions, to one variable the product ranges over at a time. See the full version for a proof [11].

▶ **Lemma 17** (Extended Product Reduction). *Suppose $\widehat{f} : \mathbb{F}^\ell \times \mathbb{F}^S \to \mathbb{F}$ is multilinear. Let $g : \{0,1\}^S \to \mathbb{F}$ be defined by $g(v) = \prod_{u \in \{0,1\}^\ell} \widehat{f}(u, v)$ and let $\widehat{g}$ be the multilinear extension of g.*

Then there is an $\ell(S+1)$ round interactive protocol with $O(\ell S \log(|\mathbb{F}|))$ bits of communication, a verifier V that runs in time $\ell S \tilde{O}(\log(|\mathbb{F}|))$ and space $O((\ell + S)\log(|\mathbb{F}|))$, and a prover P that runs in time $2^{\ell+S}\tilde{O}(\log(|\mathbb{F}|))$ which takes as input $w \in \mathbb{F}^S$, and $\alpha \in \mathbb{F}$ such that

Completeness: *If $\widehat{g}(w) = \alpha$, then when V interacts with P, V outputs a $u' \in \mathbb{F}^\ell$, $v' \in \mathbb{F}^S$, and $\alpha' \in \mathbb{F}$ such that $\widehat{f}(u', v') = \alpha'$.*

Soundness: *If $\widehat{g}(w) \neq \alpha$, then for any prover P' with probability at most $\frac{l(3S+1)}{|\mathbb{F}|}$ will V output a $u' \in \mathbb{F}^\ell$, $v' \in \mathbb{F}^S$, and $\alpha' \in \mathbb{F}$ such that $\widehat{f}(u', v') = \alpha'$.*

4.2 Low degree Approximations

To use Razborov-Smolenski efficiently, we need to be able to sample and calculate our ϵ biased sets, D_r, so they work with high probability and can be calculated efficiently.

Construction of efficient ϵ-biased sets is well researched and very efficient constructions are known [29, 38] and is equivalent to constructing good, linear codes. We use the third construction in [1] as our ϵ biased sets. This is the same ϵ biased set used in [20], except that we need to sample them more efficiently.

▶ **Lemma 18** (ϵ-Biased Set). *For any S, there is an $m = O(S)$ and a function $D' : \{0,1\}^m \times \{0,1\}^S \to \{0,1\}$ such that for any $X \subseteq \{0,1\}^S \setminus \emptyset$*

$$\Pr_{r \in \{0,1\}^m}\left[\sum_{x \in X} D'(r, x) = 1 \pmod 2\right] \geq \frac{1}{4}$$

such that D' runs in $\mathrm{poly}(S)$ time and $O(S)$ space.

Now we have a D' which with constant probability correctly converts an OR to a parity. Now we need to sample enough of these so that with constant probability, we convert 2^S ORs into parities. If we take $O(S)$ independent samples of D', then with probability less than 2^{-2S} will any of these ORs fail to be converted into a parity, so by a union bound with probability at most 2^{-S} will any of them fail to be converted into parity. Of course, we can not afford to take $O(S)$ samples of a string of length $O(S)$. So we take correlated samples using random walks on an expander graph.

Then by using a random walk on the Margulis expander, Lemma 11, with our ϵ biased sets as vertices, we can sample a good choice of ϵ biased sets with high probability.

▶ **Lemma 19** (Sampling a Good D for Many ORs). *Suppose for $n = 2^S$ and $m = 2^{S'}$, for each $i \in [m]$ there is an $f_i \in \{0,1\}$ and $u^i \in \{0,1\}^i$ such that $f_i = \bigvee_{j\in[n]} u^i_j$.*

Then for any ϵ, for $R = O(S + S' + \log(1/\epsilon))$, $L = 2^\ell = O(S' + \log(1/\epsilon))$, there is a space $O(S + \log(1/\epsilon))$, time $\mathrm{poly}(S + \log(1/\epsilon))$ algorithm D such that for each $i \in [m]$ define

$$F_i^r = 1 + \prod_{k\in\{0,1\}^\ell} (1 + \sum_{j\in[n]} D_r(k,j)u^i_j) \mod 2.$$

If $\forall i \in [m] : F_i^r = f_i$, then we say D_r is good for each f. Then $\Pr_r[D_r$ is not good for $f] < \epsilon$.

4.3 Interactive Proofs For Nondeterministic Algorithms

Now we give a summary of our proof, full version available at [11]. We start by defining our interactive proof we wish to run, assuming we got a good D_r. This is a square reduction, assuming we can use the Razborov-Smolensky formula to describe $M^{(2)}$. We call the Razborov-Smolensky style polynomial given by our pseudorandomness D as "M relative to D". We say that our D_r is good if $M^{(T)}$ relative to D is $M^{(T)}$. If D_r is good, we are done. Otherwise, D_r is bad, and makes some mistake first. Then we give an interactive proof to show where it is bad.

First, we formally define M relative to D.

▶ **Definition 20** (M Relative to D). *For any $M : \{0,1\}^S \times \{0,1\}^S \to \{0,1\}$, and $D : \{0,1\}^\ell \times \{0,1\}^S \to \mathbb{F}$, we define M relative to D as the functions, for $k = 1$, $M_D^{(1)} = M_D = M$, and for any $k > 1$ the function $M_D^{(2^k)} : \{0,1\}^S \times \{0,1\}^S \to \{0,1\}$ is*

$$M_D^{(2^k)}(u,v) = 1 + \prod_{j\in\{0,1\}^\ell} (1 + \sum_{w\in\{0,1\}^S} D(j,w)M_D^{(2^{k-1})}(u,w)M_D^{(2^{k-1})}(w,v)).$$

Similar to the deterministic case Lemma 13, we have a repeated square reduction for M relative to D. This is based on an extended product reduction, Lemma 17, and a sum check.

▶ **Lemma 21** (Repeated Square Reduction For M relative to D). *For some $M : \{0,1\}^S \times \{0,1\}^S \to \{0,1\}$, let $\widehat{M}$ be the multilinear extension of M and $\widehat{M^{(2)}}$ be the multilinear extension of $M^{(2)}$. For some $D : \{0,1\}^\ell \times \{0,1\}^S \to \mathbb{F}$ and $T = 2^t$ let $\widehat{M_D^{(T)}}$ be the multilinear extension of M relative to D given by Definition 20.*

Then there is an $O(\ell S \log(T))$ round interactive protocol with $O(\ell S \log(T) \log(|\mathbb{F}|))$ bits of communication, a verifier V that runs in time $\ell S \log(T)\tilde{O}(\log(|\mathbb{F}|))$ and space $O((S + \ell)\log(|\mathbb{F}|))$, and a prover P (with access to the truth table of M and D) that runs in time $2^{O(\ell+S)}\tilde{O}(\log(|\mathbb{F}|))$ which takes as input $u, v \in \mathbb{F}^S$, and $\alpha \in \mathbb{F}$ such that

Completeness: *If $\widehat{M_D^{(T)}}(u,v) = \alpha$, then when V interacts with P, V outputs a $u', v', w' \in \mathbb{F}^S$, $j' \in \mathbb{F}^\ell$, and $\alpha', \beta' \in \mathbb{F}$ such that $\widehat{M}(u',v') = \alpha'$ and $D(j',w') = \beta'$.*

Soundness: *If $\widehat{M_D^{(T)}}(u,v) \neq \alpha$, then for any prover P' with at most $\frac{\log(T)(\ell+2)(6S+2)}{|\mathbb{F}|}$ probability will V output a $u', v', w' \in \mathbb{F}^S$, $j' \in \mathbb{F}^\ell$, and $\alpha', \beta' \in \mathbb{F}$ such that $\widehat{M}(u',v') = \alpha'$ and $D(j',w') = \beta'$.*

If D is good, this gives our interactive protocol for nondeterministic algorithms. Unfortunately, D is not always good. But if it is not good, then a prover can show the verifier where it is bad, giving us perfect completeness. See [11] for full details.

We note here that the field size in our final protocol is $|\mathbb{F}| = \text{poly}(S)$ and $l = O(\log(S))$. So the specific polylog(S) hidden by $\tilde{O}$ in our main result is $O(\log(S)^2\text{polylog}(\log(S)))$. This is worse than the polylog(S) overhead for deterministic algorithms given in [9], which was $O(\log(S)\text{polylog}(\log(S)))$. This is because our extended product reduction is slower than a sum check, but only a $\log(S)$ factor slower.

An overview of the protocol for alternating algorithms are in Appendix A, but a full proof is given in the full version [11].

5 Open Problems

While this mostly closes the gap between the best verifier for deterministic and nondeterministic algorithms, many interesting open problems remain, including:

1. Finding a stronger relationship between verifier time (or even alternating time) and bounded space. We know, for $S \geq n$, that

   $$\mathbf{TISP}[T, S] \subseteq \mathbf{ITIME}[\tilde{O}(S\log(T))].$$

   But it is unknown whether, even with the stronger class of alternating algorithms, if

   $$\mathbf{TISP}[T, S] \subseteq \mathbf{ATIME}[o(S\log(T))].$$

2. Finding a stronger relationship between verifier time and alternating time. We know, for $T \geq n$, that

   $$\mathbf{ATISP}[T, S] \subseteq \mathbf{ITIME}[\tilde{O}(ST)].$$

   Can this factor of S in verifier time be removed?
3. Find interactive protocols for $\mathbf{BPTISP}[T, S]$ with simultaneous verifier time $\tilde{O}(n + S\log(T))$, prover time $2^{O(S)}$ *and* perfect completeness.
   Cook [9] gave a protocol with that verifier and prover time, but with imperfect completeness. Perfect completeness can be achieved in a black box way [17], but these black box reductions do not preserve the prover time.
4. Better doubly efficient proofs. In our special case of alternating algorithms, we can not get provers who run in less than exponential time, without giving sub-exponential time deterministic algorithms for nondeterministic problems.
   But even in the deterministic time and space bounded setting, for $S \geq n$, a major open problem is whether

   $$\mathbf{TISP}[T, S] \subseteq \mathbf{ITIME}[\text{poly}(S), \text{poly}(T)].$$

   We do know from [21] that

   $$\mathbf{TISP}[T, S] \subseteq \mathbf{ITIME}[\text{poly}(S), 2^{O(S)}],$$

   and from [32] that

   $$\mathbf{TISP}[T, S] \subseteq \mathbf{ITIME}[T^{o(1)}, \text{poly}(T)],$$

   but it is unknown if both the fast verifier time and prover time can be achieved simultaneously.

5. Similar verifier time for algorithms with more general kinds of quantifiers. For instance, threshold quantifiers.
   Currently the most verifier efficient known interactive protocol for threshold circuits is to use shallow circuits to compute threshold and run GKR. In this paper, we showed one can do better for unbounded fan-in AND and OR gates. Can this also be done for unbounded fan-in threshold gates? This would be interesting because threshold gates seem much more powerful than AND, OR, or parity gates.

References

1 N. Alon, O. Goldreich, J. Hastad, and R. Peralta. Simple construction of almost k-wise independent random variables. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 544–553 vol.2, 1990. doi:10.1109/FSCS.1990.89575.

2 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, USA, 1st edition, 2009.

3 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, May 1998. doi:10.1145/278298.278306.

4 Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of np. *J. ACM*, 45(1):70–122, January 1998. doi:10.1145/273865.273901.

5 L. Babai, L. Fortnow, and C. Lund. Nondeterministic exponential time has two-prover interactive protocols. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 16–25 vol.1, 1990. doi:10.1109/FSCS.1990.89520.

6 László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, STOC '91, pages 21–32. Association for Computing Machinery, 1991. doi:10.1145/103418.103428.

7 Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *Theory of Cryptography Conference*, 2016.

8 Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981. doi:10.1145/322234.322243.

9 Joshua Cook. More Verifier Efficient Interactive Protocols for Bounded Space. In Anuj Dawar and Venkatesan Guruswami, editors, *42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2022)*, volume 250 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:18, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSTTCS.2022.14.

10 Joshua Cook. More verifier efficient interactive protocols for bounded space, 2022. URL: https://eccc.weizmann.ac.il/report/2022/093/.

11 Joshua Cook and Ron Rothblum. Efficient interactive proofs for non-deterministic bounded space, 2023. URL: https://eccc.weizmann.ac.il/report/2023/097/.

12 Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 90–112. Association for Computing Machinery, 2012. doi:10.1145/2090236.2090245.

13 Irit Dinur, Eldar Fischer, Guy Kindler, Ran Raz, and Shmuel Safra. Pcp characterizations of np: Towards a polynomially-small error-probability. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, STOC '99, pages 29–40. Association for Computing Machinery, 1999. doi:10.1145/301250.301265.

14 Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. Approximating clique is almost np-complete (preliminary version). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 2–12. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185341.

15 Uriel Feige and László Lovász. Two-prover one-round proof systems: Their power and their problems (extended abstract). In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '92, pages 733–744. Association for Computing Machinery, 1992. doi:10.1145/129712.129783.

16 Lance Fortnow and Carsten Lund. Interactive proof systems and alternating time-space complexity. *Theor. Comput. Sci.*, 113(1):55–73, 1993. doi:10.1016/0304-3975(93)90210-K.

17 Martin Fürer, Oded Goldreich, Y. Mansour, Michael Sipser, and Stathis Zachos. On completeness and soundness in interactive proof systems. *Adv. Comput. Res.*, 5:429–442, 1989.

18 Ofer Gabber and Zvi Galil. Explicit constructions of linear size superconcentrators. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 364–370, 1979. doi:10.1109/SFCS.1979.16.

19 Oded Goldreich. On doubly-efficient interactive proof systems, 2018. URL: https://www.wisdom.weizmann.ac.il/~oded/de-ip.html.

20 Oded Goldreich and Guy N. Rothblum. *Constant-Round Interactive Proof Systems for AC0[2] and NC1*, pages 326–351. Springer International Publishing, Cham, 2020. doi:10.1007/978-3-030-43662-9_18.

21 Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4), September 2015. doi:10.1145/2699436.

22 Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989. doi:10.1137/0218012.

23 Edward Hirsch, Dieter van Melkebeek, and Alexander Smal. Succinct interactive proofs for quantified boolean formulas, comment 2. Electronic Colloquium on Computational Complexity (ECCC), 2013. URL: https://eccc.weizmann.ac.il/report/2012/077/comment/2/download/.

24 Nabil Kahale. Eigenvalues and expansion of regular graphs. *Journal of the ACM*, 42(5):1091–1106, 1995.

25 Yael Tauman Kalai and Ran Raz. Interactive pcp. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Part II*, ICALP '08, pages 536–547. Springer-Verlag, 2008. doi:10.1007/978-3-540-70583-3_44.

26 C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 2–10 vol.1, 1990. doi:10.1109/FSCS.1990.89518.

27 Grigorii Aleksandrovich Margulis. Explicit construction of concentrators. *Problemy Peredachi Informatsii*, 9:325–332, 1973. URL: https://cir.nii.ac.jp/crid/1572824500389149056.

28 Cody Murray and Ryan Williams. Circuit lower bounds for nondeterministic quasi-polytime: An easy witness lemma for np and nqp. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 890–901. Association for Computing Machinery, 2018. doi:10.1145/3188745.3188910.

29 Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM Journal on Computing*, 22(4):838–856, 1993. doi:10.1137/0222053.

30 Ran Raz. Quantum information and the pcp theorem. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '05, pages 459–468, USA, 2005. IEEE Computer Society. doi:10.1109/SFCS.2005.62.

31 Alexander A. Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical notes of the Academy of Sciences of the USSR*, 41:333–338, 1987.

32 Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 49–62. Association for Computing Machinery, 2016. doi:10.1145/2897518.2897652.

33 Rahul Santhanam. Circuit lower bounds for merlin-arthur classes. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 275–283. Association for Computing Machinery, 2007. doi:10.1145/1250790.1250832.

34 Adi Shamir. Ip = pspace. *J. ACM*, 39(4):869–877, October 1992. doi:10.1145/146585.146609.

35 A. Shen. Ip = space: Simplified proof. *J. ACM*, 39(4):878–880, 1992. doi:10.1145/146585.146613.

36 R. Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 77–82. Association for Computing Machinery, 1987. doi:10.1145/28395.28404.

37 Larry Stockmeyer and Uzi Vishkin. Simulation of parallel random access machines by circuits. *SIAM Journal on Computing*, 13(2):409–422, 1984. doi:10.1137/0213027.

38 Amnon Ta-Shma. Explicit, almost optimal, epsilon-balanced codes. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 238–251. Association for Computing Machinery, 2017. doi:10.1145/3055399.3055408.

39 Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 71–89, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

40 Justin Thaler. The unreasonable power of the sum-check protocol, March 2020. URL: https://zkproof.org/2020/03/16/sum-checkprotocol/.

41 Justin Thaler. Proofs, arguments, and zero-knowledge. *Found. Trends Priv. Secur.*, 4(2-4):117–660, 2022. doi:10.1561/3300000030.

42 L. Trevisan and S. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. In *Proceedings 17th IEEE Annual Conference on Computational Complexity*, pages 129–138, 2002. doi:10.1109/CCC.2002.1004348.

43 Salil P. Vadhan. Pseudorandomness. *Foundations and Trends® in Theoretical Computer Science*, 7(1–3):1–336, 2012. doi:10.1561/0400000010.

44 Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019 – 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 733–764. Springer, 2019. doi:10.1007/978-3-030-26954-8_24.

45 Jiaheng Zhang, Tianyi Liu, Weijie Wang, Yinuo Zhang, Dawn Song, Xiang Xie, and Yupeng Zhang. Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15–19, 2021*, pages 159–177. ACM, 2021. doi:10.1145/3460120.3484767.

A Interactive Proofs For Alternating Algorithms

Now we a sketch of our interactive protocols for alternating algorithms. This still uses the same Razborov-Smolensky degree reduction technique used for nondeterministic algorithms to reduce the degree of large fan in AND and ORs. The main conceptual challenge is rewriting the alternating algorithm in the correct format. So we do this first. For full proofs, see the full paper [11].

A.1 Alternation Reductions For Bounded Space

To prove our interactive protocol with alternating algorithms, we first must convert our algorithm into a simpler, layered algorithm. This is closely related to the reduction from an alternating algorithm to a low depth circuit by Ruzzo and Tompa [37], and a similar reduction was used by Fortnow and Lund [16] in their interactive proof for alternating algorithms.

▶ **Definition 22** (M with d Alternations). *For any $M : \{0,1\}^S \times \{0,1\}^S \to \{0,1\}$ and integer d, define M with time T and d alternations inductively on d as a function $B^d : \{0,1\}^S \times \{0,1\}^S \to \{0,1\}$ by*

$d = 1$: $B^1(u,v) = M(u,v)$.

d is even

$$\begin{aligned} B^d(u,v) =& \forall w \in \{0,1\}^S : M(u,w) \implies B^{d-1}(w,v) \\ =& \neg \bigvee_{w \in \{0,1\}^S} (M(u,w) \wedge \neg B^{d-1}(w,v)) \end{aligned}$$

d is odd

$$\begin{aligned} B^d(u,v) =& \exists w \in \{0,1\}^S : M(u,w) \wedge B^{d-1}(w,v) \\ =& \bigvee_{w \in \{0,1\}^S} M(u,w) \wedge B^{d-1}(w,v). \end{aligned}$$

Our interactive proof will focus on this intermediate representation of an alternating circuit as a matrix M with d quantifiers of S variables between them. Any alternating algorithm can be converted to a problem of a matrix M (which is the computation graph of a nondeterministic algorithm) with alternations. The idea is that the quantifiers guess the states at which the alternating algorithm switches quantifiers. A more detailed relationship is shown in the full version [11].

▶ **Lemma 23** (Layered Alternating Programs). *For any $L \in \mathbf{ATISP}_d[T,S]$, there is a nondeterministic algorithm A running in time $T' = O(T)$ and space $S' = O(S)$ such that on any input x, if M is the adjacency matrix of the computation graph of A on input x, then $x \in L$ if and only if the $M^{(T')}$ with d alternations, B^d as defined in Definition 22, has $B^d(a,b) = 1$ for some unique starting state a and unique accepting state b.*

A.2 Interactive Proof For Layered Alternations

Now the rest of the proof closely follows the proof for nondeterministic algorithms, defining M with alternations relative to D, showing how an alternation reduction for M with alternations relative to D, and a protocol to show that D is bad.

A subtle difference is that our interactive protocols actually reduce a statement about our alternating algorithm, to a statement about $M^{(T)}$, where M is the adjacency matrix of a nondeterministic algorithm. So we then have to apply our interactive proofs for nondeterministic algorithms. That is, we reduce our statement about alternating algorithms to one about nondeterministic ones, which we already developed the tools for.

▶ **Definition 24** (M with d Alternations, Relative to D). *For any $M : \{0,1\}^S \times \{0,1\}^S \to \{0,1\}$, and $D : \{0,1\}^\ell \times \{0,1\}^S \to \{0,1\}$, we define M with d alternations, relative to D, inductively on d as a function $B_D^d : \{0,1\}^S \times \{0,1\}^S \to \{0,1\}$ by*

$d = 1$: $B_D^1(u,v) = M(u,v)$.

d is even $B_D^d(u,v) = \prod_{k \in \{0,1\}^\ell} (1 + \sum_{w \in \{0,1\}^S} D_r(k,w)(M(u,w) + M(u,w) B_D^{d-1}(w,v))) \mod 2$.

d is odd $B_D^d(u,v) = 1 + \prod_{k \in \{0,1\}^\ell} (1 + \sum_{w \in \{0,1\}^S} D_r(k,w)(M(u,w) B_D^{d-1}(w,v))) \mod 2$.

Now there is an interactive protocol for reducing the number of alternations by 1. Similar to our product reduction for nondeterministic algorithms, it uses our extended product reduction, Lemma 17, and a sum check.

▶ **Lemma 25** (IP for M with d Alternations, Relative To D, Single Step)**.** *For any $M : \{0,1\}^S \times \{0,1\}^S \to \{0,1\}$ and integer $d > 1$, $D : \{0,1\}^\ell \times \{0,1\}^S \to \{0,1\}$ let $B_D^d : \{0,1\}^S \times \{0,1\}^S \to \{0,1\}$ be M with d layered alternations relative to D, as defined in Definition 24. Let $\widehat{B_D^d}$ be the multilinear extension of B_D^d.*

Then there is an $O(\ell S)$ round interactive protocol with $O(\ell S \log(|\mathbb{F}|))$ bits of communication, a verifier V that runs in time $\ell S\tilde{O}(\log(|\mathbb{F}|))$, space $O((\ell+S)\log(|\mathbb{F}|))$, and a prover P (given the truth table of M, B_D^d and D) that runs in time $2^{O(\ell+S)}\tilde{O}(\log(|\mathbb{F}|))$ which takes as input $u, v \in \mathbb{F}^S$, and $\alpha \in \mathbb{F}$ such that

Completeness: *If $\widehat{B_D^d}(u,v) = \alpha$, then when V interacts with P, V outputs a $u', v', w' \in \mathbb{F}^S$, $j' \in \mathbb{F}^\ell$, and $\alpha', \beta', \gamma' \in \mathbb{F}$ such that $\widehat{B_D^{d-1}}(w',v') = \alpha'$, $\widehat{D}(j',w') = \beta'$, and $\widehat{M}(u',w') = \gamma'$.*

Soundness: *If $\widehat{B_D^d}(u,v) \neq \alpha$, then for any prover P' with probability at most $\frac{(\ell+1)(6S+1)}{|\mathbb{F}|}$ will V output a $u', v', w' \in \mathbb{F}^S$, $j' \in \mathbb{F}^\ell$, and $\alpha', \beta', \gamma' \in \mathbb{F}$ such that $\widehat{B_D^{d-1}}(w',v') = \alpha'$, $\widehat{D}(j',w') = \beta'$, and $\widehat{M}(u',w') = \gamma'$.*

Applying this many times gives an interactive protocol reducing a statement about our alternating algorithm to one about a nondeterministic one, which we can solve using the ideas in Lemma 21.

▶ **Lemma 26** (IP for M with d Alternations, Relative To D)**.** *For any $M : \{0,1\}^S \times \{0,1\}^S \to \{0,1\}$ and integer d, $D : \{0,1\}^\ell \times \{0,1\}^S \to \{0,1\}$ let $B_D^d : \{0,1\}^S \times \{0,1\}^S \to \{0,1\}$ be M with d layered alternations relative to D, as defined in Definition 24. Let $\widehat{B_D^d}$ be the multilinear extension of B_D^d.*

Then there is an $O(\ell S d)$ round interactive protocol with $O(\ell S d \log(|\mathbb{F}|))$ bits of communication, a verifier V that runs in time $\ell S d\tilde{O}(\log(|\mathbb{F}|))$, space $O((\ell+S)\log(|\mathbb{F}|))$, and a prover P (given the truth table of M, B_D^d and D) that runs in time $d2^{O(\ell+S)}\tilde{O}(\log(|\mathbb{F}|))$ which takes as input $u, v \in \mathbb{F}^S$, and $\alpha \in \mathbb{F}$ such that

Completeness: *If $\widehat{B_D^d}(u,v) = \alpha$, then when V interacts with P, V outputs a $u', v', w' \in \mathbb{F}^S$, $j' \in \mathbb{F}^\ell$, and $\alpha', \beta' \in \mathbb{F}$ such that $\widehat{M}(u',v') = \alpha'$ and $\widehat{D}(j',w') = \beta'$.*

Soundness: *If $\widehat{B_D^d}(u,v) \neq \alpha$, then for any prover P' with probability at most $\frac{d(\ell+2)(6S+2)}{|\mathbb{F}|}$ will V output a $u', v', w' \in \mathbb{F}^S$, $j' \in \mathbb{F}^\ell$, and $\alpha', \beta' \in \mathbb{F}$ such that $\widehat{M}(u',v') = \alpha'$ and $\widehat{D}(j',w') = \beta'$.*

This would be enough if D was always good, but D may be bad, which must be handled to get perfect completeness. First, let us define what it means for D to be good.

▶ **Definition 27** (D is good for M up to d Alternations)**.** *For any $M : \{0,1\}^S \times \{0,1\}^S \to \{0,1\}$, d and $D : \{0,1\}^\ell \times \{0,1\}^S \to \{0,1\}$, we say that D is good for M with up to d alternations if for all $k \in [d]$ with $k > 1$ we have $B_D^k = B^k$.*

But when D is bad, we give a protocol showing where it is bad. A similar protocol exists for non deterministic algorithms. The idea is to find the first quantifier that D is not good for, and tell them both which clause it has the wrong value on, and which input should have given it a different value.

For instance, if D would claim $(\forall y : \phi(x,y)) = 1$, but it isn't, the prover says which y gives this wrong claim, and which x would have $\phi(x,y) = 0$. Our protocol can then show the verifier that indeed D claims that $\phi(x,y) = 1$, but $\phi(x,y) = 0$ since D is correct all the way up to the quantification on y.

More detailed proofs can be found in the full version [11].

▶ **Lemma 28** (Proving D is Bad for M with Alternations)**.** *For some $M : \{0,1\}^S \times \{0,1\}^S \to \{0,1\}$, and integer d, let $\widehat{M}$ be the multilinear extension of M. Let ℓ be an integer and $D : \mathbb{F}^\ell \times \mathbb{F}^S \to \mathbb{F}$ a multilinear function.*

Then there is a round $O(\ell S d)$ interactive protocol with $O(\ell S d \log(|\mathbb{F}|))$ bits of communication, a verifier V that runs in time $\ell S d \tilde{O}(\log(|\mathbb{F}|))$ and space $O((\ell + S)\log(|\mathbb{F}|))$, and a prover P (with access tot he truth table of M) that runs in time $d 2^{O(\ell+S)} \tilde{O}(\log(|\mathbb{F}|))$ such that

Completeness: *If D is not good for M up to d alternations, then when V interacts with P, V outputs $u', v', w' \in \mathbb{F}^S$, $j' \in \mathbb{F}^\ell$, and $\alpha', \beta' \in \mathbb{F}$ such that $\widehat{M}(u', v') = \alpha'$ and $\widehat{D}(j', w') = \beta'$.*

Soundness: *If D is good for M up to d alternations, then when V interacts with P, with probability at most $\frac{d(\ell+2)(6S+2)}{|\mathbb{F}|}$ will V output $u', v', w' \in \mathbb{F}^S$, $j' \in \mathbb{F}^\ell$, and $\alpha', \beta' \in \mathbb{F}$ such that $\widehat{M}(u', v') = \alpha'$ and $\widehat{D}(j', w') = \beta'$.*

Combining all of these gives our main theorem. Again, we see that the polylogarithmic factor overhead is $O(\log(S)^2 \text{polylog}(\log(S)))$. As noted in the nondeterministic section, this is worse than the $O(\log(S)\text{polylog}(\log(S)))$ factor overhead for deterministic algorithms in [9].

For a full proof, see [11].