More Verifier Efficient Interactive Protocols for Bounded Space

Joshua Cook
University of Texas at Austin, TX, USA

Abstract

Let $\mathbf{TISP}[T, S]$, $\mathbf{BPTISP}[T, S]$, $\mathbf{NTISP}[T, S]$ and $\mathbf{CoNTISP}[T, S]$ be the set of languages recognized by deterministic, randomized, nondeterministic, and co-nondeterministic algorithms, respectively, running in time T and space S. Let $\mathbf{ITIME}[T_V, T_P]$ be the set of languages recognized by an interactive protocol where the verifier runs in time T_V and the prover runs in time T_P.

For $S = \Omega(\log(n))$ and T constructible in time $\log(T)S + n$, we prove:

$$\mathbf{TISP}[T, S] \subseteq \mathbf{ITIME}[\tilde{O}(\log(T)S + n), 2^{O(S)}] \tag{1}$$

$$\mathbf{BPTISP}[T, S] \subseteq \mathbf{ITIME}[\tilde{O}(\log(T)S + n), 2^{O(S)}] \tag{2}$$

$$\mathbf{NTISP}[T, S] \subseteq \mathbf{ITIME}[\tilde{O}(\log(T)^2 S + n), 2^{O(S)}] \tag{3}$$

$$\mathbf{CoNTISP}[T, S] \subseteq \mathbf{ITIME}[\tilde{O}(\log(T)^2 S + n), 2^{O(S)}]. \tag{4}$$

The best prior verifier time is from Shamir [21, 11]:

$$\mathbf{TISP}[T, S] \subseteq \mathbf{ITIME}[\tilde{O}(\log(T)(S + n)), 2^{O(\log(T)(S+n))}].$$

Our prover is faster, and our verifier is faster when $S = o(n)$.

The best prior prover time uses ideas from Goldwasser, Kalai, and Rothblum [9]:

$$\mathbf{NTISP}[T, S] \subseteq \mathbf{ITIME}[\tilde{O}(\log(T)S^2 + n), 2^{O(S)}].$$

Our verifier is faster when $\log(T) = o(S)$, and for deterministic algorithms.

To our knowledge, no previous interactive protocol for **TISP** simultaneously has the same verifier time and prover time as ours. In our opinion, our protocol is also simpler than previous protocols.

2012 ACM Subject Classification Theory of computation → Interactive proof systems

Keywords and phrases Interactive Proofs, Verifier Time, Randomized Space, Nondeterministic Space, Fine Grain Complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2022.14

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2022/093/`

Funding Funded by NSF grant number 1705028 and 2200956.

Acknowledgements Thanks to Dana Moshkovitz for suggestions on writing and presentation, and Tayvin Otti for spellchecking.

1 Introduction

One of the most celebrated results of computer science is the proof that **IP** = **PSPACE** [21, 11]. Any language computable in polynomial space can be verified in polynomial time by a verifier with access to randomness and an untrusted, computationally unbounded prover.

Interactive proofs have many applications, for example proving circuit lower bounds for **MA**/1 [19], and for **NQP** [14]. More verifier time efficient **PCP**s [13] improve the results of [19]. Even pseudo random generators [5] use interactive proofs [9].

The previous best verifier time in an interactive protocol for an algorithm running in time T and space S was by Shamir [21], whose verifier runs in time $\tilde{O}(\log(T)S)$ for $S = \Omega(n)$. We improve this result to apply to any $\log(T)S = \Omega(n)$. Our prover is also more efficient. For instance, if $S = \sqrt{n}$ and $T = 2^{\sqrt{n}}$, then we show that

$$\mathbf{TISP}[2^{\sqrt{n}}, \sqrt{n}] \subseteq \mathbf{ITIME}[\tilde{O}(n), 2^{O(\sqrt{n})}],$$

while Shamir only gives

$$\mathbf{TISP}[2^{\sqrt{n}}, \sqrt{n}] \subseteq \mathbf{ITIME}[\tilde{O}(n^{1.5}), 2^{O(n^{1.5})}].$$

That is, our verifier only requires time $\tilde{O}(n)$, but Shamir's requires time $\tilde{O}(n^{1.5})$. Our prover only requires time $2^{O(\sqrt{n})}$, but Shamir's requires time $2^{O(n^{1.5})}$.

The previous best prover time in an interactive protocol for an algorithm running in time T and space S was by Goldwasser, Kalai and Rothblum [9], whose prover runs in a similar time to ours, but whose verifier requires time $\log(T)S^2$. We improve the verifier time by making the quadratic dependence on S linear. If $T = \mathbf{poly}(S)$, our protocol improves the verifier time from $\tilde{O}(S^2)$ to $\tilde{O}(S)$. If $T = 2^{O(S)}$, our protocol improves the verifier time from $\tilde{O}(S^3)$ to $\tilde{O}(S^2)$.

Our results prove a more direct, more efficient, and (in our opinion) simpler protocol than that given in [21] using sum check [11]. We use a reduction to matrix exponentiation instead of quantified Boolean formulas, and a direct arithmetization of the algorithm instead of a Boolean formula. We then apply this protocol to the special cases of deterministic, randomized, and nondeterministic algorithms.

1.1 Results

Let $\mathbf{ITIME}[T_V, T_P]$ be the class of languages computed by an interactive protocol where the verifier runs in time T_V and the prover runs in time T_P. Similarly, let $\mathbf{ITIME}^1[T_V, T_P]$ be the same with perfect completeness. Let $\mathbf{TISP}[T, S]$ be the class of languages computable in simultaneous time T and space S. Our first result is:

▶ **Theorem 1** (Efficient Interactive Protocol For **TISP**). *Let S and T be computable in time $\tilde{O}(\log(T)S + n)$ with $S = \Omega(\log(n))$. Then*

$$\mathbf{TISP}[T, S] \subseteq \mathbf{ITIME}^1[\tilde{O}(\log(T)S + n), 2^{O(S)}].$$

Our protocol has several other desirable properties. The verifier only needs space $\tilde{O}(S)$. This protocol is also public coin, non adaptive, and unambiguous (as described in [16]). This protocol can also verify an $O(\log(T)S + n)$ bit output, not just membership in a language.

We note that $L \in \mathbf{ITIME}^1[T_V, T_P]$ implies that $L \in \mathbf{SPACE}[O(T_V)]$, since a prover can find an optimal prover strategy in a space efficient way. Thus our dependence on S is essentially optimal. It is open whether one can remove the $\log(T)$ factor.

Using Nisan's PRG for bounded space [15], we extend Theorem 1 to get a similar result for randomized bounded space algorithms. Let $\mathbf{BPTISP}[T, S]$ be the class of languages computable in simultaneous randomized time T and space S.

▶ **Theorem 2** (Efficient Interactive Protocol For **BPTISP**). *Let S and T be computable in time $\tilde{O}(\log(T)S + n)$ with $S = \Omega(\log(n))$. Then*

$$\mathbf{BPTISP}[T, S] \subseteq \mathbf{ITIME}[\tilde{O}(\log(T)S + n), 2^{O(S)}].$$

If one tries to use Nisan's PRG with Shamir's protocol, you increase the input size to $\log(T)S$, which gives a time $\tilde{O}(\log(T)^2S + \log(T)n)$ verifier. One can also use Saks and Zhou [18] to reduce $\mathbf{BPSPACE}[S]$ to $\mathbf{SPACE}[S^{1.5}]$, then apply an **IP**, but this also only gives a time S^3 verifier.

The protocol used for Theorem 1 internally counts the number of accepting paths in a nondeterministic algorithm, modulo a prime. By appropriately sampling a prime, we can get an efficient interactive protocol for **NTISP**.

▶ **Theorem 3** (Efficient Interactive Protocol For **NTISP**). *Let S and T be computable in time $\tilde{O}(\log(T)^2S + n)$ with $S = \Omega(\log(n))$. Then*

$$\mathbf{NTISP}[T, S] \cup \mathbf{CoNTISP}[T, S] \subseteq \mathbf{ITIME}^1[\tilde{O}(\log(T)^2S + n), 2^{O(S)}].$$

While $\mathbf{NSPACE}[O(S)] = \mathbf{Co-NSPACE}[O(S)]$ [10, 23], the same result is not known for time bounded computation. So the case for **NTISP** and **CoNTISP** are both interesting and potentially different.

This version of the paper only proves the protocol for deterministic space. To see the rest of the proofs, see the full paper [6].

1.2 Related Work

This work builds on techniques used by Lund, Fortnow, Karloff and Nisan [11] to prove that $\#\mathbf{P} \in \mathbf{IP}$ and extended by Shamir [21] to show that $\mathbf{PSPACE} = \mathbf{IP}$. Shamir used Savitch's theorem [20] to reduce space bounded computation to a quantified Boolean formula, and then gave a sum check similar to [11] to verify it.

Although it was not shown, the bounded space variation of Shamir's protocol (given at the end of [21]) can be implemented time efficiently for the verifier. [1]

▶ **Theorem 4** (Shamir's Protocol). *Let S and T be time $\tilde{O}(\log(T)(S + n))$ computable with $S = \Omega(\log(n))$. Then*

$$\mathbf{TISP}[T, S] \subseteq \mathbf{ITIME}^1[\tilde{O}(\log(T)(S + n)), 2^{O(\log(T)(S+n))}].$$

One can extend Shamir's protocol to nondeterministic algorithms using other ideas in the same paper to get

▶ **Theorem 5** (Shamir's Protocol for Nondeterministic Algorithms). *Let S and T be time $\tilde{O}(\log(T)^2(S + n))$ computable with $S = \Omega(\log(n))$. Then*

$$\mathbf{NTISP}[T, S] \subseteq \mathbf{ITIME}^1[\tilde{O}(\log(T)^2(S + n)), 2^{O(\log(T)(S+n))}].$$

It is not clear from Shamir's work how to get an efficient prover, or how to handle the case where $S = o(n)$.

Shen [22] gave a highly influential variation of Shamir's proof that is frequently taught (for instance [1]). Shen's result gives a less efficient verifier. Meir gave a proof that $\mathbf{IP} = \mathbf{PSPACE}$ [12] which uses a different kind of sum check with a different kind of code. Or Meir's approach also does not improve the verifier time, but introduces useful techniques [17].

[1] Shamir's reduction from general quantified Boolean formulas is not this efficient. The low space or time for the verifier uses the specific form given by the reduction from bounded space to a quantified Boolean formula.

Sum check was also used by Babai, Fortnow and Lund [4] to prove that **MIP** = **NEXP**. This line of work is foundational to many **PCP**s [3, 2].

In a very influential paper, Goldwasser, Kalai and Rothblum gave doubly efficient interactive proofs for depth bounded computation [9]. Doubly efficient proofs are proofs where the prover runs in time polynomial in the algorithm it wishes to prove. GKR is very efficient for uniform, low depth algorithms, like those in **NC**, both for the verifier and the prover.

▶ **Theorem 6** (GKR For Depth)**.** *Let L be a language computed by a family of $O(\log(w))$-space uniform Boolean circuits of width w and depth d where w and d are computable in time $(n+d)\mathbf{polylog}(w)$. Then*

$$L \in \mathbf{ITIME}^1[(n+d)\mathbf{polylog}(w), \mathbf{poly}(wd)].$$

A space S and a time T nondeterministic algorithm, A, can be converted to a width $2^{O(S)}$ and depth $O(\log(T)S)$ circuit, C, using repeated squaring on the adjacency matrix of A's computation graph. Using Theorem 6 with this circuit we get a protocol for bounded space. The circuit is very simple, so we believe[2] the $\mathbf{polylog}(w) = \mathbf{poly}(S)$ term can be made $\tilde{O}(S)$. This gives:

▶ **Theorem 7** (GKR for **NSPACE**)**.** *Let S and T be time $\tilde{O}(\log(T)S^2 + n)$ computable with $S = \Omega(\log(n))$. Then*

$$\mathbf{NTISP}[T,S] \cup \mathbf{CoNTISP}[T,S] \subseteq \mathbf{ITIME}^1[\tilde{O}(\log(T)S^2+n), 2^{O(S)}].$$

The GKR protocol, as well as ours, only have polynomial time provers when $T = 2^{\Omega(S)}$. Reingold, Rothblum and Rothblum [16] gave a doubly efficient protocol for any time T with constantly many rounds of communication, but is only efficient for the verifier when T is polynomial.

▶ **Theorem 8** (RRR Protocol)**.** *For any constant $\delta > 0$, and integers S and T computable in time $T^{O(\delta)}S^2$, and $T = \Omega(n)$ we have*

$$\mathbf{TISP}[T,S] \subseteq \mathbf{ITIME}^1[O(n\mathbf{polylog}(T) + T^{O(\delta)}S^2), T^{1+O(\delta)}\mathbf{poly}(S)].$$

Further the prover only sends $\left(\frac{1}{\delta}\right)^{O(1/\delta)}$ messages to the verifier.

The specific, S^2 power in the verifier time comes from a note by Goldreich [7], confirmed by the authors of [16]. Our result gives a more efficient verifier ($\log(T)S$ vs $T^\delta S^2$), but a less efficient prover ($2^{O(S)}$ vs $\mathbf{poly}(T)$). We note the result in the [16] paper allows sub constant δ, but is complex and can not give a verifier with time $\mathbf{poly}(\log(T)S)$.

There has been work on other notions of verifier efficiency in interactive protocols. Goldwasser, Gutfreund, Healy, Kaufman and Rothblum [8] studied the computation depth required by verifiers. They showed that for any k round interactive proof, there is a $k + O(1)$ round interactive proof where the computation of the verifier during each round is in $\mathbf{NC}^0$. But the total time to evaluate the new verifier (or the total verifier circuit size) is greater than the verifier time of the original protocol.

[2] Achieving this claimed performance with GKR is not trivial, but we believe it can be done.

Table 1 Comparison of different protocol times, with polylogarithmic factors ommited. The first three columns are verifier times for three special cases and the last column is prover time, which is the same for all three cases. RRR does not work for nondeterministic algorithms.

	TISP	**BPTISP**	**NTISP**	Prover
Shamir	$\log(T)(S+n)$	$\log(T)^2 S + \log(T)n$	$\log(T)^2(S+n)$	$2^{O(\log(T)(S+n))}$
GKR	$\log(T)S^2+n$	$\log(T)S^2+n$	$\log(T)S^2+n$	$2^{O(S)}$
RRR	$T^{O(\delta)}S^2+n$	$T^{O(\delta)}S^2+n$	-	$T^{1+O(\delta)}S^{O(1)}$
Ours	$\log(T)S+n$	$\log(T)S+n$	$\log(T)^2S+n$	$2^{O(S)}$

2 Proof Idea

Our protocol uses sum check [11], but unlike Shamir, Shen, and Meir [21, 22, 12], we do not reduce to a quantified Boolean formula first. Instead, we reduce to matrix exponentiation. This gives us **IP** = **PSPACE** from sum check with fewer steps.

Our protocol improves over Shamir's in two important ways. One, it gives better results when $S = o(n)$. This is due to a better arithmetization of RAM algorithms directly, allowing us to leave the input out of the algorithm's state. This more efficient arithmetization is the primary reason we use the RAM model.

The other is a more efficient prover. This comes from a different reduction to matrix exponentiation instead of quantified formulas. Our reductions gives a more efficient prover algorithm. We will now describe our protocol.

For an algorithm A running on input x in time T and space S, we want to know whether $A(x) = 1$, or if $A(x) = 0$. In an interactive proof, there is a computationally bounded verifier, V, who wants to know $A(x)$, and an unbounded, untrusted prover, P, who wants to convince V of a value for $A(x)$, but may lie. Verifier V can ask many questions to prover P and P answers instantly. If V was deterministic, this would just be NP, so V has access to randomness that P cannot predict.

Our proof is based on low degree polynomials over some field $\mathbb{F}$ of size $\mathbf{poly}(S)$. For space, we only prove the case for deterministic algorithms. See [6] for the full version of this paper. We separate our main proof into several ideas:

1. Computation Graph and Matrix Exponentiation.
   Let M be the adjacency matrix of the computation graph of algorithm A on input x. That is, $M_{a,b} = 1$ if and only if when A on input x is in state a, it transitions to state b in one step. See that $(M^T)_{a,b} = 1$ (where M^T is M to the Tth power, not the transpose of M) if and only if $A(x)$ is in state b after T steps when starting in state a.
2. Arithmetization.
   For any field $\mathbb{F}$, a verifier can efficiently compute the multilinear extension of M, which we denote $\widehat{M} : \mathbb{F}^S \times \mathbb{F}^S \to \mathbb{F}$. That is, $\widehat{M}$ is multilinear and for $a, b \in \{0,1\}^S$, we have $\widehat{M}(a,b) = M_{a,b}$. See Definition 17.
   The goal is to give a protocol for reducing a claim that $\alpha = \widehat{M^2}(a,b)$, to a claim that $\alpha' = \widehat{M}(a',b')$. If we can construct such a protocol, applying it $\log(T)$ times reduces our claim that M^T ends in an accept state to a claim about $\widehat{M}$, which the verifier can compute itself.
3. Sum check.
   We can write $\widehat{M^2}$, the multilinear extension of M^2, in terms of $\widehat{M}$ as $\widehat{M^2}(a,b) = \sum_{c\in\{0,1\}^S} \widehat{M}(a,c)\widehat{M}(c,b)$. Using the sum check protocol from [11], we can reduce a claim that $\alpha = \widehat{M^2}(a,b)$ for some $\alpha \in \mathbb{F}$ and $a, b \in \mathbb{F}^S$ to the claim that $\beta = \widehat{M}(a,c)\widehat{M}(c,b)$ for some $\beta \in \mathbb{F}$ and a random $c \in \mathbb{F}^S$.

4. Product reduction.
   There is a trick used in [9] that reduces the claim that $\beta = \widehat{M}(a,c)\widehat{M}(c,b)$ to the claim that $\alpha' = \widehat{M}(a',b')$ for some $\alpha' \in \mathbb{F}$ and $a', b' \in \mathbb{F}^S$.
5. Repeated square rooting.
   Applying the above protocols $\log(T)$ times, we reduce a claim that M^T has a one in the transition from the start state to the end state, to a claim that $\alpha' = \widehat{M}(a',b')$ which the verifier can calculate and check itself.

Now we explain each of these ideas in a little more detail.

2.1 Computation Graphs

For an algorithm A running in space S on input x, its computation graph G has as vertices S bit states and as edges the state transitions for A on input x. That is, there is an edge from state a to b if and only if when A on input x is in state a, after one step, A is in state b. Let M be the adjacency matrix of G. If A runs in T steps starting in state a, and b is the accept state, then A accepts if and only if $(M^T)_{a,b}$ is one.

2.2 Arithmetization and Low Degree Extensions

For this strategy to work, the verifier needs to be able to compute some error correcting code of the computation, to compare against the claim of the prover. This will be a multilinear extension of M.

A key difference between our arithmetization and Shamir's is that our model of computation is the RAM model, and Shamir's is a single tape Turing machine. So inherent to Shamir's protocol, the state must include the input, but ours does not. This is why we get better results for $S = o(n)$. Further, by arithemtizing the transition function directly instead of reducing to a formula first, we are able to get the stronger multilinear extension instead of just a low degree extension.

We say that $\widehat{\phi} : \mathbb{F}^S \times \mathbb{F}^S \to \mathbb{F}$ is the multilinear extension of $\phi : \{0,1\} \times \{0,1\} \to \mathbb{F}$ if $\widehat{\phi}$ has degree at most 1 in each variable and for all $a, b \in \{0,1\}^S$, $\phi(a,b) = \widehat{\phi}(a,b)$. Note multilinear extensions are unique (see Lemma 16). If M is a $2^S \times 2^S$ matrix with $M_{a,b} = \phi(a,b)$, then we define the multilinear extension of M, denoted $\widehat{M} : \mathbb{F}^S \times \mathbb{F}^S \to \mathbb{F}$, by $\widehat{M}(a,b) = \widehat{\phi}(a,b)$. See Definition 17.

In general, it may be hard to compute low degree extensions. For general functions, it requires reading the whole size 2^{2S} truth table! But many classes of functions have low degree extensions that are easy to compute. Specifically, it is easy to compute the low degree extension of the state transition function of RAM algorithms.

▶ **Lemma 9** (Algorithm Arithmetization). *Let A be a nondeterministic RAM algorithm running in space S and time T on length n inputs, and x be an input with $|x| = n$. Define M to be the $2^S \times 2^S$ matrix such that for any two states $a, b \in \{0,1\}^S$, we have $M_{a,b} = 1$ if when A is running on input x is in state a, then b as a valid transition, and $M_{a,b} = 0$ otherwise.*

Then we can compute the multilinear extension of M ($\widehat{M}$ in Definition 17) in time $\tilde{O}(\log(|\mathbb{F}|)(n+S))$.

The idea is to sum over all instructions the multilinear polynomial identifying that instruction being the current instruction, times the multilinear polynomial of that instruction being performed. Since algorithm A is constant, there are only constantly many instructions we could be on. It is easy to compute multilinear extensions of register operations, and easy to compute multilinear extensions of loading or storing from main memory. See Section 4.3 for details on how to perform the arithmetization.

2.3 Sum Check

The key part of our protocol is the sum check from LFKN [11]. Suppose we have a degree d polynomial over S variables: $q : \mathbb{F}^S \to \mathbb{F}$. Then sum check allows an efficient verifier with access to randomness to verify the sum of q on all Boolean inputs with the help of an untrusted prover.

▶ **Lemma 10** (Sum Check Protocol). *Let S be an integer, d be an integer, p be a prime, and $\mathbb{F}$ be a field with characteristic p where $|\mathbb{F}| \geq d+1$. Suppose $q : \mathbb{F}^S \to \mathbb{F}$ be a polynomial with degree at most d in each variable individually. For a multi-linear polynomial, $d = 1$.*

Then there is a interactive protocol with verifier V and prover P such that on input $\alpha \in \mathbb{F}$ behaves in the following way:

Completeness: *If $\alpha = \sum_{a \in \{0,1\}^S} q(a)$, then when V interacts with P, verifier V outputs some $a' \in \mathbb{F}^S$ and $\alpha' \in \mathbb{F}$ such that $\alpha' = q(a')$.*

Soundness: *If $\alpha \neq \sum_{a \in \{0,1\}^S} q(a)$, then for any prover P', when V interacts with P', with probability at* most *$\frac{dS}{|\mathbb{F}|}$ will V output some $a' \in \mathbb{F}^S$ and $\alpha' \in \mathbb{F}$ such that $\alpha' = q(a')$.*

Verifier Time: *V runs in time $\tilde{O}(\log(|\mathbb{F}|))O(Sd)$.*

Prover Time: *P runs in time $\tilde{O}(\log(|\mathbb{F}|))\mathbf{poly}(d)2^S$ using $O(d2^S)$ oracle queries to q.*

The idea of sum check is to choose the elements of a' one at a time, and ask about univariate polynomials that are partial sums: filled in with the chosen parts of a', leaving one variable unfixed, and summed over the rest of the variables. These are error correcting codes, and one is calculated from the next, which can be checked. Due to Schwartz–Zippel, with high probability, if the claimed equality is wrong, each of these low degree polynomials from the proof must be wrong to not be caught by the verifier. See Appendix A for a proof.

Now our protocol wants to reduce a claim that $\alpha = \widehat{M^2}(a,b)$ to a claim that $\alpha' = \widehat{M^2}(a',b')$. When we inspect $\widehat{M^2}(a,b)$, we see that $\widehat{M^2}(a,b) = \sum_{c \in \{0,1\}^S} \widehat{M}(a,c)\widehat{M}(c,a)$. Then considering the degree 2 in each variable polynomial $q(c) = \widehat{M}(a,c)\widehat{M}(c,a)$, sum check reduces the claim that $\alpha = \widehat{M^2}(a,b)$ to the claim that $\beta = \widehat{M}(a,c)\widehat{M}(c,a)$ for some c. This is very nearly what we want.

2.4 Product Reduction

Now that we have a claim that $\beta = \widehat{M}(a,c)\widehat{M}(c,a)$, our verifier needs to check this. One obvious idea is to ask the prover for both $\alpha_1 = \widehat{M}(a,c)$, and $\alpha_2 = \widehat{M}(c,a)$. Then the prover could prove both of these statements separately. But this would double the number of claims the verifier must check everytime we reduce $\widehat{M^2}$ to $\widehat{M}$. Since we have to do this $\log(T)$ times, the verifier would need to check T claims! So this cannot be done.

Instead, the verifier needs to reduce to a claim about $\widehat{M}$ at a single point to avoid this. The idea is to take a line, $\psi : \mathbb{F} \to (\mathbb{F}^S \times \mathbb{F}^S)$, that passes through both (a,c) and (c,b), and ask the prover for $\widehat{M} \circ \psi$. The function $\widehat{M} \circ \psi$ will have degree $2S$, since $\widehat{M}$ is multilinear. [9] uses a similar trick.

Now $\widehat{M} \circ \psi(0) = \widehat{M}(a,c)$ and $\widehat{M} \circ \psi(1) = \widehat{M}(c,a)$. If the $\beta \neq \widehat{M}(a,c)\widehat{M}(c,a)$, then the prover cannot be honest about $\widehat{M} \circ \psi$, or the verifier will reject. Now the verifier chooses a $d \in \mathbb{F}$, and wants to know $\alpha' = \widehat{M}(\psi(d))$. If the prover was honest, then this will give us that at $(a',b') = \psi(d)$, we have $\alpha' = \widehat{M}(a',b')$. But if the prover lied, then by Schwartz–Zippel, with high probability, $\alpha' \neq \widehat{M}(a',b')$. See Lemma 18 for more details.

2.5 Protocol For Deterministic Algorithms

Using sum check and product reduction, there is a protocol that takes a claim that $\alpha = \widehat{M^2}(a,b)$ to the claim that $\alpha' = \widehat{M}(a',b')$. Note that this protocol works with ANY matrix M, not just the M from the adjacency matrix of the computation graph.

In particular, our prover can start by claiming that for start state a, and end state b, that $\widehat{M^T}(a,b) = 1$. The verifier itself can confirm that a is the start state, and b is an accept, or reject state. If A is deterministic, then A accepts x if and only $\widehat{M^T}(a,b) = 1$.

Using the matrix squared to matrix protocol, there is a protocol to reduce the claim that $\widehat{M^T}(a,b) = 1$ to the claim that $\widehat{M^{T/2}}(a_1,b_1) = \alpha_1$. Then to the claim that $\widehat{M^{T/4}}(a_2,b_2) = \alpha_2$. After repeating this $t = \log(T)$ times, we reduce to the claim that $\widehat{M}(a_t,b_t) = \alpha_t$. But this can be directly checked by the verifier.

3 Preliminaries

We use RAM algorithms with registers and a program as our model of computation. This is used for our efficient arithmetization. But the verifier is very simple and can be implemented efficiently in other common models of computation, like multi-tape Turing machines. We may assume that on accepting or rejecting, our machines instantly clear their states to some canonical accept or reject state. We also assume that $S = \Omega(\log(n))$, otherwise our algorithm can't read its entire input.

We think of our algorithms as defining invalid and valid state transitions. For deterministic algorithms, every state has exactly one valid transition. Then a deterministic algorithm accepts if and only if there is some sequence of memory states such that every transition is valid, the first is the start state, and the final is the accept state.

3.1 Complexity Classes

We use $\tilde{O}$ to suppress poly logarithmic factors.

▶ **Definition 11** ($\tilde{O}$). *For $f, g : \mathbb{N} \to \mathbb{N}$, we say $f(n) = \tilde{O}(g(n))$ if and only if for some constant k, $f(n) = O(g(n)\log(g(n))^k)$.*

We focus on languages with simultaneous time and space constraints.

▶ **Definition 12** (**TISP**). *For functions $T, S : \mathbb{N} \to \mathbb{N}$, we say language L is in* **TISP**$[T,S]$ *if there is an algorithm, A, running in time T and space S that recognizes L.*

For space, we focus on deterministic algorithms, see the full paper [6] for the randomized and nondeterministic cases.

In an interactive protocol for some function f, we want our verifier to output $f(x)$ with high probability if the prover is honest, and outputs the wrong answer with low probability regardless of the prover. Our verifier can either reject or output some value. We use double sided completeness, since that is what we prove.

Let us formally define the interaction of a protocol.

▶ **Definition 13** (Interaction Between Verifier and Prover (Int)). *Let Σ be a alphabet and $\perp$ be a symbol not in Σ. Let V be a RAM machine with access to randomness, that can make oracle queries, and V outputs one of $\Sigma \cup \{\perp\}$. Let P' be any function, and x be an input.*

Now we define the interaction of V and P' on input x. For all i, define y_i to be V's ith oracle query given its first $i-1$ queries were answered with $z_1, \ldots, z_{i-1}$ and define $z_i = P(x, y_1, \ldots, y_i)$.

Define the output of V when interacting with P', $\text{Int}(V,P,x)$*, as the output of V on input x when its oracle queries are answered by $z_1, z_2, \ldots$.*

Now we define interactive time.

▶ **Definition 14** (Interactive Time (**ITIME**)). *Let Σ be an alphabet. If for function $f : \{0,1\}^* \to \Sigma$, soundness $s \in [0,1]$, completeness $c \in [0,1]$, verifier V and prover P we have*
Completeness: $\Pr[\text{Int}(V,P,x) = f(x)] \geq c$, *and*
Soundness: *for any function P' we have* $\Pr[\text{Int}(V,P',x) \notin \{f(x), \perp\}] \leq s$,
then we say V and P are an interactive protocol for f with soundness s and completeness c.

If in addition L is a language with $f(x) = 1_{x \in L}$, 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 $L \in \mathbf{ITIME}[T_V, T_P]$. Finally, if completeness $c = 1$, then we say the protocol has perfect completeness and $L \in \mathbf{ITIME}^1[T_V, T_P]$.

3.2 Multilinear Extensions

Sum check is generally used on Boolean functions. So first the Boolean function must be converted to a low degree polynomial, a technique broadly called arithmetization. Arithmetization is an important part of sum check [11]. Shamir [21] arithmetized Boolean formulas formulas. Boolean formulas of size C have a low degree extension of total degree C. The idea is to rewrite every $\alpha \wedge \beta$ into $\alpha \cdot \beta$, every $\alpha \vee \beta$ as $\alpha + \beta - \alpha \cdot \beta$ and every $\neg\alpha$ as $1 - \alpha$.

This indeed gives a low degree polynomial for formulas, but we use a very strong type of low degree polynomial: multilinear extensions.

▶ **Definition 15** (Multilinear Extension). *For a field $\mathbb{F}$, integer S and a function $\phi : \{0,1\}^S \to \mathbb{F}$, we define the multilinear extension of ϕ as the polynomial $\widehat{\phi} : \mathbb{F}^S \to \mathbb{F}$ that is degree 1 in any individual variable such that for all $a \in \{0,1\}^S$ we have $\phi(a) = \widehat{\phi}(a)$.*

One useful property of multilinear extensions is that unlike low degree extensions, multilinear extensions are unique. See the full version for a proof [6].

▶ **Lemma 16** (Multilinear Extension Exist and are Unique). *For a field $\mathbb{F}$, integer S and a function $\phi : \{0,1\}^S \to \mathbb{F}$, there exists $\widehat{\phi}$ that is a multilinear extension of ϕ. Further, $\widehat{\phi}$ is unique.*

For notation, we will also define the multilinear extension of matrices as the multilinear extension of the function which indexes into that matrix.

▶ **Definition 17** (Matrix Multilinear Extension). *Let S be an integer, $\mathbb{F}$ a field, and M be a $2^S \times 2^S$ matrix containing elemtents in $\mathbb{F}$. Identify an element $x \in \{0,1\}^S$ with an element of $[2^S]$ by interpreting x as a binary number.*

Then define the multilinear extension of M to be the function $\widehat{M} : \mathbb{F}^S \to \mathbb{F}^S \to \mathbb{F}$ such that $\widehat{M}$ is multilinear and for $a, b \in \{0,1\}^S$ we have $\widehat{M}(a,b) = M_{a,b}$. See that $\widehat{M}$ exists and is unique by Lemma 16.

4 Efficient IP for TISP

The main building block is a protocol to reduce a statement about a matrix squared to a statement about the matrix itself. Then I show how to combine this with arithemtization to give a protocol for bounded space.

The first step in the matrix squared to matrix reduction is the sum check protocol Lemma 10. See Appendix A for details about the sum check protocol. This reduces a statement about $\widehat{M^2}$ to a statement about a product of $\widehat{M}$.

4.1 Product Reduction

After the sum check, we need to reduce a statement about a product of $\widehat{M}$ evaluated at two points to a statement about $\widehat{M}$ evaluated at one point.

▶ **Lemma 18** (Product Reduction). *Let S be an integer, d be an integer, p be a prime, and $\mathbb{F}$ be a field with characteristic p where $|\mathbb{F}| \geq d+1$. Suppose $q : \mathbb{F}^S \to \mathbb{F}$ is a polynomial with total degree at most d. For a multi-linear polynomial, $d = S$.*

Then there is a interactive protocol with verifier V and prover P such that on input $\alpha \in \mathbb{F}$, $a, b \in \mathbb{F}^S$ behaves in the following way:

Completeness: *If $\alpha = q(a)q(b)$, then when V interacts with P, verifier V outputs some $a' \in \mathbb{F}^S$ and $\alpha' \in \mathbb{F}$ such that $\alpha' = q(a')$.*

Soundness: *If $\alpha \neq q(a)q(b)$, then for any prover P', when V interacts with P', with probability at* most $\frac{d}{|\mathbb{F}|}$ *will V output some $a' \in \mathbb{F}^S$ and $\alpha' \in \mathbb{F}$ such that $\alpha' = q(a')$.*

Verifier Time: *V runs in time $\tilde{O}(\log(|\mathbb{F}|))O(S+d)$.*

Prover Time: *P runs in time $\tilde{O}(\log(|\mathbb{F}|))S\mathbf{poly}(d)$ using $d+1$ oracle calls to q.*

Proof. The idea is to choose a line, $\psi : \mathbb{F} \to \mathbb{F}^S$, such that $\psi(0) = a$ and $\psi(1) = b$. That is, $\psi(c) = (1-c)a + cb$. Then the verifier asks the prover for the degree d polynomial $g : \mathbb{F} \to \mathbb{F}$ defined by $g(c) = q(\psi(c))$. For such a g, see that $q(a)q(b) = g(0)g(1)$.

Let g' be the degree d polynomial returned by the prover. If $\alpha \neq g'(0)g'(1)$, the verifier rejects. Otherwise, the verifier chooses a random $c \in \mathbb{F}$, sets $a' = \psi(c)$ and sets $\alpha' = g'(c)$.

Now I show this protocol has the desired properties.

Completeness: If $\alpha = q(a)q(b)$, an honest prover responds with $g' = g$, so

$$\alpha = q(a)q(b) = q(\psi(0))q(\psi(1)) = g(0)g(1) = g'(0)g'(1).$$

Then the verifier chooses c and $\alpha' = g'(c) = g(c) = q(\psi(c)) = q(a')$.

Soundness: If $\alpha \neq q(a)q(b)$, then if $g' = g$, we have $g'(0)g'(1) = q(a)q(b) \neq \alpha$, so the verifier rejects. Otherwise, $g \neq g'$, so with probability at most $\frac{d}{|\mathbb{F}|}$ does $g(c) = g'(c)$. If $g(c) \neq g'(c)$, then $\alpha' = g'(c) \neq g(c) = q(\psi(c)) = q(a')$. Thus with probability at most $\frac{d}{|\mathbb{F}|}$ does $q(a') = \alpha'$.

Verifier Time: V computes g' three times, and each time this only takes $O(d)$ field operations. V computes $\psi(c)$ once, which takes $O(S)$ field operations. Every field operation takes $\tilde{O}(\log(|\mathbb{F}|))$ time.

Prover Time: P queries q at $d+1$ points to calculate q. These $d+1$ query locations can be calculated in $O(dS)$ field operations by evaluating ψ. Then the coefficients for g can be calculated in time polynomial time in d using guassian elimination. ◀

4.2 Matrix Squared To Matrix Reduction

Now for the main lemma used in our proof. First, we need a lemma that shows $\widehat{M^2}$ can be written as a simple function of $\widehat{M}$.

▶ **Lemma 19** ($\widehat{M^2}$ is a sum of products of $\widehat{M}$). *Let S be an integer and $\mathbb{F}$ be a field. Suppose M is a $2^S \times 2^S$ matrix containing values in $\mathbb{F}_p$. Then for any $a, b \in \mathbb{F}^S$,*

$$\widehat{M^2}(a,b) = \sum_{c \in \{0,1\}^S} \widehat{M}(a,c)\widehat{M}(c,b),$$

where $\widehat{M}$ and $\widehat{M^2}$ is as defined in Definition 17.

Proof. For notation, define ψ by $\psi(a,b) = \sum_{c\in\{0,1\}^S} \widehat{M}(a,c)\widehat{M}(c,b)$ so that we are trying to prove that $\widehat{M^2} = \psi$. By Lemma 16, $\widehat{M^2} = \psi$ if and only if ψ is multilinear and they agree on all binary inputs.

To see that ψ is multilinear, we show that for any $c \in \{0,1\}^S$ we have $\widehat{M}(a,c)\widehat{M}(c,b)$ is multilinear as a function of a and b. But they are since $\widehat{M}$ is multilinear and the two calls to $\widehat{M}$ don't share any variables. Thus ψ is the sum of multilinear polynomials, and is thus multilinear itself.

To see that ψ agrees with $\widehat{M^2}$ on binary elements, take any $a, b \in \{0,1\}^S$. Then we have

$$\widehat{M^2}(a,b) = (M^2)_{a,b} = \sum_{c\in\{0,1\}^S} M_{a,c}M_{c,b} = \sum_{c\in\{0,1\}^S} \widehat{M}(a,c)\widehat{M}(c,b) = \psi(a,b).$$

So ψ must be the unique multilinear polynomial agreeing with $\widehat{M^2}$ on all binary inputs, which is $\widehat{M^2}$ itself. ◀

▶ **Lemma 20** (Matrix Squared to Matrix Protocol). [3] *Let S be an integer, p be a prime, and $\mathbb{F}$ be a field with characteristic p and $|\mathbb{F}| > 2S+1$. Suppose M is a $2^S \times 2^S$ matrix containing values in $\mathbb{F}_p$. Define $\widehat{M}$ as in Definition 17.*

Then there is a interactive protocol with verifier V and prover P such that on input $a, b \in \mathbb{F}^S$ and $\alpha \in \mathbb{F}$ behaves in the following way:

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

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

Verifier Time: *V runs in time $\tilde{O}(\log(|\mathbb{F}|))O(S)$.*

Prover Time: *P runs in time $\tilde{O}(\log(|\mathbb{F}|))2^S$ when given oracle access to $\widehat{M}$.*

Proof. The protocol is a sum check Lemma 10 followed by a product reduction Lemma 18. From Lemma 19, see that $\alpha = \widehat{M^2}(a,b) = \sum_{c\in\{0,1\}^S} \widehat{M}(a,c)\widehat{M}(c,b)$.

The first sum check either fails, or gives the claim that for some $c \in \mathbb{F}^S$, and $\beta \in \mathbb{F}$ that $\beta = \widehat{M}(a,c)\widehat{M}(c,b)$. Then the product reduction fails, or gives the claim that for some $a', b' \in \mathbb{F}^S$ and $\alpha' \in \mathbb{F}$ that $\alpha' = \widehat{M}(a',b')$.

Now we can check that this protocol gives the desired results.

Completeness: Suppose $\alpha = \widehat{M^2}(a,b)$. Then by completeness of the sum check, $\beta = \widehat{M}(a,c)\widehat{M}(c,b)$. Then by completeness of the product reduction, $\alpha' = \widehat{M}(a',b')$.

Soundness: Suppose $\alpha \neq \widehat{M^2}(a,b)$.

Let $q : \mathbb{F}^S \to \mathbb{F}^S$ be defined by $q(c) = \widehat{M}(a,c)\widehat{M}(c,a)$. That is, q is the function sum check is performed on. For any variable y, we know q is degree at most 2 in y as the product of two degree one polynomials in y.

Then by soundness of sum check, with probability at most $\frac{2S}{|\mathbb{F}|}$ will the verifier output β and c such that $\beta = \widehat{M}(a,c)\widehat{M}(c,b)$.

See that since $\widehat{M}$ is multilinear, $\widehat{M}$ has total degree at most $2S$. If $\beta \neq \widehat{M}(a,c)\widehat{M}(c,b)$, then by the soundness of the product reduction, with probability at most $\frac{2S}{|\mathbb{F}|}$ does $\alpha' = \widehat{M}(a',b')$. Thus by a union bound, with probability at most $\frac{4S}{|\mathbb{F}|}$ does $\alpha' = \widehat{M}(a',b')$.

Verifier Time: V takes the time of the sum check, plus the time of product reduction, which is $\tilde{O}(\log(|\mathbb{F}|))O(S)$.

Prover Time: P runs in the max of the time for the sum check and the time for the product reduction, which is $\tilde{O}(\log(|\mathbb{F}|))2^S$ when given oracle access to $\widehat{M}$. ◀

[3] Lemma 20 was first proven by Thaler [24].

4.3 Arithmetization

To use this matrix square reduction, we need to actually compute the multilinear extension of the adjacency matrix of the computation graph. The multilinear extensions of many simple functions are efficient to calculate, for instance, the equality function. See Appendix B for examples.

Now we show how to calculate the multilinear extension of a RAM computer's state transition function.

▶ **Lemma 9** (Algorithm Arithmetization). *Let A be a nondeterministic RAM algorithm running in space S and time T on length n inputs, and x be an input with $|x| = n$. Define M to be the $2^S \times 2^S$ matrix such that for any two states $a, b \in \{0,1\}^S$, we have $M_{a,b} = 1$ if when A is running on input x is in state a, then b as a valid transition, and $M_{a,b} = 0$ otherwise.*

Then we can compute the multilinear extension of M ($\widehat{M}$ in Definition 17) in time $\tilde{O}(\log(|\mathbb{F}|)(n+S))$.

Proof. The state of any RAM algorithm has 3 parts:

- The instruction pointer into some fixed program.
- Constant number of registers of $O(\log(n+S))$ length for performing register register operations. We assume these operations are simple operations like move, bit shift, bitwise or, bitwise and, addition, and conditional jumps in the program counter.
- S bits of memory that can be changed by the load and store commands.

Then we decompose any state, $a \in \{0,1\}^S$ into three parts $a = (p, r, m)$ where p are constantly many bits for the instruction pointer, r is $O(\log(n+S))$ many bits for the registers, and m is $O(S)$ many bits for main memory.

Suppose there are I instructions. For $i \in [I]$, let P_i be the multilinear function that identifies if the program is at instruction i. Let Q_i be the multilinear function of whether the current state on instruction i yields the next state. Then a formula for $\widehat{M}$ is given by

$$\widehat{M}((p_0, r_0, m_0), (p_1, r_1, m_1)) = \sum_{i \in [I]} P_i(p_0) Q_i(r_0, m_0, p_1, r_1, m_1).$$

Thus if we can calculate each Q_i, we can calculate $\widehat{M}$. The exact possible instructions depend on our choice of instruction set, but we cover the main ones here.

- Register to Register Arithmetic.
  Register register operations (like bit shifts, additions, etc) can be computed very efficiently. So Q_i just is the product of: the appropriate registers being updated correctly, the instruction pointer being incremented by one, all other registers being equal in r_0 and r_1, and m_0 and m_1 being equal.
- Conditionals.
  For conditional jumps, first compute equality of the state before and after, besides the program counter and the condition bit. Then multiply that by the sum over the multilinear extension of the condition bit leading to the correct location of the program counter.
- Load Input into a Register.
  A load from input instruction operates on two registers, one to store the input, and a pointer to the location in the input. The rest of the state is just an equality.
  Suppose the location you want to retrieve is indicated by $a_1, \ldots, a_{\log(n)}$, and you want to store it in variable b_1. Then the extension of this part is

$$\sum_{i \in \{0,1\}^{\log(n)}} \prod_{j \in [\log(n)]} (a_j i_j + (1-a_j)(1-i_j))(x_i b_1 + (1-x_i)(1-b_1)).$$

  This can be computed efficiently, see the full version of the paper [6].

- Load Memory into a Register.
  Basically uses the same formula as above, but uses a $\log(S)$ bit address and uses memory instead of the input. That is, replace x with m_0. One slight technicality is that we must also assert that $m_0 = m_1$. That is, main memory doesn't change. Specifically, we calculate

$$\sum_{i\in\{0,1\}^{\log(S)}} \left(\prod_{j\in[\log(S)]} (a_j i_j + (1-a_j)(1-i_j)) \right) ((m_0)_i (m_1)_i b_1 + (1-(m_0)_i)(1-(m_1)_i)(1-b_1)) \prod_{k\in\{0,1\}^{\log(S)}\setminus\{i\}} ((m_0)_k (m_1)_k + (1-(m_0)_k)(1-(m_1)_k)).$$

- Store Register into Memory.
  This is essentially the same as loading, except instead of saying the future register should be equal to the current memory, we instead say the future memory is equal to the current register.

Thus each Q_i can be efficiently calculated in $O(S+n)$ field operations, so $\widehat{M}$ can be calculated in time $\tilde{O}(\log(|\mathbb{F}|))O(S)$. ◀

4.4 Protocol for TISP

Now we give our protocol for deterministic algorithms.

▶ **Theorem 1** (Efficient Interactive Protocol For **TISP**). *Let S and T be computable in time $\tilde{O}(\log(T)S+n)$ with $S = \Omega(\log(n))$. Then*

$$\mathbf{TISP}[T,S] \subseteq \mathbf{ITIME}^1[\tilde{O}(\log(T)S+n), 2^{O(S)}].$$

Proof. We start by outlining how to convert acceptance to a matrix problem, then we show how to apply Lemma 20 to solve that. Let a be the canonical starting state of algorithm A, and b the canonical accept state.

Take T to be a power of two so that $T = 2^t$. If T is not a power of 2, we can take T to be the smallest power of two greater than the original T. Let k be the smallest integer so that $2^k > 12\log(T)S$. Let $q = 2^k$ and $\mathbb{F}$ be the field with q elements. Note that $|\mathbb{F}| \leq 24\log(T)S$.

Let M be the adjacency matrix for the computation graph of A on input x so that for any two states, a' and b', we have $M_{a',b'} = 1$ if A on input x starting in state a' can be b' after one step, and $M_{a',b'} = 0$ otherwise. For any matrix M', let $\widehat{M'}$ be the multilinear extension of M' so that for all binary inputs a' and b' we have $\widehat{M'}(a',b') = M'_{a',b'}$.

Observe that $M^{2^i}_{a',b'} = 1$ if when A on input x is in state a' after 2^i steps is in state b', and $M^{2^i}_{a',b'} = 0$ otherwise. Thus our verifier wants to output $M^T(a,b)$, or $\widehat{M^{2^t}}(a,b)$.

In the full protocol, the prover first provides the verifier with a candidate $\alpha \in \{0,1\}$ with the claim that $\alpha = \widehat{M^{2^t}}(a,b)$. Now we use Lemma 20 t times to get the claim that for some $\alpha' \in \mathbb{F}$ and some $a', b' \in \mathbb{F}^S$ we have $\alpha' = \widehat{M}(a',b')$. Finally, the verifier uses Lemma 9 to calculate $\widehat{M}(a',b')$ and compare it with α'.

Completeness: For an honest prover, indeed $\alpha = \widehat{M^{2^t}}(a,b)$. Let $\alpha_0 = \alpha$, $a_0 = a$ and $b_0 = b$. By induction and completeness of Lemma 20, for every $i \in [0, t-1]$ we have $\alpha_i = M^{2^{t-i}}(a_i, b_i)$, and our protocol gives an α_{i+1}, a_{i+1}, and b_{i+1} such that $\alpha_{i+1} = M^{2^{t-(i+1)}}(a_{i+1}, b_{i+1})$. Thus $\alpha_t = \widehat{M}(a_t, b_t)$. Thus the verifier check whether $\alpha_t = \widehat{M}(a_t, b_t)$ succeeds, and the verifier outputs α.

Soundness: If $\alpha = \widehat{M^{2^t}}(a,b)$, the verifier either outputs α or rejects, either satisfies our assumption. So suppose $\alpha \neq \widehat{M^{2^t}}(a,b)$. Let $\alpha_0 = \alpha$, $a_0 = a$ and $b_0 = b$. By induction and soundness of Lemma 20, for every $i \in [0, t-1]$ if the verifier hasn't rejected and $\alpha_i \neq \widehat{M^{2^{t-i}}}(a_i, b_i)$, the probability the verifier does not reject and $\alpha_{i+1} = \widehat{M^{2^{t-(i+1)}}}(a_{i+1}, b_{i+1})$ is at most $\frac{4S}{|\mathbb{F}|}$. By a union bound, the probability that the verifier does not reject and for any i we have $\alpha_i = \widehat{M^{2^{t-i}}}(a_i, b_i)$ is at most $\frac{4S\log(T)}{|\mathbb{F}|} \leq \frac{1}{3}$.

In particular, the probability the verifier does not reject and $\alpha_t = \widehat{M}(a_t, b_t)$ is at most $\frac{1}{3}$. See that if $\alpha_t \neq \widehat{M}(a_t, b_t)$, then the verifier rejects. So the probability the verifier does not reject is at most $\frac{1}{3}$.

Verifier Time: This verifier takes time $\tilde{O}(\log(|\mathbb{F}|))S\log(T)$ to run Lemma 20 t times plus $\tilde{O}(\log(|\mathbb{F}|))O(S+n)$ to calculate $\widehat{M}(a_t, b_t)$ (using Lemma 9), so in total takes time $\tilde{O}(\log(|\mathbb{F}|))(S\log(T)+n) = \tilde{O}(\log(S))O(\log(T)S + n)$.

Prover Time: First, the prover calculates M^{2^i} in time $\tilde{O}(\log(|\mathbb{F}|))S2^{3S}$ for every $i \in [t]$. Then we can query the multilinear extension of these matrices, which is $\widehat{M^{2^i}}$, at any location in time $\tilde{O}(\log(|\mathbb{F}|))2^{2S}$ with the formula given in Lemma 16.

Finally, given oracle access to each $\widehat{M^{2^i}}$, the prover in Lemma 20 only takes time $\tilde{O}(\log(|\mathbb{F}|))2^S$. Thus of any call to Lemma 20 only takes time $\mathbf{poly}(\log(|\mathbb{F}|))2^{3S}$. Thus the prover runs in time $2^{O(S)}$. ◀

We can extend this result into multi bit outputs by storing the output in the final end state and asking the prover for the end state first. Or to be more efficient when outputs are larger than S, we can first ask for the output. Then include the output in the input and change the algorithm to instead verify the output is correct. Then run the protocol on this new verification algorithm instead. This allows us to verify program outputs larger than S more efficiently.

For the randomized and nondeterministic cases, see the full paper [6].

5 Open Problems

Here are some open problems.

1. Could one remove the dependence on $\log(T)$? Is it true that, for $S = \omega(n)$, we have $\mathbf{SPACE}[S] \subseteq \mathbf{ITIME}[\tilde{O}(S)]$?
   This would prove an equivalence, up to polylogarithmic factors, between $\mathbf{SPACE}[S]$ and $\mathbf{ITIME}[S]$. Our recent work, [13], showed a similar equivalence between $\mathbf{NTIME}[T]$ and languages verified by $\mathbf{PCP}$s with $\log(T)$ time verifiers, for $\log(T) = \Omega(n)$.
2. Is there a strong hierarchy theorem for interactive time? Can we show that for any T we have $\mathbf{ITIME}[\tilde{O}(T)] \not\subset \mathbf{ITIME}[T]$? Here $\mathbf{ITIME}[T]$ is the class of languages with an interactive protocol who's verifiers run in time T.
   Using Theorem 1 gives $\mathbf{ITIME}^1[T] \subseteq \mathbf{SPACE}[O(T)] \subseteq \mathbf{ITIME}^1[\tilde{O}(T^2)]$. This gives a hierarchy theorem, $\mathbf{ITIME}^1[\tilde{O}(T^2)] \not\subset \mathbf{ITIME}^1[O(T)]$, by using the space hierarchy theorem. Improving the interactive protocols for bounded space is one way to give a stronger hierarchy.
3. Can interactive protocols for $\mathbf{NTISP}$ be as efficient as those for $\mathbf{TISP}$?
   For $S = \Omega(n)$, we get $\mathbf{SPACE}[S] \subseteq \mathbf{ITIME}[\tilde{O}(S^2)]$, but only show $\mathbf{NSPACE}[S] \subseteq \mathbf{ITIME}[\tilde{O}(S^3)]$. Does nondeterministic space require more verifier time than deterministic space?

4. Can we get double efficiency with a similar verifier time? Is it true that $\mathbf{TISP}[T, S] \subseteq \mathbf{ITIME}[\mathbf{polylog}(T)S, \mathbf{poly}(T)]$?
   This would make the verification of polynomial time algorithms only take as long (up to polylogairthmic factors) as the space of those algorithms, with a proof that still only takes polynomial time. This would make directly using interactive proofs for delegating computation more practical.

References

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

2 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, 1998. doi:10.1145/278298.278306.

3 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.

4 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.

5 Lijie Chen and Roei Tell. Hardness vs randomness, revised: Uniform, non-black-box, and instance-wise. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 125–136, 2022. doi:10.1109/FOCS52979.2021.00021.

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

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

8 Shafi Goldwasser, Dan Gutfreund, Alexander Healy, Tali Kaufman, and Guy N. Rothblum. Verifying and decoding in constant depth. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 440–449. Association for Computing Machinery, 2007. doi:10.1145/1250790.1250855.

9 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.

10 Neil Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988. doi:10.1137/0217058.

11 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.

12 Or Meir. IP = PSPACE using error-correcting codes. *SIAM Journal on Computing*, 42(1):380–403, 2013. doi:10.1137/110829660.

13 Dana Moshkovitz and Joshua Cook. Tighter ma/1 circuit lower bounds from verifier efficient pcps for pspace, 2022. URL: https://eccc.weizmann.ac.il/report/2022/014/.

14 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.

15 Noam Nisan. Pseudorandom generators for space-bounded computations. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, STOC '90, pages 204–212. Association for Computing Machinery, 1990. doi:10.1145/100216.100242.

16 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.

17 Noga Ron-Zewi and Ron D. Rothblum. Proving as fast as computing: Succinct arguments with constant prover overhead. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 1353–1363. Association for Computing Machinery, 2022. doi:10.1145/3519935.3519956.

18 Michael Saks and Shiyu Zhou. $\mathrm{BP}_H\mathrm{SPACE(S)} \subseteq \mathrm{DSPACE}(\mathrm{S}^{3/2})$. *J. Comput. Syst. Sci.*, 58(2):376–403, April 1999. doi:10.1006/jcss.1998.1616.

19 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.

20 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, April 1970. doi:10.1016/S0022-0000(70)80006-X.

21 Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, October 1992. doi:10.1145/146585.146609.

22 A. Shen. IP = SPACE: Simplified Proof. *J. ACM*, 39(4):878–880, 1992. doi:10.1145/146585.146613.

23 Róbert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988. doi:10.1007/BF00299636.

24 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.

A Sum Check

Sum check [11] takes a claim that for some $\alpha \in \mathbb{F}$, we have $\alpha = \sum_{s\in\{0,1\}^s} q(s)$, and reduces it to the claim that for some $\beta \in \mathbb{F}$ and for some random $r \in \mathbb{F}^S$ we have $\beta = q(r)$.

▶ **Lemma 10** (Sum Check Protocol). *Let S be an integer, d be an integer, p be a prime, and $\mathbb{F}$ be a field with characteristic p where $|\mathbb{F}| \geq d+1$. Suppose $q : \mathbb{F}^S \to \mathbb{F}$ be a polynomial with degree at most d in each variable individually. For a multi-linear polynomial, $d = 1$.*

Then there is a interactive protocol with verifier V and prover P such that on input $\alpha \in \mathbb{F}$ behaves in the following way:

Completeness: *If $\alpha = \sum_{a\in\{0,1\}^S} q(a)$, then when V interacts with P, verifier V outputs some $a' \in \mathbb{F}^S$ and $\alpha' \in \mathbb{F}$ such that $\alpha' = q(a')$.*

Soundness: *If $\alpha \neq \sum_{a\in\{0,1\}^S} q(a)$, then for any prover P', when V interacts with P', with probability at* most $\frac{dS}{|\mathbb{F}|}$ *will V output some $a' \in \mathbb{F}^S$ and $\alpha' \in \mathbb{F}$ such that $\alpha' = q(a')$.*

Verifier Time: *V runs in time $\tilde{O}(\log(|\mathbb{F}|))O(Sd)$.*

Prover Time: *P runs in time $\tilde{O}(\log(|\mathbb{F}|))\mathbf{poly}(d)2^S$ using $O(d2^S)$ oracle queries to q.*

Sketch. The idea is to choose the random field elements for a' one at a time and consider the partial sums. Let $\alpha_0 = \alpha$. For every i, we want to reduce a claim that

$$\alpha_i = \sum_{a\in\{0,1\}^{S-i}} p(a'_1, \ldots, a'_i, a),$$

to a claim that

$$\alpha_{i+1} = \sum_{a\in\{0,1\}^{S-(i+1)}} p(a'_1, \ldots, a'_i, a'_{i+1}, a).$$

The observation is that if we define $g_i : \mathbb{F} \to \mathbb{F}$ by

$$g_i(x) = \sum_{a\in\{0,1\}^{S-(i+1)}} p(a'_1, \ldots, a'_i, x, a)$$

then g_i is a low degree polynomial. The verifier asks for g_i, and checks if $g_i(0) + g_i(1) = \alpha_i$. If it doesn't, the verifier knows the prover lied and rejects. Otherwise, it chooses r_{i+1} and sets $\alpha_{i+1} = g_i(r_{i+1})$.

If g_i is correct, then $\alpha_{i+1} = \sum_{s' \in \{0,1\}^{S-(i+1)}} p(r_1, \ldots, r_i, r_{i+1}, s')$. Thus for an honest prover, this equality will hold for every i, and the completeness holds.

Now if at any step $\alpha_i \neq \sum_{s \in \{0,1\}^{S-i}} p(r_1, \ldots, r_i, s)$, the prover cannot provide the correct g_i, or the verifier will see that $g_i(0) + g_i(1) \neq \alpha_i$ and reject. But if g_i is incorrect, by Schwartz-Zippel, the probability the provided g_i agrees with the true g_i is $\frac{d}{|\mathbb{F}|}$. If they disagree at r_{i+1}, then $\alpha_{i+1} \neq \sum_{s' \in \{0,1\}^{S-(i+1)}} p(r_1, \ldots, r_i, r_{i+1}, s')$. So by a union bound, the probability the verifier ever has a correct α_i is at most $\frac{Sd}{|\mathbb{F}|}$. ◀

This is a commonly taught protocol, and can be found in "Computational Complexity: A Modern Approach" [1].

B Arithmetization Examples

Here are some examples of efficient multilinear extensions.

▶ **Lemma 21** (Equality has Efficient Multilinear Extension). *Let $\phi : \{0,1\}^S \times \{0,1\}^S \to \{0,1\}$ be the Boolean function such that $\phi(a,b) = 1$ if and only if $a = b$.*

Then for any field $\mathbb{F}$, the multilinear extension of ϕ, denoted $\widehat{\phi}$, can be calculated in time $\tilde{O}(\log(|\mathbb{F}|))O(S)$.

Proof. One can write the formula for $\widehat{\phi}$ as

$$\widehat{\phi}(a,b) = \prod_{i \in [S]} \left(a_i b_i + (1-a_i)(1-b_i)\right).$$

To see the above equality, see that when a and b are restricted to binary variables, the right hand side is one if and only if for every i we have $a_i = b_i$. Further, the right hand side is degree one any individual a_i or b_i.

Further, it can be calculated in time $\tilde{O}(\log(|\mathbb{F}|))O(S)$ since each field operation only takes time $\tilde{O}(\log(|\mathbb{F}|))$, and as written, the function only needs $O(S)$ field operations. ◀

▶ Remark 22. One can easily extend this to check equality of 3, 4, or any arbitrary number of variables. Extra equality checks are often easy to add. To assert $d = e$ in most formulas, just replace every occurrence of d with de and $(1-d)$ with $(1-d)(1-e)$. This works as long as the formula can be rewritten as a sum of products where every product has either d or $1-d$ exactly once.

This is true for equality, cyclic bit shifting, and even addition.

Similarly, inequality, or any constant bit shift can also be computed efficiently. For a more complex example, we can efficiently compute the multilinear extension of binary addition.

▶ **Lemma 23** (Binary Addition has Efficient Multilinear Extension). *Let $\phi_S : \{0,1\}^S \times \{0,1\}^S \times \{0,1\}^S \to \{0,1\}$ be the Boolean function such that $\phi(a,b,c) = 1$ if and only if $a + b = c$ where addition is binary and we ignore the final carry.*

Then for any field $\mathbb{F}$, the multilinear extension of ϕ_S, $\widehat{\phi_S}$ can be calculated in time $\tilde{O}(\log(|\mathbb{F}|))O(S)$.

Proof. We prove this recursively. Define $\psi_S : \{0,1\}^S \times \{0,1\}^S \times \{0,1\}^S \to \{0,1\}$ as the Boolean function such that $\psi(a,b,c) = 1$ if and only if $a + b + 1 = c$ where addition is binary and we ignore the final carry. Let $\widehat{\psi_S}$ be the multilinear extension of ψ_S. The idea is to implement a ripple carry adder.

Now we will compute $\widehat{\psi}_S$ and $\widehat{\phi}_S$ by induction. See that

$$\begin{aligned}\widehat{\phi}_1(a,b,c) =&(a(1-b)+b(1-a))c+(ab+(1-a)(1-b))(1-c)\\ \widehat{\psi}_1(a,b,c) =&(a(1-b)+b(1-a))(1-c)+(ab+(1-a)(1-b))c.\end{aligned}$$

These are multilinear as written, and correctly compute ψ_1 and ϕ_1 when given binary inputs. Each of these only require a constant number of field operations.

Assume for $a',b',c' \in \mathbb{F}^{S-1}$, we have computed $\widehat{\phi}_{S-1}(a',b',c')$ and $\widehat{\psi}_{S-1}(a',b',c')$. We show how to calculate $\widehat{\psi}_S$ and $\widehat{\phi}_S$. For $a,b,c \in \mathbb{F}$, I claim that

$$\begin{aligned}\widehat{\phi}_S((a,a'),(b,b'),(c,c')) =&\widehat{\phi}_{S-1}(a',b',c')(1-a)(1-b)(1-c)\\ &+\widehat{\phi}_{S-1}(a',b',c')(a(1-b)+(1-a)b)c\\ &+\widehat{\psi}_{S-1}(a',b',c')ab(1-c)\\ \widehat{\psi}_S((a,a'),(b,b'),(c,c')) =&\widehat{\phi}_{S-1}(a',b',c')(1-a)(1-b)c\\ &+\widehat{\psi}_{S-1}(a',b',c')(a(1-b)+(1-a)b)(1-c)\\ &+\widehat{\psi}_{S-1}(a',b',c')abc.\end{aligned}$$

These are multilinear by induction. For binary inputs, this implements a ripple carry adder. For instance, if $a=b=0$, then $\widehat{\phi}((a,a'),(b,b'),(c,c'))$ is one if and only if $c=0$ and, by induction, $a'+b'=c'$. If a and b are zero, than c should be 0 and there is no carry. The rest of the cases can be checked similarly

Given that $\widehat{\psi}_{S-1}$ and $\widehat{\phi}_{S-1}$ were already calculated, it only requires constantly many more field operations to calculate $\widehat{\psi}_S$ and $\widehat{\phi}_S$. Thus, $\widehat{\psi}_S$ and $\widehat{\phi}_S$ only require $O(S)$ field operations. Thus $\widehat{\psi}_S$ only takes time $\tilde{O}(\log(|\mathbb{F}|))S$ to calculate. ◀

Note, we could modify this to give a multilinear extension of the function $\phi' : \{0,1\}^{5S} \to \{0,1\}$ that not only checks if $a+b=c$, but also whether $a=d$ and $b=e$, as described in Remark 22.

One can compute the multilinear extensions of other register, register operations in a simple register RAM model computer.