Electronic Colloquium on Computational Complexity, Revision 4 of Report No. 93 (2022)

More Verifier Efficient Interactive Protocols for Bounded Space

Joshua Cook[*]

January 8, 2023

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 [Sha92; Lun+90]:

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

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 [GKR15]:

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

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.

[*]jac22855@utexas.edu. Department of Computer Science, UT Austin. This material is based upon work supported by the National Science Foundation under grant numbers 1705028 and 2200956.

ISSN 1433-8092

Contents

1 Introduction **3**
1.1 Results 3
1.2 Related Work 4

2 Proof Idea **6**
2.1 Computation Graphs 8
2.2 Arithmetization and Low Degree Extensions 8
2.3 Sum Check 9
2.4 Product Reduction 10
2.5 Protocol For Deterministic Algorithms 10
2.6 Number Of Paths Mod Prime 10
2.7 Randomized and Non Deterministic Algorithms 11

3 Preliminaries **11**
3.1 Complexity Classes 11
3.2 Prime Testing and PRG 13
3.3 Sum Check 14
3.4 Multilinear Extensions 15

4 Efficient IP for TISP **16**
4.1 Product Reduction 17
4.2 Matrix Squared To Matrix Reduction 18
4.3 Arithmetization 20
4.4 Number of Paths Mod a Prime 24
4.5 Protocol for **TISP** 26

5 Efficient IP for BPTISP **26**

6 Efficient IP for NTISP **28**
6.1 Finding Good Primes With High Probability 28
6.2 Improving Paths Mod P for Small Space 29
6.3 Proving **IP** for **NTISP** 30

7 Open Problems **32**

A Efficient Arithmetization of Multitape Turing Machine **35**

1 Introduction

One of the most celebrated results of computer science is the proof that $\mathbf{IP} = \mathbf{PSPACE}$ [Sha92; Lun+90]. 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 $\mathbf{MA}/1$ [San07], and for **NQP** [MW18]. More verifier time efficient PCPs [MC22] improve the results of [San07]. Even pseudo random generators [CT22] use interactive proofs [GKR15].

The previous best verifier time in an interactive protocol for an algorithm running in time T and space S was by Shamir [Sha92], 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 [GKR15], 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 [Sha92] using sum check [Lun+90]. 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 [RRR16]). 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 [Nis90], 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 $\tilde{O}(\log(T)^2 S + \log(T)n)$ verifier. One can also use Saks and Zhou [SZ99] to reduce $\mathbf{BPSPACE}[S]$ to $\mathbf{SPACE}[S^{1.5}]$, then apply an $\mathbf{IP}$, 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 $\tilde{O}(\log(T)^2 S + n)$ with $S = \Omega(\log(n))$. Then*

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

While $\mathbf{NSPACE}[O(S)] = \mathbf{coNSPACE}[O(S)]$ [Imm88; Sze88], the same result is not known for time bounded computation. So the case for **NTISP** and **coNTISP** are both interesting and potentially different.

1.2 Related Work

This work builds on techniques used by Lund, Fortnow, Karloff and Nisan [Lun+90] to prove that $\#\mathbf{P} \in \mathbf{IP}$ and extended by Shamir [Sha92] to show that $\mathbf{PSPACE} = \mathbf{IP}$. Shamir used Savitch's theorem [Sav70] to reduce space bounded computation to a quantified Boolean formula, and then gave a sum check similar to [Lun+90] to verify it.

Although it was not shown, the bounded space variation of Shamir's protocol (given at the end of [Sha92]) 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*

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

[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.

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 [She92] gave a highly influential variation of Shamir's proof that is frequently taught (for instance [AB09]). Shen's result gives a less efficient verifier. Meir gave a proof that **IP** = **PSPACE** [Mei13] 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 [RZR22].

Sum check was also used by Babai, Fortnow and Lund [BFL90] to prove that **MIP** = **NEXP**. This line of work is foundational to many PCPs [AS98; Aro+98].

In a very influential paper, Goldwasser, Kalai and Rothblum gave doubly efficient interactive proofs for depth bounded computation [GKR15]. 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 [RRR16] 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 [Gol18], confirmed by the authors of [RRR16]. 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 [RRR16] 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 [Gol+07] studied the computation depth

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

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.

Here is a table that compares different protocols.

	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	$\log(T)S+n$	$\log(T)S+n$	$\log(T)^2 S+n$	$2^{O(S)}$

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.

Thaler [Tha13] gave a sum check for matrices that is very similar to the one we use. For matrices A, B and C, he gave a protocol for proving $AB = C$. In fact, he briefly describes the same interactive protocol as Lemma 28 with the suggestion to use it on repeated squaring. But Thaler never applied this result to space bounded computations.

2 Proof Idea

Our protocol uses sum check [Lun+90], but unlike Shamir, Shen, and Meir [Sha92; She92; Mei13], 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. Although, this also works with multitape Turing Machines with a read only tape.

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 simplicity, we explain the proof for deterministic algorithms first. 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 25.

   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 [Lun+90], 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 [GKR15] 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, before showing how to use this protocol for randomized or nondeterministic algorithms.

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 non zero. Since we consider nondeterministic algorithms, $(M^T)_{a,b}$ will contain the number of computation paths ending in b. Since we work in a finite field with characteristic p, entry $(M^T)_{a,b}$ will contain the number of computation paths mod p.

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 arihemtizing 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 24). 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 25.

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 25) in time $\tilde{O}(\log(|\mathbb{F}|))(n+S)$ and space $O(\log(|\mathbb{F}|)\log(S+n))$. Alternatively, $\widehat{M}$ can be calculated in time $\tilde{O}(\log(|\mathbb{F}|))(n+S)^2$ and space $O(\log(|\mathbb{F}|) + \log(S+n))$.

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. For completeness, arithmetization of a multitape TM with a read only input tape is included in Appendix A.

2.3 Sum Check

The key part of our protocol is the sum check from LFKN [Lun+90]. 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 $\tilde{O}(\log(|\mathbb{F}|))O(Sd)$.*

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

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 Section 3.3 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 every time 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. [GKR15] 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 26 for more details.

This is the only step the prover is asked to provide a polynomial with degree more than 2. This step is somewhat analogous to what the for-all quantifiers do in Shamir's proof, except that our degree doesn't increase in the way Shamir's does. So we don't need degree reduction or require some simplified form.

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.

2.6 Number Of Paths Mod Prime

Now consider if A is not a deterministic algorithm, but a nondeterministic algorithm. What does this change? Before, for every state in a, there was exactly one state b so that $M_{a,b} = 1$. In a nondeterministic algorithm, there may be many such b. Thus $M^T_{a,b}$ is actually the number of computation paths of length T from state a to state b. Or if b is the unique accept state and a the start state, the number of proofs of $A(x)$.

But we are not working with M over the integers, we are working with M over a finite field, $\mathbb{F}$. So if $\mathbb{F}$ has characteristic p, the entries in M^T only contain the number of paths of length T, mod p. The protocol works for ANY matrix M, and any original claim that $\alpha = \widehat{M^T}(a,b)$, regardless of whether α is 0, 1, or anything else in $\mathbb{F}$. As long as we can calculate $\widehat{M}$ efficiently, this protocol works. The arithmetization of RAM algorithms works just as well for nondeterministic RAM algorithms.

2.7 Randomized and Non Deterministic Algorithms

To handle randomized algorithms, we use Nisan's pseudo random generator (PRG) for bounded space computation. This PRG has seed length $O(\log(T)S)$ and can be calculated in time $\mathbf{poly}(S)$ and space $O(S)$. After choosing a seed, this PRG gives us a length $n + O(\log(T)S)$ input for a deterministic algorithm running in space $O(S)$ and time $\mathbf{poly}(T)$ which agrees with the randomized algorithm with high probability. Now we can run our deterministic protocol on this input.

To handle nondeterministic algorithms, we choose a random prime p. The protocol works if p does not divide the number of accepting computation paths. Let Q be the set of primes between $m = 100T$ and $2m = 200T$. If $w \leq 2^T$ is the number of accepting paths, then the number of prime factors of w in Q is at most $\frac{T}{\log(m)}$. By the prime number theorem, for large enough T, set Q should contain at least $\frac{0.5m}{\ln(m)}$ elements. If we can randomly sample one p from Q and $w \neq 0$, the probability $w \mod p = 0$ is at most $\frac{1}{10}$.

One final issue in the case of the nondeterministic algorithm is that when S is small, we cannot afford to calculate $\widehat{M}$ as this would take time $\tilde{\Omega}(\log(T)n)$, which may be larger than $\tilde{O}(\log(T)^2 S + n)$ for small S and large T. So instead of calculating it directly, we use our interactive protocol for deterministic algorithms to verify it's value for us.

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. For completeness, we also present the arithmetization of a multitape Turing Machine in Appendix A. 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 $\mathbf{TISP}[T, S]$ if there is an algorithm, A, running in time T and space S that recognizes L.*

Since we are working with space bounded computation, we define access to randomness through a read once input.

Definition 13 (Read Once Input)**.** *We say algorithm A uses read once input W if there is a special instruction in A that, for all i, on the ith time being called loads the ith symbol of W into a register.*

If A uses an input x and a read once input W, we define $A(x, W)$ as the output of A when run with input x and read once input W.

Note if A runs in time T, we can upper bound the size of W as $|W| \leq T$.

Now we define **BPTISP**.

Definition 14 (**BPTISP**)**.** *Let L be a language and A be an algorithm with read once input. If for all x we have $\Pr[A(x, U) = 1_{x \in L}] \geq \frac{2}{3}$, where U is the uniform distribution, then we say A is a randomized algorithm for L.*

*If for $T, S : \mathbb{N} \to \mathbb{N}$ algorithm A runs in time T and space S, then $L \in$ **BPTISP**$[T, S]$.*

We also define **NTISP** and **coNTISP** using a similar instruction as Definition 13. This allows nondeterministic algorithms to have multiple valid transitions. We must be careful though as the number of witnesses using a witness definition may be different from the number of accepting paths. For instance, if the entire witness is not always read.

Definition 15 (Nondeterministic Algorithms)**.** *We say algorithm A uses nondeterminism if there is a special instruction in A such that when it is called, there is a valid state transitions to 2 program states. Such an A is called a nondeterministic algorithm.*

We call a sequence of memory states such that every pair of adjacent states is a valid state transition for algorithm A on input x a path in the computation graph of A on x, or a computation path.

We call a path in the computation graph of A on input x an "accepting path" if it starts at the start state and ends at the accept state. Then we say A accepts x if and only if there is an accepting path.

Now we can define **NTISP** and **coNTISP** in an analogues way to **TISP** but for nondeterministic algorithms.

Definition 16 (**NTISP** and **coNTISP**)**.** *Let L be a language, and A be a nondeterministic algorithm running such that $x \in L$ if and only if A accepts x.*

*If for some $T, S : \mathbb{N} \to \mathbb{N}$ nondeterministic algorithm A runs in time T and space S, then $L \in$ **NTISP**$[T, S]$. We say $L \in$ **coNTISP**$[T, S]$ if the complement of L is in **NTISP**$[T, S]$.*

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 17 (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 18 (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: $\Pr[\mathrm{Int}(V, P, x) = f(x)] \geq c$, *and*

Soundness: *for any function P' we have* $\Pr[\mathrm{Int}(V, P', x) \notin \{f(x), \perp\}] \leq s$,

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 Prime Testing and PRG

For primality, we use the Rabin-Miller [Mil75; Rab80] primality test. It is well known using fast algorithms for multiplication that we can get the following result:

Theorem 19 (Miller-Rabin Primality Test)**.** *There is a randomized algorithm, A, that given an n bit number a and $\epsilon > 0$ runs in time $\tilde{O}(\log(\frac{1}{\epsilon})n^2)$ such that if a is prime, A outputs 1, and if a is not prime, A outputs 1 with probability at most ϵ.*

Then by choosing $n = O(\log(m))$ random numbers between m and $2m$, by the prime number theorem, we select a prime number with high probability.

Theorem 20 (Miller-Rabin Probably Prime Generation)**.** *There is a randomized algorithm, A, that when given a number m (with $n = \log(m)$) and $\epsilon > 0$, algorithm A runs in time $\tilde{O}(\mathbf{polylog}(\frac{1}{\epsilon})n^3)$ and with probability $1 - \epsilon$ outputs a uniform prime between m and $2m$.*

We use Nisan's PRG for space bounded computation [Nis90]. First, we define a PRG.

Definition 21 (Pseudo Random Generator (PRG))**.** *For integers n and s, we say that a function $G : \{0,1\}^s \to \{0,1\}^n$, is a Pseudo Random Generator (*PRG*) with seed length s that ϵ fools function $F : \{0,1\}^n \to \{0.1\}$ if*

$$|\mathbb{E}[F(G(U_s)) - F(U_n)]| \leq \epsilon$$

where U_s and U_n are the uniform distribution over s and n bits respectively.

Specifically, we say G fools randomized algorithm A if for all x we have

$$|\mathbb{E}[A(x, G(U_s)) - A(x, U_n)]| \leq \epsilon.$$

Then Nisan gives an efficient PRG which fools algorithms that use small space.

Theorem 22 (Nisan's PRG)**.** *For any space S, time T and error $\epsilon > 0$, there exists a* PRG *with seed length $O(S \log(T/\epsilon))$ computed by an algorithm running in time $\mathbf{poly}(S)$ and space $O(S)$ that ϵ fools randomized algorithms running in time T and space S.*

3.3 Sum Check

Sum check [Lun+90] takes a claim that for some $\alpha \in \mathbb{F}$,

$$\alpha = \sum_{s \in \{0,1\}^S} q(s),$$

and reduces it to the claim that for some $\beta \in \mathbb{F}$ and for some random $r \in \mathbb{F}^S$

$$\beta = q(r).$$

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 $\tilde{O}(\log(|\mathbb{F}|))O(Sd)$.*

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

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" [AB09].

3.4 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 [Lun+90]. Shamir [Sha92] 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 23 (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.

Lemma 24 (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.*

Proof. We can define $\widehat{\phi}$ with the following formula:

$$\widehat{\phi}(a) = \sum_{c \in \{0,1\}^S} \left(\prod_{i \in [S]} a_i c_i + (1 - a_i)(1 - c_i) \right) \phi(c).$$

First, I show $\widehat{\phi}$ is degree 1 in any given variable. Consider variable a_i. For any c, variable a_i only appears in one term in the product, so these products are linear in variable a_i. And since $\widehat{\phi}$ is the sum of linear functions in a_i, $\widehat{\phi}$ is a linear function of a_i. So $\widehat{\phi}$ is multilinear.

Now for any $a \in \{0,1\}^S$, for any $c \in \{0,1\}^S$, if $c \neq a$, then for some $i \in [S]$, we have $a_i c_i + (1-a_i)(1-c_i) = 0$. So $\left(\prod_{i\in[S]} a_i c_i + (1-a_i)(1-c_i)\right) = 0$. If $c = a$, then for each $i \in [S]$ we have $a_i c_i + (1-a_i)(1-c_i) = 1$. Thus the only term in the sum that is non zero is when $a = c$, which is equal to $\phi(c)$. Thus $\widehat{\phi}(a) = \phi(a)$.

Therefore, $\widehat{\phi}$ is a multilinear extension of ϕ.

We show $\widehat{\phi}$ is unique with a counting argument. See that by the construction above each $\phi : \{0,1\}^S \to \mathbb{F}$ has at least one multilinear extension, and every multilinear function is the extension of at most one ϕ, namely the one it agrees with on binary inputs. Further, there are $|\mathbb{F}|^{2^S}$ different such ϕ. So if any ϕ had multiple multilinear extensions, there would be more than $|\mathbb{F}|^{2^S}$ multilinear functions.

Now I show there are only $|\mathbb{F}|^{2^S}$ multilinear functions. Any multilinear function can be expanded to a sum of monomials. Each monomial has any of the S variables at most once, since our function is linear in each variable. So for each monomial, for each variable, either that variable is in that monomial, or not. So there are at most 2^S monomials. The coefficient on each of these monomials can be set to any element of $\mathbb{F}$ independently. So there are at most $|\mathbb{F}|^{2^S}$ multilinear functions.

Thus no function ϕ can have more than one multilinear extension, or there would be more than $|\mathbb{F}|^{2^S}$ multilinear functions. □

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

Definition 25 (Matrix Multilinear Extension)**.** *Let S be an integer, $\mathbb{F}$ a field, and M be a $2^S \times 2^S$ matrix containing elements 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 24.

4 Efficient IP for TISP

We first give a protocol that outputs the number of accepting paths in a non deterministic algorithm, mod a prime. This directly implies a protocol for **UTISP** (nondeterministic algorithms with a unique proof), which contains **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 arithmetization 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 Section 3.3 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 26 (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 $\tilde{O}(\log(|\mathbb{F}|))O(S+d)$.*

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

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.

Verifier Space: V only needs space for $O(S)$ field elements which store a and a', $O(d)$ field elements to store g', and one field element to store c. This takes $O(\log(|\mathbb{F}|)(S + d))$ space to store.

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 Gaussian 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 27 ($\widehat{M^2}$ 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 25.

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 24, $\widehat{M^2} = \psi$ 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

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

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

Lemma 28 (Matrix Squared to Matrix Protocol)**.** *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 25.*

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 $\tilde{O}(\log(|\mathbb{F}|))O(S)$.*

Verifier Space: *V runs in space $O(\log(|\mathbb{F}|)S)$.*

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 26. From Lemma 27, see that

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

Then 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 $\tilde{O}(\log(|\mathbb{F}|))O(S)$.

Verifier Space: V runs in space which is the max of the sum check and the product reduction, which is $O(\log(|\mathbb{F}|)S)$.

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 $\widehat{M}$.

□

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. Here we give an overview of how to arithmetize RAM algorithms, but we include how to arithmetize multitape Turing machines in Appendix A. The multilinear extensions of many simple functions are efficient to calculate, for instance, the equality function.

Lemma 29 (Equality has Efficient Multilinear Extension)**.** *Let* $\mathrm{equ} : \{0,1\}^S \times \{0,1\}^S \to \{0,1\}$ *be the Boolean function such that* $\mathrm{equ}(a, b) = 1$ *if and only if* $a = b$.

Then for any field $\mathbb{F}$*, the multilinear extension of* equ*, denoted* $\widehat{\mathrm{equ}}$*, can be calculated in time* $\tilde{O}(\log(|\mathbb{F}|))O(S)$ *and space* $O(\log(|\mathbb{F}|) + \log(S))$.

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

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

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 1. *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 30 (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 $\tilde{O}(\log(|\mathbb{F}|))O(S)$ and space $O(\log(|\mathbb{F}|) + \log(S))$.

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 $\tilde{O}(\log(|\mathbb{F}|))S$ and space $O(\log(|\mathbb{F}|) + \log(S))$ to calculate. $\square$

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 1.

One can compute the multilinear extensions of other register, register operations in a simple register RAM model computer. 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 25) in time $\tilde{O}(\log(|\mathbb{F}|))(n+S)$ and space $O(\log(|\mathbb{F}|)\log(S+n))$. Alternatively, $\widehat{M}$ can be calculated in time $\tilde{O}(\log(|\mathbb{F}|))(n+S)^2$ and space $O(\log(|\mathbb{F}|) + \log(S+n))$.

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, bit-wise or, bit-wise 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.

  As outlined before, most 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.

  Note the multilinear extension of equality only takes $O(S)$ field operations and stores $O(1)$ field elements.

- Conditionals and Nondeterminism.

  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. Nondeterminism is similar, except that we don't need to include the condition bit.

- 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

$$\sum_{i\in\{0,1\}^{\log(n)}} \prod_{j\in[\log(n)]} (a_j i_j + (1-a_j)(1-i_j))(x_i b_1 + (1-x_i)(1-b_1)). \tag{5}$$

Where we interpret x_i as indexing into x using i as a binary number. We can rewrite Eq. (5) as

$$\begin{aligned} &\sum_{i_1\in\{0,1\}} (a_1 i_1 + (1-a_1)(1-i_1)) \\ &\quad \sum_{i_2\in\{0,1\}} (a_2 i_2 + (1-a_2)(1-i_2)) \\ &\quad \vdots \\ &\quad \sum_{i_{\log(n)}\in\{0,1\}} (a_{\log(n)} i_{\log(n)} + (1-a_{\log(n)})(1-i_{\log(n)})) \\ &\quad (x_i b_1 + (1-x_i)(1-b_1)). \end{aligned} \tag{6}$$

Following the straightforward reading of Eq. (6), you only use the first sum once, the second sum twice, etc. So on average, you only need to do a constant number of field operations per x_i. Thus this only takes $O(n)$ field operations, only storing $O(\log(n))$ field elements at a time.

At the cost of time, this can also be done more space efficiently by evaluating Eq. (5) directly, taking $O(n^2)$ field operations, but only requiring storing $O(1)$ field elements plus $O(\log(n))$ bits to store i and j.

If we also want to assert that the a register is unchanged, we just replace every instance of a_i with $(a_0)_i(a_1)_i$ and $1-a_i$ with $(1-(a_0)_i)(1-(a_1)_i)$. This works since we can expand this sum into products where for every i, either a_i or $1-a_i$ appears in every product.

- 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, one needs to calculate

$$\begin{aligned} &\sum_{i\in\{0,1\}^{\log(S)}} \left(\prod_{j\in[\log(S)]} (a_j i_j + (1-a_j)(1-i_j)) \right) \\ &\quad ((m_0)_i(m_1)_i b_1 + (1-(m_0)_i)(1-(m_1)_i)(1-b_1)) \\ &\quad\quad \prod_{k\in\{0,1\}^{\log(S)}\setminus\{i\}} ((m_0)_k(m_1)_k + (1-(m_0)_k)(1-(m_1)_k)). \end{aligned} \tag{7}$$

  First see that we can calculate

$$\prod_{k\in\{0,1\}^{\log(S)}\setminus\{i\}} ((m_0)_k(m_1)_k + (1-(m_0)_k)(1-(m_1)_k))$$

for every i efficiently by calculating each

$$\prod_{k<i}((m_0)_k(m_1)_k + (1-(m_0)_k)(1-(m_1)_k))$$

and each

$$\prod_{k>i}((m_0)_k(m_1)_k + (1-(m_0)_k)(1-(m_1)_k))$$

which only take $O(S)$ field operations, but requires storing $O(S)$ field operations. We can instead only keep track of these for the current i, and multiply or divide by the next term to update the product. This allows us to store only a constant number of field elements while still using only $O(S)$ field operations.

Then we can rewrite the sum in the same way we did for the read input case to efficiently calculate the rest of the sum. As in the input case, this only requires $O(S)$ field operations and storing $O(\log(S))$ field elements.

Alternatively, we can naively calculate Eq. (7) using $O(S^2)$ field operations, while only storing a constant number of field elements at a time, plus $O(\log(S))$ bits to hold i, j, and 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 with $O(S+n)$ field operations while storing $O(\log(S+n))$ field elements. So $\widehat{M}$ can be calculated in time $\tilde{O}(\log(|\mathbb{F}|))(S+n)$ and space $O(\log(|\mathbb{F}|)\log(S+n))$. Alternatively, we can get the better space of $O(\log(|\mathbb{F}|)+\log(S+n))$ by using time $\tilde{O}(\log(|\mathbb{F}|))(S+n)^2$. □

While this arithmetization may seem complex, and indeed, it is, we note that in the reduction of **PSPACE** to quantified Boolean formulas, the function that needs to be arithmetized is the same (in a different model of computation), plus several extra equality checks, expressed as a Boolean formula. We directly arithmetize M without reducing it to a Boolean formula first. And we get a multilinear function specifically as a result.

4.4 Number of Paths Mod a Prime

Now we give our protocol for counting the number of accepting paths in a computation. Note we also give a variant of this algorithm specialized for the small space, large p in Theorem 34.

Theorem 31 (Number of Accepting Paths Mod P)**.** *Suppose A is a nondeterministic algorithm running in space S, and time T where S and T are time $O(\log(T)S)$ computable with $S = \Omega(\log(n))$. Then there is an interactive protocol with verifier V and prover P such that, when given input x, state b, error bound $\epsilon > 0$ and prime p behaves the following:*

Completeness: *When V interacts with prover P on input x, V outputs the number of computation paths of A on input x ending at b, mod p.*

Soundness: *Given any prover P', when V interacts with prover P', V outputs an incorrect number of computation paths of A on input x ending in state b, mod p, with probability at most ϵ.*

That is, V will reject with probability $1-\epsilon$ if P' does not give the correct number of computation paths of A on input x ending in state b, mod p.

Verifier Time: *V runs in time $\tilde{O}(\log(pS/\epsilon))\,O(\log(T)S+n)$.*

Verifier Space: *V runs in space $O(\log(pS/\epsilon)S)$.*

Prover Time: *P runs in time* $\mathbf{polylog}(pS/\epsilon)2^{O(S)}$.

Proof. We start by outlining how to convert this number of computation paths to a matrix problem, then we show how to apply Lemma 28 to solve that. Let a be the canonical starting state of algorithm A.

Take T to be a power of two so that $T=2^t$. If T is not a power of 2, we can just take T to be the smallest power of two greater than the original T. Let k be the smallest integer so that $p^k > \frac{4\log(T)S}{\epsilon}$. Let $q=p^k$ and $\mathbb{F}$ be the field with q elements. Note that $|\mathbb{F}| \le p\frac{4\log(T)S}{\epsilon}$.

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 $\widehat{M'}(a',b')=M'_{a',b'}$, as in Definition 25.

Observe that $M^{2^i}_{a',b'}$ is just the number of computation paths of length 2^i from a' to b', mod p, since M is an adjacency matrix with entries in $\mathbb{F}_p$. Thus our verifier wants to output $M^T(a,b)$. Or since a and b are states (thus, expressed in binary), the verifier wants to output $\widehat{M^{2^t}}(a,b)$.

In the full protocol, the prover first provides the verifier with a candidate $\alpha \in \mathbb{F}$ with the claim that $\alpha=\widehat{M^{2^t}}(a,b)$. Now we use Lemma 28 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 28, 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 28, 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

$$\frac{4S\log(T)}{|\mathbb{F}|} \le \epsilon.$$

In particular, the probability the verifier does not reject and $\alpha_t = \widehat{M}(a_t, b_t)$ is at most ϵ. 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 ϵ.

Verifier Time: This verifier takes time $\tilde{O}(\log(|\mathbb{F}|))S\log(T)$ to run Lemma 28 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

$$\tilde{O}(\log(|\mathbb{F}|))(S\log(T)+n) = \tilde{O}(\log(pS/\epsilon))O(\log(T)S+n).$$

Verifier Space: Between subsequent applications of Lemma 28, the verifier only needs to store some i, a_i, b_i, and α_i, which only takes space $O(\log(|\mathbb{F}|)S)$. Each call to Lemma 28 also only needs space $O(\log(|\mathbb{F}|)S)$. Finally, the verifier only needs space $O(\log(|\mathbb{F}|)S)$ to calculate $\widehat{M}(a_t, b_t)$. So the overall verifier only requires space

$$O(\log(|\mathbb{F}|)S) = O(\log(pS/\epsilon)S).$$

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 24.

Finally, given oracle access to each $\widehat{M^{2^i}}$, the prover in Lemma 28 only takes time $\tilde{O}(\log(|\mathbb{F}|))2^S$. Thus of any call to Lemma 28 only takes time

$$\mathbf{poly}(\log(|\mathbb{F}|))2^{3S}.$$

Thus the prover runs in time $\mathbf{poly}\left(\log(pS/\epsilon)\right)2^{O(S)}$.

□

4.5 Protocol for TISP

As an immediate corollary of Theorem 31, since deterministic algorithms always have either one accepting computation path or zero, by using $p = 2$, and setting b to be some canonical end state, we have Theorem 1.

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.

5 Efficient IP for BPTISP

Our protocol for randomized algorithms first uses a PRG for bounded space to convert our randomized algorithm into a deterministic algorithm, then applies our deterministic protocol. Note our PRG uses $O(\log(T)S)$ random bits, so these can't be stored in the algorithms state without incurring an extra $\log(T)$ factor in the verifier run time. This is fine since our PRG only uses $O(S)$ bits of working space and our protocol works for small space.

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*

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

Proof. Suppose $L\in\mathbf{BPTISP}[T,S]$ is computed by randomized algorithm A running in space S and time T. For x of length n, we want to verify if $x\in L$ or not. We will construct a new input, x', of length $n+O(S\log(T))$ decided by deterministic algorithm A' running in time $\mathbf{poly}(T)$ and space $O(S)$ such that if $x\in L$, then with high probability $A'(x')=1$, and if $x\notin L$, then with high probability $A'(x')=0$. Then we use Theorem 1 on A' and x' to verify whether $A'(x')=1$, which with high probability is equivalent to whether $x\in L$.

As a technical detail, to maintain soundness and completeness, we will first need to amplify A to get a new protocol, A^*, by repeating it a constant number of times and taking the majority output. Similarly, we repeat the interactive protocol.

Since A is a randomized algorithm, $\Pr[A(x,U)=1_{x\in L}]\geq\frac{2}{3}$. Let A^* be the algorithm which runs A three times and outputs the majority. Algorithm A^* runs in time $O(T)$, uses space $S+O(1)$, and $\Pr[A^*(x,U)=1_{x\in L}]\geq\frac{20}{27}$.

Nisan's PRG (Theorem 22) gives a function G with seed length $l=O(\log(T)S)$ computable in space $O(S)$ and time $\mathbf{poly}(S)$ that $\frac{1}{27}$ fools A^*. That is,

$$|\mathbb{E}[A^*(x,G(U))-A^*(x,U)]|<\frac{1}{27}$$

where U is uniform random bits.

Let $A'(x,s)=A^*(x,G(s))$ and L' be the language accepted by A'. Then by a triangle inequality,

$$\begin{aligned}\Pr_s[A'(x,s)\neq 1_{x\in L}]=&\left|\mathbb{E}_s[A'(x,s)-A^*(x,U)+A^*(x,U)-1_{x\in L}]\right|\\ <&\frac{1}{27}+\frac{7}{27}\\ <&\frac{8}{27}.\end{aligned}$$

See that A' runs in time $T'=\mathbf{poly}(T)$ and space $S'=O(S)$.

In our interactive protocol, the verifier first chooses $l=O(\log(T)S)$ bits, s, for our PRG and sends them to the prover. Let $x'=(x,s)$, so our new input length is $m=n+l=O(\log(T)S+n)$. By Theorem 1, there is an interactive protocol for whether $x'\in L'$ with perfect completeness and soundness $\frac{1}{3}$ where the verifier, V', runs in time $\tilde{O}(\log(T)S+n)$ and the prover, P', runs in time $2^{O(S)}$.

Our final protocol repeats the above protocol three times, and outputs that $x\in L$ if the prover proves $x'\in L'$ three times, or outputs that $x\notin L$ if the prover proves $x'\notin L'$ three times, and rejects otherwise.

Completeness If $x\in L$, with probability $\frac{19}{27}>\frac{2}{3}$ we have $x'\in L'$. If $x'\in L'$, by completeness of our deterministic **IP**, our prover will convince our verifier $x'\in L'$. Thus the verifier outputs $x\in L$ with probability at least $\frac{2}{3}$. Similarly for $x\notin L$.

Soundness If $x\in L$, then with probability at most $\frac{8}{27}$ will we have $x'\notin L'$. If $x'\in L'$, by soundness of our deterministic **IP**, the probability any prover convinces V' that $x'\notin L'$ is at most $\frac{1}{3}$. So the probability it convinces V' that $x'\notin L'$ three times, and thus convincing V to output $x\notin L$, is at most $\frac{1}{27}$. So by a union bound, the probability V outputs that $x\notin L$ is at most $\frac{1}{3}$. Similarly for $x\notin L$.

Time The final verifier spends $O(\log(T)S)$ time choosing s, then runs V' three times, which takes time $\tilde{O}(\log(T)S + n)$. The final prover is just P', which takes time $2^{O(S)}$.

□

6 Efficient IP for NTISP

The non deterministic algorithm uses Theorem 31, but some care must be taken to choose p so that the number of accepting paths is not 0 mod p. In general, the number of accepting paths may be an adversarial number, so we choose p randomly. For instance, it could be the product of every number less than $\frac{T}{\log(T)}$. Thus this strategy may need $p = \Omega(\frac{T}{\log(T)})$.

6.1 Finding Good Primes With High Probability

First we show such a prime p can be found with high probability.

Lemma 32 (Find Non-Divisor With High Probability)**.** *There is an algorithm A taking integer W, and constant $\epsilon > 0$ running in time $\tilde{O}(\mathbf{polylog}(\frac{1}{\epsilon})\log(W)^3)$ such that for any $w \leq 2^W$ with probability at least $1 - \epsilon$, algorithm A outputs prime $p = O(W/\epsilon)$ and w mod $p \neq 0$.*

Proof. First, for any integer m, let k_m be the number of prime numbers dividing w greater than m. Then we have

$$\begin{aligned} m^{k_m} \leq& w \\ \leq& 2^W \\ k_m \leq& \frac{W}{\log(m)} \\ \leq& \frac{W}{\ln(m)}. \end{aligned}$$

By the prime number theorem, for large enough W, for any $m \geq W$, the number of primes less than m are at most $\frac{1.25m}{\ln(m)}$, and the number of primes less than $2m$ are at least $\frac{1.75m}{\ln(m)}$. Thus there are at least $\frac{0.5m}{\ln(m)}$ primes between m and $2m$. If W is too small, just hard code some large, constant prime.

Otherwise, let $m = \frac{4W}{\epsilon}$. Then the number of primes between m and $2m$ is at least $\frac{2W}{\ln(m)\epsilon}$. Recall the total number of primes larger than m dividing w is at most $\frac{W}{\ln(m)}$. Therefore, at most $\frac{\epsilon}{2}$ fraction of the primes between m and $2m$ divide w.

Then using Miller Rabin tests on randomly chosen numbers (see Theorem 20), there is an algorithm running in time

$$\tilde{O}(\mathbf{polylog}(\frac{2}{\epsilon})\log(m)^3) = \tilde{O}(\mathbf{polylog}(\frac{1}{\epsilon})\log(W)^3)$$

outputting a uniform prime p between m and $2m$ with probability at least $\frac{\epsilon}{2}$. Then by a union bound the probability it fails to output a uniform prime p or that p divides w is at most ϵ. □

A corollary is that any w has some prime number not dividing it with size $O(\log(w))$.

Corollary 33 (Log Size Non Divisors Exist)**.** *For any integer w, there exists a prime number $p = O(\log(w))$ such that p does not divide w.*

Then using this procedure, we can with high probability find an appropriate prime, p, so that if the number of accepting paths, w, is non zero, then $w \mod p \neq 0$.

6.2 Improving Paths Mod P for Small Space

Another subtle issue is that computing the multilinear extension of M takes verifier time $\tilde{O}(\log(T)n)$ when using Theorem 31. This does not achieve the parameters of Theorem 3. For instance, the parameters in Theorem 31 show that $\mathbf{NSPACE}[n^{1/3}] \subseteq \mathbf{ITIME}[\tilde{O}(n^{4/3})]$, whereas Theorem 3 claims $\mathbf{NSPACE}[n^{1/3}] \subseteq \mathbf{ITIME}[\tilde{O}(n)]$.

Fortunately, we already have an efficient **IP** for verifying deterministic, low space algorithms. And calculating the multilinear extension of M is an efficient low space deterministic algorithm. So instead of having the verifier calculate $\widehat{M}$, we ask the prover to prove the output of the algorithm of $\widehat{M}$.

Theorem 34 (Small Space Accepting Paths Mod P)**.** *Suppose A is a nondeterministic algorithm running in space S, and time T where S and T are time $O(\log(T)S)$ computable with $S = \Omega(\log(n))$. Then there is an interactive protocol with verifier V and prover P such that, when given input x, state b, error bound $\epsilon > 0$ and prime p behaves the following:*

Completeness: *When V interacts with prover P on input x, V outputs the number of computation paths of A on input x ending at b, mod p.*

Soundness: *Given any prover P', when V interacts with prover P', V outputs an incorrect number of computation paths of A on input x ending in state b, mod p, with probability at most ϵ.*

That is, V will reject with probability $1 - \epsilon$ if P' does not give the correct number of computation paths of A on input x ending in state b, mod p.

Verifier Time: *V runs in time $\tilde{O}(\log(pS/\epsilon)S(\log(T) + \log(1/\epsilon)) + \log(1/\epsilon)n)$.*

Verifier Space: *V runs in space $\tilde{O}\left(\log(pS/\epsilon)S\right)$.*

Prover Time: *P runs in time $2^{O(\log(|\mathbb{F}|)+S)}$.*

Remark 2. *Note the difference between Theorem 34 and Theorem 31 is that the verifier time no longer has a $O(\log(pS/\epsilon))$ multiplicative factor on the n term.*

Proof. We follow the proof of Theorem 31 almost exactly, except in the end, instead of actually calculating $\widehat{M}$, we run Theorem 1 to verify the output of $\widehat{M}$.

Let k be the smallest integer so that $p^k > \frac{8\log(T)S}{\epsilon}$. Let $q = p^k$ and $\mathbb{F}$ be the field with q elements. Note that $|\mathbb{F}| \leq p\frac{8\log(T)S}{\epsilon}$.

In the full protocol, the prover first provides the verifier with a candidate $\alpha \in \mathbb{F}$ with the claim that $\alpha = \widehat{M^{2^t}}(a, b)$. Now we use Lemma 28 t times to get the claim that for some $\alpha' \in \mathbb{F}$ and some $a', b' \in \mathbb{F}^S$ we have $\alpha' = \widehat{M}(a', b')$.

Instead of calculating $\widehat{M}$ using Lemma 9, we will instead use Theorem 1 to verify a calculation of $\widehat{M}$. The space efficient calculation of Lemma 9 for $\alpha' = \widehat{M}(a', b')$ takes time $T' = \tilde{O}(\log(|\mathbb{F}|))(S + n)^2$ and space $S' = O(\log(|\mathbb{F}|) + \log(S + n))$ on a length $n' =$

$O(\log(|\mathbb{F}|)S + n)$ input. Then running Theorem 1 on this algorithm gives an **IP** with perfect completeness, verifier time $\tilde{O}(\log(T')S' + n') = \tilde{O}(\log(|\mathbb{F}|)S + n)$, verifier space $\tilde{O}(S') = \tilde{O}(\log(|\mathbb{F}|) + \log(S + n))$, and prover time $2^{O(S')} = \mathbf{poly}(S + n)2^{O(\log(|\mathbb{F}|))}$. To get soundness $\frac{\epsilon}{2}$, we can repeat this protocol $O(\ln(1/\epsilon))$ times.

Completeness: Similar to Theorem 31 we have $\alpha' = \widehat{M}(a', b')$. By completeness of Theorem 1, our interactive protocol accepts.

Soundness: Similar to Theorem 31, with probability at most $\frac{\epsilon}{2}$ we have $\alpha' = \widehat{M}(a', b')$.

Suppose $\alpha' \neq \widehat{M}(a', b')$, then by soundness of Theorem 1, after running Theorem 1 $O(\ln(1/\epsilon))$ times, the verifier accepts with probability at most $\frac{\epsilon}{2}$.

So by a union bound, the verifier accepts with probability at most ϵ.

Verifier Time: The $O(\log(T))$ sum checks to reduce to the claim that $\alpha' = \widehat{M}(a', b')$ take time $\tilde{O}(\log(|\mathbb{F}|))S\log(T)$. Running Theorem 1 for $O(\ln(1/\epsilon))$ times takes time $\log(1/\epsilon)\tilde{O}((\log(|\mathbb{F}|)S + n))$. Thus the total time is

$$\begin{aligned}&=\tilde{O}(\log(|\mathbb{F}|))S\log(T) + \log(1/\epsilon)\tilde{O}((\log(|\mathbb{F}|)S + n))\\&=\tilde{O}(\log(pS/\epsilon)S(\log(T) + \log(1/\epsilon)) + \log(1/\epsilon)n).\end{aligned}$$

Verifier Space: The verifier space from the matrix sum check is $O(\log(|\mathbb{F}|)S)$. The space from Theorem 1 is space $\tilde{O}(\log(|\mathbb{F}|) + S)$, so the total space is

$$\tilde{O}(\log(|\mathbb{F}|)S) = \tilde{O}(\log(pS/\epsilon)S).$$

Prover Time: The prover for the matrix sum check takes time $\mathbf{poly}(\log(|\mathbb{F}|))2^{3S}$, as in Theorem 31. Then the prover from Theorem 1 takes time $\mathbf{poly}(S + n)2^{O(\log(|\mathbb{F}|))}$, so the total prover time is $2^{O(\log(|\mathbb{F}|)+S)}$.

$\square$

6.3 Proving IP for NTISP

Now we can prover our result for nondeterministic algorithms.

Theorem 35 (Verifier Efficient Interactive Protocol For **NTISP**)**.** *Let S, T and W be computable in time $\tilde{O}(\log(W)\log(T)S + n)$. Suppose L is recognized by a nondeterministic algorithm, A, running in time T and space S where the total number of accepting witnesses are at most 2^W. Then*

$$L \cup L^c \subseteq \mathbf{ITIME}^1[\tilde{O}(\log(W)\log(T)S + n), 2^{O(S)}]$$

where L^c is the complement of L.

Proof. We begin by describing with verifier V and honest prover P. We describe the protocol for L, the protocol for L^c is similar. Let $w = O(2^W)$ be the number of accepting paths of A on x. First P outputs whether $w = 0$ or $w \neq 0$. The protocol splits into two cases depending on what the prover claimed about w:

$w \neq 0$: Then P chooses a prime number $p = O(W)$ such that $w \mod p \neq 0$. Such a prime is guaranteed to exist by Corollary 33. Prover P gives p to the verifier V.

Then V tests if p is prime with the Miller Rabin primality test (Theorem 19) with soundness $\frac{1}{3}$ and rejects if it fails. If p passes the primality test, V performs the interactive protocol of Theorem 34 with verifier V' and honest prover P' to confirm that $w \mod p \neq 0$ with soundness $\frac{1}{3}$. Verifier V outputs $x \in L$ if V' outputs that $w \mod p \neq 0$ and rejects otherwise.

$w = 0$: Then V uses Lemma 32 to choose a prime $p = O(W/\epsilon)$ that does not divide w with probability at least $\frac{5}{6}$. If the selected p is not prime, then P proves it by sending a factorization of p. If the factorization is correct, V gives up and says $x \notin L$.

Otherwise, V performs the interactive protocol from Theorem 34 with verifier V' and honest prover P' to verify $w \mod p$ is 0 with soundness $\frac{1}{6}$. The verifier V outputs $x \notin L$ if V' outputs that $w \mod p = 0$ and rejects otherwise.

Now we prove completeness, soundness, verifier time and prover time.

Completeness First, P truthfully outputs whether $w = 0$.

$w \neq 0$: Suppose $x \in L$. Then for number accepting paths w, there exists a $p = O(W)$ such that $w \mod p \neq 0$ by Corollary 33, which P provides. By completeness of the Miller Rabin primality test Theorem 19, V confirms p is prime. By the completeness of Theorem 34, P' convinces V' that $w \mod p \neq 0$. Thus V outputs $x \in L$.

$w = 0$: Suppose $x \notin L$. If V does not find a prime p, prover P proves the candidate is not prime. Thus V outputs $x \notin L$.

If V does find prime p, by the completeness of the protocol in Theorem 34, P' convinces V' that $w \mod p = 0$. Thus V outputs $x \notin L$.

Soundness: Consider any prover $\tilde{P}$. We use two cases, depending on whether w is actually 0, or not.

$w \neq 0$: Suppose $x \in L$. Then if $\tilde{P}$ claims $w \neq 0$, then V either confirms $x \in L$, or rejects. So suppose $\tilde{P}$ claims $w = 0$.

Since $w \neq 0$, by soundness of Lemma 32, the probability V chooses a p so that $w \mod p \neq 0$ is at least $\frac{5}{6}$. If $w \mod p \neq 0$, then from the soundness of Theorem 34, the probability V' accepts that $w \mod p = 0$ is at most $\frac{1}{6}$. Thus by a union bound, V rejects with probability at least $\frac{2}{3}$.

$w = 0$: Suppose $x \notin L$. If $\tilde{P}$ claims $w = 0$, then V either confirms $x \notin L$, or rejects. So suppose $\tilde{P}$ claims $w \neq 0$.

Then for any number p provided by $\tilde{P}$, if p is not prime, by soundness of Theorem 19, V rejects with probability $\frac{2}{3}$. If p is prime, then $w \mod p = 0$, so by soundness of Theorem 34, $\tilde{P}$ can only convince V' that $w \mod p \neq 0$ with probability $\frac{1}{3}$. So V rejects with probability at least $\frac{2}{3}$.

Verifier Time: Since prime generation and testing run in time $\tilde{O}(\log(W)^3)$, and the verifier in Theorem 34 runs in time

$$\tilde{O}(\log(pS)S\log(T) + n) = \tilde{O}(\log(W)S\log(T) + n),$$

and $\log(W) \le \log(T) \le S$, the total verifier time is

$$\tilde{O}(\log(W) S \log(T) + n).$$

Prover Time: The number of accepting paths, w, can be calculated in time $2^{O(S)}$ by repeated squaring of the computation graph of A on input x. Given $w \neq 0$, the prover can find p such that $w \mod p \neq 0$ through exhaustive search in time $\mathbf{poly}(W) = 2^{O(S)}$. Similarly, factorizing a composite p by exhaustive search takes time $\mathbf{poly}(p) = \mathbf{poly}(W) = 2^{O(S)}$. Finally, the prover in Theorem 34 runs in time $2^{O(\log(W)+S)} = 2^{O(S)}$. So the prover runs in time $2^{O(S)}$.

□

One can trivially upper bound the total number of accepting paths with $W = O(T)$. This is because at each time step, only the number of computation paths can only double. So there are only at most 2^T possible computation paths. This gives us the immediate corollary of Theorem 3.

7 Open Problems

Here are some related open problems.

1. Could one remove the dependence on $\log(T)$? Is it true that, for $S = \omega(n)$,

$$\mathbf{SPACE}[S] \subseteq \mathbf{ITIME}[\tilde{O}(S)]?$$

This would prove an equivalence, up to polylogarithmic factors, between $\mathbf{SPACE}[S]$ and $\mathbf{ITIME}[S]$. Our recent work, [MC22], showed a similar equivalence between $\mathbf{NTIME}[T]$ and languages verified by PCPs 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

$$\mathbf{ITIME}[\tilde{O}(T)] \not\subseteq \mathbf{ITIME}[T]?$$

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,

$$\mathbf{ITIME}^1[\tilde{O}(T^2)] \not\subseteq \mathbf{ITIME}^1[O(T)],$$

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 **NTISP** be as efficient as those for **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 polylogarithmic 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.

Acknowledgments

Thanks to Dana Moshkovitz for feedback on this paper, and Tayvin Otti for proof reading.

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* 1st. USA: Cambridge University Press, 2009. ISBN: 0521424267.

[AS98] Sanjeev Arora and Shmuel Safra. "Probabilistic Checking of Proofs: A New Characterization of NP". In: *J. ACM* 45.1 (Jan. 1998), 70–122. ISSN: 0004-5411. DOI: 10.1145/273865.273901. URL: https://doi.org/10.1145/273865.273901.

[Aro+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. "Proof Verification and the Hardness of Approximation Problems". In: *J. ACM* 45.3 (1998), 501–555. ISSN: 0004-5411. DOI: 10.1145/278298.278306. URL: https://doi.org/10.1145/278298.278306.

[BFL90] 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.* 1990, 16–25 vol.1. DOI: 10.1109/FSCS.1990.89520.

[CT22] 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).* 2022, pp. 125–136. DOI: 10.1109/FOCS52979.2021.00021.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. "Delegating Computation: Interactive Proofs for Muggles". In: *J. ACM* 62.4 (Sept. 2015). ISSN: 0004-5411. DOI: 10.1145/2699436. URL: https://doi.org/10.1145/2699436.

[Gol+07] 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. San Diego, California, USA: Association for Computing Machinery, 2007, 440–449. ISBN: 9781595936318. DOI: 10.1145/1250790.1250855. URL: https://doi.org/10.1145/1250790.1250855.

[Gol18] Oded Goldreich. *On Doubly-Efficient Interactive Proof Systems.* 2018. URL: https://www.wisdom.weizmann.ac.il/~oded/de-ip.html.

[Imm88] Neil Immerman. “Nondeterministic Space is Closed under Complementation”. In: *SIAM Journal on Computing* 17.5 (1988), pp. 935–938. DOI: 10.1137/0217058. eprint: https://doi.org/10.1137/0217058. URL: https://doi.org/10.1137/0217058.

[Lun+90] 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.* 1990, 2–10 vol.1. DOI: 10.1109/FSCS.1990.89518.

[MC22] 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/.

[MW18] 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. Los Angeles, CA, USA: Association for Computing Machinery, 2018, 890–901. ISBN: 9781450355599. DOI: 10.1145/3188745.3188910. URL: https://doi.org/10.1145/3188745.3188910.

[Mei13] Or Meir. “IP = PSPACE Using Error-Correcting Codes”. In: *SIAM Journal on Computing* 42.1 (2013), pp. 380–403. DOI: 10.1137/110829660. eprint: https://doi.org/10.1137/110829660. URL: https://doi.org/10.1137/110829660.

[Mil75] Gary L. Miller. “Riemann’s Hypothesis and Tests for Primality”. In: *Proceedings of the Seventh Annual ACM Symposium on Theory of Computing.* STOC ’75. Albuquerque, New Mexico, USA: Association for Computing Machinery, 1975, 234–239. ISBN: 9781450374194. DOI: 10.1145/800116.803773. URL: https://doi.org/10.1145/800116.803773.

[Nis90] Noam Nisan. “Pseudorandom Generators for Space-Bounded Computations”. In: *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing.* STOC ’90. Baltimore, Maryland, USA: Association for Computing Machinery, 1990, 204–212. ISBN: 0897913612. DOI: 10.1145/100216.100242. URL: https://doi.org/10.1145/100216.100242.

[RRR16] 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. Cambridge, MA, USA: Association for Computing Machinery, 2016, 49–62. ISBN: 9781450341325. DOI: 10.1145/2897518.2897652. URL: https://doi.org/10.1145/2897518.2897652.

[RZR22] 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. Rome, Italy: Association for Computing Machinery, 2022, 1353–1363. ISBN: 9781450392648. DOI: 10.1145/3519935.3519956. URL: https://doi.org/10.1145/3519935.3519956.

[Rab80] Michael O Rabin. “Probabilistic algorithm for testing primality”. In: *Journal of Number Theory* 12.1 (1980), pp. 128–138. ISSN: 0022-314X. DOI: https://doi.org/10.1016/0022-314X(80)90084-0. URL: https://www.sciencedirect.com/science/article/pii/0022314X80900840.

[SZ99] Michael Saks and Shiyu Zhou. “$\mathrm{BP}_H\mathrm{SPACE(S)}\subseteq\mathrm{DSPACE(S^{3/2})}$”. In: *J. Comput. Syst. Sci.* 58.2 (Apr. 1999), 376–403. ISSN: 0022-0000. DOI: 10.1006/jcss.1998.1616. URL: https://doi.org/10.1006/jcss.1998.1616.

[San07] Rahul Santhanam. “Circuit Lower Bounds for Merlin-Arthur Classes”. In: *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing.* STOC ’07. San Diego, California, USA: Association for Computing Machinery, 2007, 275–283. ISBN: 9781595936318. DOI: 10.1145/1250790.1250832. URL: https://doi.org/10.1145/1250790.1250832.

[Sav70] Walter J. Savitch. “Relationships between Nondeterministic and Deterministic Tape Complexities”. In: *J. Comput. Syst. Sci.* 4.2 (Apr. 1970), 177–192. ISSN: 0022-0000. DOI: 10.1016/S0022-0000(70)80006-X. URL: https://doi.org/10.1016/S0022-0000(70)80006-X.

[Sha92] Adi Shamir. “IP = PSPACE”. In: *J. ACM* 39.4 (Oct. 1992), 869–877. ISSN: 0004-5411. DOI: 10.1145/146585.146609. URL: https://doi.org/10.1145/146585.146609.

[She92] A. Shen. “IP = SPACE: Simplified Proof”. In: *J. ACM* 39.4 (1992), 878–880. ISSN: 0004-5411. DOI: 10.1145/146585.146613. URL: https://doi.org/10.1145/146585.146613.

[Sze88] Róbert Szelepcsényi. “The method of forced enumeration for nondeterministic automata”. In: *Acta Informatica* 26 (1988), 279–284. DOI: 10.1007/BF00299636. URL: https://doi.org/10.1007/BF00299636.

[Tha13] Justin Thaler. “Time-Optimal Interactive Proofs for Circuit Evaluation”. In: *Advances in Cryptology – CRYPTO 2013*. Ed. by Ran Canetti and Juan A. Garay. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 71–89.

A Efficient Arithmetization of Multitape Turing Machine

For completeness, we also present how to compute the arithmetization of a Turing Machine algorithm. Recall that a Turing machine is defined by a alphabet Σ, a set of states Q, a start state $a' \in Q$, an accept state $b' \in Q$, a state transition function δ. For convenience, we assume we have a two tape Turing machine with a read only input tape, and a working tape. It will be clear from the proof how to extend it to more tapes. Let $D = \{-1, 0, 1\}$ be the directions a head in a Turing Machine can move. Then δ is a partial function $\delta : Q \times \Sigma \times \Sigma \to Q \times D \times D \times \Sigma$ where $\delta(q, x, y) = (q', d_1, d_2, y')$ means when the Turing Machine is in state q with x under the input head and y under the working head, change the state to q', write y' into the working tape, then move the input head d_1 and the working head d_2. The Turing Machine rejects if the current configuration ever has an undefined transition.

We will assume symbols in Σ and states in Q are given by some binary encoding so that our multilinear extension is properly defined. Further we assume that the input head of the Turing Machine is only ever at a position between 0 and n, similarly the work head is never at a negative position.

Now recall that we can alternatively view δ as some set $\Lambda = \delta \subseteq (Q \times \Sigma \times \Sigma) \times (Q \times D \times D \times \Sigma)$. Now the idea is to write the adjacency matrix of the computation graph as the sum of the adjacency matrices for each individual $\lambda \in \Lambda$. Then the multilinear extension of the adjacency matrix is just the sum of these adjacency matrices, each of which can be computed efficiently.

For notation, I will use the term "total state" to refer to the tuple of the Turing Machines current state, q, the position of the input tape head, h, the position of the working tape head, w, and the contents of the working tape, m.

Lemma 36 (Turing Machine Arithmetization)**.** *Let A be a multitape Turing Machine described by alphabet $\Sigma = \{0,1\}$, states Q, start state $a' \in Q$, accept state $b' \in Q$, and state transitions $\Lambda \subseteq (Q \times \Sigma \times \Sigma) \times (Q \times D \times D \times \Sigma)$.*

Suppose A uses space S 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 total state a, then b is a valid total state transition, and $M_{a,b} = 0$ otherwise.

Then we can compute the multilinear extension of M ($\widehat{M}$ in Definition 25) in time $\tilde{O}(\log(|\mathbb{F}|))(n+S)$ and space $O(\log(|\mathbb{F}|)\log(S+n))$. Alternatively, $\widehat{M}$ can be calculated in time $\tilde{O}(\log(|\mathbb{F}|))(n+S)^2$ and space $O(\log(|\mathbb{F}|) + \log(S+n))$.

Proof. I will rewrite $M_{a,b}$ as a sum over the transitions in Λ of that transition being correct. This transition being correct will be the product of the state being updated correctly, the input tape being moved correctly, and the work tape being updated correctly. Each of these will use disjoint variables, so the multilinear extension of this product is just the product of their multilinear extensions.

Let $a = (q_0, h_0, w_0, m_0)$ be our total start state, $b = (q_1, h_1, w_1, m_1)$ be our total end state, and $\lambda = (q_0', x_0', y_0') \times (q_1', d_1, d_2, y_1')$ be our state transition function. Let equ be the function that takes 2 arguments and returns one if and only if they are equal. Similarly, let $\widehat{\text{equ}}$ be its multilinear extension. Note that from Lemma 29, $\widehat{\text{equ}}$ can be computed efficiently.

Then we can define the indicator of the input head being moved according to λ as

$$\text{Input}_\lambda(h_0, h_1) = \sum_{i=0}^{n} \text{equ}(x_i, x_0')\text{equ}(i, h_0)\text{equ}(i + d_1, h_1).$$

The sum is just the sum over all potential input head positions of the head being at the position and the input in that head position being the correct input, then moving correctly. Note that h_0 and h_1 are the only variables of Input_λ, and that h_0 only appears in one equ, and h_1 in the other. Thus the multilinear extension of Input_λ is just

$$\widehat{\text{Input}_\lambda}(h_0, h_1) = \sum_{i=0}^{n} \text{equ}(x_i, x_0')\widehat{\text{equ}}(i, h_0)\widehat{\text{equ}}(i + d_1, h_1).$$

Now we show that $\widehat{\text{Input}_\lambda}$ can be calculated quickly, and with small space.

The idea is to iteratively calculate each of these equ terms for one i, using the result from the last i. For $\widehat{\text{equ}}(i, h_0)$, we store $\log(S)$ partial products, for $j \in [\log(n)]$, of the form $\widehat{\text{equ}}(i_{[j]}, (h_0)_{[j]})$, assuming the least significant bits are last. Then in expectation, we can calculate these for $i+1$ while only changing a constant number of these partial products (since $i+1$ in expectation only differs from i on a constant number of bits), similar to Lemma 9. This only takes $O(n)$ field operations and only requires storing $O(\log(n))$ field elements. Similarly for $\widehat{\text{equ}}(i + d_1, h_1)$.

Similarly, we can define the indicator of the workspace being updated correctly

$$\begin{aligned}\text{Work}_\lambda(w_0, m_0, w_1, m_1) = \sum_{i=0}^{S} & \text{equ}(i, w_0)\text{equ}(i + d_2, w_1) \\ & \text{equ}(y'_0, (m_0)_i)\text{equ}(y'_1, (m_1)_i) \\ & \text{equ}((m_0)_{[i-1]}, (m_1)_{[i-1]}) \\ & \text{equ}((m_0)_{[S]\setminus[i]}, (m_1)_{[S]\setminus[i]}).\end{aligned}$$

See that $\text{equ}(i, w_0)\text{equ}(i+d_2, w_1)$ confirms the work head is moved correctly, $\text{equ}(y'_0, (m_0)_i)\text{equ}(y'_1, (m_1)_i)$ confirms the cell under the work head is updated correctly, and finally $\text{equ}((m_0)_{[i-1]}, (m_1)_{[i-1]})$ and $\text{equ}((m_0)_{[S]\setminus[i]}, (m_1)_{[S]\setminus[i]})$ confirm the rest of the memory is unchanged.

Again, each of the variables only appear in one of the equ in each term of the sum. Thus:

$$\begin{aligned}\widehat{\text{Work}_\lambda}(w_0, m_0, w_1, m_1) = \sum_{i=0}^{S} & \widehat{\text{equ}}(i, w_0)\widehat{\text{equ}}(i + d_2, w_1) \\ & \widehat{\text{equ}}(y'_0, (m_0)_i)\widehat{\text{equ}}(y'_1, (m_1)_i) \\ & \widehat{\text{equ}}((m_0)_{[i-1]}, (m_1)_{[i-1]}) \\ & \widehat{\text{equ}}((m_0)_{[S]\setminus[i]}, (m_1)_{[S]\setminus[i]}).\end{aligned}$$

Now we show that $\widehat{\text{Work}_\lambda}$ can be calculated quickly, and with small space. This can be done similarly to Lemma 9.

The idea is to iteratively calculate each of these equ terms for one i, using the result from the last i. For $\widehat{\text{equ}}(i, w_0)$, we store $\log(S)$ partial products of the form, $\widehat{\text{equ}}(i_{[j]}, (w_0)_{[j]})$, assuming the least significant bits are last. Then in expectation, we can calculate these for $i + 1$ while only changing a constant number of these partial products (since $i + 1$ in expectation only differs from i on a constant number of bits). This only takes $O(S)$ field operations and only requires storing $O(\log(S))$ field elements. Similarly for $\widehat{\text{equ}}(i + d_2, w_1)$.

Since $\widehat{\text{equ}}(y'_0, (m_0)_i)$ and $\widehat{\text{equ}}(y'_0, (m_0)_i)$ only involve a constant number of field elements, they can be calculated directly for each i efficiently.

See that $\widehat{\text{equ}}((m_0)_{[i]}, (m_1)_{[i]})$ can be efficiently be calculated from $\widehat{\text{equ}}((m_0)_{[i-1]}, (m_1)_{[i-1]})$ by

$$\widehat{\text{equ}}((m_0)_{[i]}, (m_1)_{[i]}) = \widehat{\text{equ}}((m_0)_{[i-1]}, (m_1)_{[i-1]})\widehat{\text{equ}}((m_0)_i, (m_1)_i).$$

Finally, see that $\widehat{\text{equ}}((m_0)_{[S]\setminus[i+1]}, (m_1)_{[S]\setminus[i+1]})$ can be calculated from $\widehat{\text{equ}}((m_0)_{[S]\setminus[i]}, (m_1)_{[S]\setminus[i]})$ by

$$\widehat{\text{equ}}((m_0)_{[S]\setminus[i+1]}, (m_1)_{[S]\setminus[i+1]}) = \frac{\widehat{\text{equ}}((m_0)_{[S]\setminus[i]}, (m_1)_{[S]\setminus[i]})}{\widehat{\text{equ}}((m_0)_{i+1}, (m_1)_{i+1})}.$$

There is an important edge case we have to handle here: what if $\widehat{\text{equ}}((m_0)_{i+1}, (m_1)_{i+1}) = 0$? Then we cannot perform this division. Fortunately, in this case, all the terms in the sum where $w_0 \neq i{+}1$ are 0, because either $\widehat{\text{equ}}((m_0)_{[i-1]}, (m_1)_{[i-1]}) = 0$, or $\widehat{\text{equ}}((m_0)_{[S]\setminus[i]}, (m_1)_{[S]\setminus[i]}) = 0$. Thus in this case we only need to calculate a single term in the sum, which can be done efficiently.

Finally, see that

$$\begin{aligned}M_{a,b} = \sum_{\lambda\in\Lambda} &\mathrm{equ}(q_0, q_0')\mathrm{equ}(q_1, q_1')\\ &\mathrm{Work}_\lambda(w_0, m_0, w_1, m_1)\\ &\mathrm{Input}_\lambda(h_0, h_1).\end{aligned}$$

where q_0' and q_1' are from λ, and the rest of the variables are from a and b as described before. See the equation is 1 if and only if there is some state transition where the Turing machine state is updated correctly, the input head is updated correctly and points to the correct symbol, the work head is updated correctly and points to the correct symbol, and the work tape is updated correctly.

Since each term in the sum is a product of functions with disjoint variables, the multilinear extension of the sum is just a multilinear extension of those functions, so

$$\begin{aligned}\widehat{M}(a, b) = \sum_{\lambda\in\Lambda} &\widehat{\mathrm{equ}}(q_0, q_0')\widehat{\mathrm{equ}}(q_1, q_1')\\ &\widehat{\mathrm{Work}}_\lambda(w_0, m_0, w_1, m_1)\\ &\widehat{\mathrm{Input}}_\lambda(h_0, h_1).\end{aligned}$$

Then since there are only constantly many transitions in Λ, we have $\widehat{M}$ can be computed with only $O(S + n)$ field operations while storing only $O(\log(S + n))$ field elements, which only takes time $\tilde{O}(\log(|\mathbb{F}|))(S + n)$ and space $O(\log(|\mathbb{F}|) \log(S + n))$.

Alternatively, a naive calculation of $\widehat{\mathrm{Work}}_\lambda$ only stores a constant number of field elements at a time, and only uses $O(S^2)$ field operations. Similarly a naive calculation of $\widehat{\mathrm{Input}}_\lambda$ only stores a constant number of field elements at a time, and only uses $O(n^2)$ field operations. This gives a more space efficient, but less time efficient calculation of $\widehat{M}$ running in time $\tilde{O}(\log(|\mathbb{F}|))(S + n)^2$ and space $O(\log(|\mathbb{F}|) + \log(S + n))$. □