Tighter **MA**/1 Circuit Lower Bounds From Verifier Efficient PCPs for PSPACE

Joshua Cook* Dana Moshkovitz†

September 9, 2025

Abstract

We prove that for some constant $a > 1$, for all $k \leq a$,

$$\mathbf{MATIME}[n^{k+o(1)}]/1 \not\subset \mathbf{SIZE}[O(n^k)],$$

for some specific $o(1)$ function. This is a super linear polynomial circuit lower bound.

Previously, Santhanam [San07] showed that there exists a constant $c > 1$ such that for all $k > 1$:

$$\mathbf{MATIME}[n^{ck}]/1 \not\subset \mathbf{SIZE}[O(n^k)].$$

Inherently to Santhanam's proof, c is a large constant and there is no upper bound on c. Using ideas from Murray and Williams [MW18], one can show for all $k > 1$:

$$\mathbf{MATIME}[n^{10k^2}]/1 \not\subset \mathbf{SIZE}[O(n^k)].$$

To prove this result, we construct the first **PCP** for **SPACE**$[n]$ with quasi-linear verifier time: our **PCP** has a $\tilde{O}(n)$ time verifier, $\tilde{O}(n)$ space prover, $O(\log(n))$ queries, and polynomial alphabet size. Prior to this work, **PCP**s for **SPACE**$[O(n)]$ had verifiers that run in $\Omega(n^2)$ time. This **PCP** also proves that **NE** has **MIP** verifiers which run in time $\tilde{O}(n)$.

*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.

†danama@cs.utexas.edu. Department of Computer Science, UT Austin. This material is based upon work supported by the National Science Foundation under grant numbers 1705028, 2200956, and 2312573.

ISSN 1433-8092

Contents

1 Introduction

Some of the most fundamental problems in complexity theory are proving circuit lower bounds for uniform complexity classes. One such conjecture is that **NP** does not have polynomial size circuits, which is a strong version of $\mathbf{P} \neq \mathbf{NP}$. Very little is known on such lower bounds. In particular, there are no known proofs that **NEXP** does not have polynomial sized circuits! However, there are some closely related results that could be loosely seen as relaxations.

One can strengthen **NP** slightly by giving the non-deterministic algorithm access to randomness, as well as an extra bit of trusted advice. This gives the complexity class $\mathbf{MA}/1$. We can weaken polynomial sized circuits to circuits of fixed polynomial size: $\mathbf{SIZE}[n^k]$ for constant k.

Santhanam [San07] proved that for any constant k, $\mathbf{MA}/1 \nsubseteq \mathbf{SIZE}[n^k]$. The $\mathbf{MA}/1$ algorithm runs in time n^{ck} for a large $c > 1$. In fact, inherently to Santhanam's proof, there is no upper bound on c (We will explain why when we describe Santhanam's proof in Section 1.2.1). One can use ideas from Murray and Williams [MW18] to get, for some explicit c with $2 < c < 10$, the result $\mathbf{MATIME}[n^{ck^2}]/1 \not\subset \mathbf{SIZE}[n^k]$.

The goal of this paper is to prove a fine grained separation of $\mathbf{MA}/1$ from fixed polynomial size circuits, namely,

$$\mathbf{MATIME}[n^{k+o(1)}]/1 \not\subset \mathbf{SIZE}[n^k].$$

We believe that the gold standard for separations should be fine grained separations. Fine grained separations are necessary for key results in complexity theory, e.g., Williams' program (See, e.g., [Wil11]) and optimal derandomization [Dor+20].

Some fine grained separations are known, namely, hierarchy theorems that show that giving algorithms more time allows them to solve more problems [HS65; Coo72]. Hierarchy theorems are known for many complexity classes. While no hierarchy theorems are known for **MA**, they are known for $\mathbf{MA}/1$. Fortnow, Santhanam, and Trevisan showed that **MA** with a small amount of advice can solve more problems when given more time [FST05]. Van Melkebeek and Pervyshev showed that for any $1 < b < d$, $\mathbf{MATIME}[n^b]/1 \subsetneq \mathbf{MATIME}[n^d]/1$ [MP06]. In particular, they imply that even $\mathbf{MATIME}[n^{2k}]/1$ is much larger than $\mathbf{MATIME}[n^{k+o(1)}]/1$.

1.1 Results

In this work, we give a fine grained separation for $\mathbf{MA}/1$ and $\mathbf{SIZE}[n^k]$. We show that for at least some $k > 1$, there is an **MA** protocol with one bit of advice whose verifier has time almost n^k such that any circuit solving the same problem also requires size n^k. Formally:

Theorem 1.1.1 (Fine Grained **MA** Lower Bound)**.** *There exists a constant $a > 1$, such that for all $k < a$, for some $f(n) = o(1)$,*

$$\mathbf{MATIME}[O(n^{k+f(n)})]/1 \not\subset \mathbf{SIZE}[O(n^k)].$$

We stress that we give **super linear** polynomial lower bounds. Our result holds for some a *strictly* greater than 1, even though we don't know which a. This result removes the large polynomial factor in the gap between the $\mathbf{MA}/1$ time and the circuit size in Santhanam's result. It may be the case that a is small, like $a = 1.0001$. But in that case, we get the following result for all k:

Theorem 1.1.2 (**MA** Lower Bound for Small a)**.** *If the a from Theorem 1.1.1 is finite, then for all $k > 0$, for some $f(n) = o(1)$,*

$$\mathbf{MATIME}[O(n^{ak+f(n)})]/1 \not\subset \mathbf{SIZE}[O(n^k)].$$

This gives us a win-win scenario: if a is large, we get a strong result for a large range of k, but if a is small we get a similar result for all k.

When we describe our proof we will explain why we only get separations for $k < a$ for an (unknown) $a > 1$ and not for all $k > 1$. For now we would like to stress that: (1) under plausible complexity assumptions the upper bound a is in fact super-constant in n; (2) even the case of a constant $a > 1$ as promised in our theorem is highly interesting, since it is unknown how to prove that $\mathbf{NP} \nsubseteq \mathbf{SIZE}[n^k]$ for *any* $k > 1$.

Santhanam's original proof uses an interactive protocol for **PSPACE**. To prove our circuit lower bound, we replace the interactive protocol with a new, more efficient **PCP**. To get our fine grained results, we need a **PCP** for space $S = O(n)$ and time $T = 2^{O(n)}$ algorithms, where the verifier simultaneously has $\tilde{O}(n)$ time and $\mathsf{poly}(\log(n))$ many queries. To be consistent with prior work, we will refer to the time of the verifier as the "decision complexity" of the PCP. Further, the **PCP** needs a prover that can compute the proof in $\tilde{O}(n)$ space. Notably, we do not need any bounds on the proof length or randomness complexity.

The **PCP** given by Babai, Fortnow, and Lund in their proof that **MIP** = **NEXP** [BFL90] required $\Omega(\log(T))$ queries, while we want $O(\log(\log(T)))$ queries.

Holmgren and Rothblum in their work on delegated computation [HR18] improved on the BFL **PCP** in several ways that can[1] be used to give a **PCP** with decision complexity $\tilde{O}(n + \log(T))$. Unfortunately, it still requires $\Omega(\log(T))$ queries.

Ben-Sasson, Goldreich, Harsha, Sudan, and Vadhan [Ben+05] gave a **PCP** that uses a constant number of queries, but has decision complexity $\mathsf{poly}(\log(T))$, while we need $\tilde{O}(n+\log(T))$ decision complexity. Similar results were given by subsequent work [Mei09; BV14; Ben+13].

The small space requirement for the prover is achieved by Holmgren and Rothblum [HR18]. In some **PCP**s, like the **PCP** in Ben-Sasson, Chiesa, Genkin, and Tromer's work on the concrete efficiency of **PCP**s [Ben+13], the prover requires space $\Omega(T)$. In contrast, our result needs prover space $\tilde{O}(S + n)$.

A sufficiently efficient **PCP** was not known, so we construct a new **PCP**.

Theorem 1.1.3 (Verifier Efficient **PCP**)**.** *Let $S, T = \Omega(n)$ be functions, and L be any language computed by a simultaneous time T and space S algorithm. Let $\delta \in (0, 1/2)$ be a constant. Then there is a* **PCP** *for L with:*

1. *Decision complexity*[2] $\tilde{O}(n + \log(T))$.
2. *Query time*[3] $\tilde{O}(\log(T))$.
3. $O(\log(n) + \mathsf{polylog}(\log(T)))$ *queries.*
4. *Alphabet Σ with* $\log(|\Sigma|) = O(\log(\log(T)))$.
5. *Randomness* $O(\log(T)\log(n\log(T)))$.
6. *Prover space* $\tilde{O}(S)$.
7. *Perfect completeness and soundness δ.*

We believe we can achieve a similar decision complexity, query time and prover space while also achieving constant number of queries and alphabet size. We do not need these improvements for our main result, so we only prove this simpler result.

Only our prover requires the space bound for its efficient computation. If we remove this space limitation, we get a similar **PCP** for nondeterministic algorithms.

Theorem 1.1.4 (Verifier Efficient **PCP** for Nondeterministic Algorithms)**.** *Let $T = \Omega(n)$, $\delta \in (0, 1/2)$ be a constant, and $L \in$* **NTIME**$[T]$*. Then there is a* **PCP** *for L with:*

1. *Decision complexity* $\tilde{O}(n + \log(T))$.
2. *Query time* $\tilde{O}(\log(T))$.
3. $O(\log(n) + \mathsf{polylog}(\log(T)))$ *queries.*
4. *Alphabet Σ with* $\log(|\Sigma|) = O(\log(\log(T)))$.
5. *Randomness* $O(\log(T)\log(n\log(T)))$.

[1] The **PCP** constructed by Holmgren and Rothblum was built to have no signalling soundness and has many steps that take longer than $\tilde{O}(\log(T))$ time to compute. Still, the basic elements of of their **PCP** needed for a standard **PCP** are computable in $\tilde{O}(n + \log(T))$ time.

[2] We note that a prior version of this paper called decision complexity verifier time, and achieved decision complexity $\tilde{O}(n + \log(T))$. This prior result has a slightly longer and less general proof than the **PCP** included in this result.

[3] In prior works, query time was referred to as verifier time. We avoid calling the query time the verifier time because the time used by a verifier in a PCP system is related to both the query time and the decision complexity.

6. Perfect completeness and soundness δ.

An immediate corollary of Theorem 1.1.4 is a more fine grained equivalence between **MIP** and **NEXP**.

Corollary 1.1.5 (Fine Grained Equivalence of **MIP** = **NEXP**)**.** *For any time constructible function $p(n) = \Omega(n)$, language $L \in$* **NTIME**$[2^{\tilde{O}(p(n))}]$ *if and only if there is a two prover, one round* **MIP** *protocol for L whose verifier runs in time $\tilde{O}(p(n))$.*

Note this equivalence implies a hierarchy theorem for **MIP** since there are hierarchy theorems for **NTIME** [Coo72; SFM78; Žá83; FS11].

A special case is **MIP** protocols for **NE**.

Corollary 1.1.6 (**NE** Has Quasi-linear Time Verifiers)**.** *For any language $L \in$* **NE***, there is a two prover, one round* **MIP** *protocol for L whose verifier runs in time $\tilde{O}(n)$.*

Note this decision complexity is nearly optimal since the verifier requires linear time to read its entire input.

All previous **PCP**s fail to achieve such an efficient **MIP** verifier. If the original **PCP** makes $\Omega(n)$ queries of size $\Omega(n)$, then it takes $\Omega(n^2)$ time to send the queries even if we allow more provers. And all previous **PCP**s with fewer queries require decision complexity $\Omega(n^2)$ to either verify the response or compute the queries.

1.2 Proof Idea

1.2.1 MA Lower Bounds Using PCP

We first review Santhanam's original proof for any fixed k, that **MA** $\not\subset$ **SIZE**$[n^k]$.

Santhanam's original result uses the fact that if **PSPACE** $\subset$ **P/poly**, then **PSPACE** = **MA**. This follows from the famous result that **IP** = **PSPACE** [Sha92; Lun+92]. The idea is that if **PSPACE** $\subset$ **P/poly**, then an **MA** protocol can guess a circuit deciding any problem in **PSPACE**. The prover in the interactive protocol for **PSPACE** is also computable in **PSPACE**. So to solve any **PSPACE** problem in **MA**, the **MA** protocol first guesses the circuit for a prover, then simulates the verifier using the circuit we guessed as the prover.

Using this, Santhanam's original proof then considered two cases: either **PSPACE** $\subset$ **P/poly**, or **PSPACE** $\not\subset$ **P/poly**.

If **PSPACE** $\subset$ **P/poly**, then we already know **PSPACE** = **MA**. Now we just need to show **PSPACE** $\not\subset$ **SIZE**$[n^k]$. There is a straightforward, space efficient algorithm that exhaustively finds a circuit of size larger than n^k that computes a function that cannot be computed by a smaller circuit. In fact, such an algorithm only requires space $\tilde{O}(n^k)$. So **PSPACE** $\not\subset$ **SIZE**$[n^k]$. In this case, **PSPACE** = **MA**, so **MA** $\not\subset$ **SIZE**$[n^k]$.

If **PSPACE** $\not\subset$ **P/poly**, then we know a hard problem that is not in **SIZE**$[n^k]$, namely any **PSPACE** complete problem. Let us take a **PSPACE** complete, randomly downward self reducible language, Y. Now Y may be too hard for **MA** to solve, but if we give it enough padding, eventually the padded version of Y will be computable by size n^k circuits. But for this amount of padding, **MA** can pull the same trick it does in the **PSPACE** $\subset$ **P/poly** case. Namely, guess a circuit for Y and then simulate the **IP** protocol for Y. For some **PSPACE** complete Y, the language itself is its proof and this works. The trick is to use just the right amount of padding so it requires circuits of at least size n^k, but not much larger. Santhanam uses the single bit of advice in a clever way to figure out when there is just the right amount of padding.

In either case, the time of this protocol is roughly the time of the verifier in the **IP** protocol, plus the size of the prover circuit times the number of times the prover is queried.

There are two reasons the **MA** protocol could take polynomially more time than the size of the circuits it wants to compute in the case **PSPACE** $\subset$ **P/poly**. One is that the **IP** from the original Santhanam result has polynomial decision complexity and a polynomial time interaction with the prover, making the verifier in the **MA**/1 protocol take polynomially longer than the circuit complexity of the problem being solved. By using a **PCP**, we get better results. The other is that the prover circuit complexity could be large, depending on the circuit size required for **PSPACE** (could be any polynomial when **PSPACE** $\subset$ **P/poly**). This is the reason there is no upper bound on the polynomial run time of the **MA**/1 protocol in Santhanam's proof. To avoid this issue we consider a finer case analysis.

We break the problem into three cases. For some $\mathbf{SPACE}[O(n)]$ complete language, X, we have one[4] of the following:

1. $X \notin \mathbf{P/poly}$.
2. $X \in \mathbf{SIZE}[n^{1+o(1)}]$.
3. $X \in \mathbf{SIZE}[n^{a+o(1)}] \setminus \mathbf{SIZE}[n^{a-o(1)}]$ for some $a > 1$.

The original proof only used the two cases $X \notin \mathbf{P/poly}$ and $X \in \mathbf{P/poly}$. The case where $X \notin \mathbf{P/poly}$ is completely unchanged. Note that this is the plausible case, and here there is no constant upper bound a on k.

If $X \in \mathbf{P/poly}$, we use our efficient **PCP**, Theorem 1.1.3, instead of the **IP** Santhanam uses. With this substitution, the case where $X \in \mathbf{SIZE}[n^{1+o(1)}]$ is almost unchanged from the original proof. By separating this into its own case, we get tight bounds for all k in this case.

If $X \in \mathbf{SIZE}[n^{a+o(1)}] \setminus \mathbf{SIZE}[n^{a-o(1)}]$ for some $a > 1$, then we use the same padding technique we use if $X \notin \mathbf{P/poly}$, just using our new **PCP**. In this case, we can only do this if for some $k < a$, we are trying to show $\mathbf{MATIME}[n^{k+o(1)}]/1 \not\subset \mathbf{SIZE}[n^{k-o(1)}]$. This is the case where a is finite, but in this case, we can use Santhanam's argument using our **PCP** to get Theorem 1.1.2.

To see why $k > a$ poses a difficulty, suppose $\mathbf{SPACE}[O(n)] \not\subseteq \mathbf{SIZE}[o(n^2)]$, but $\mathbf{SPACE}[O(n^2)] \subseteq \mathbf{SIZE}[O(n^2)]$. Then to get a language requiring size n^3 circuits, we need to use a space n^3 algorithm. But the prover for a space n^3 language is a language running on an input with length n^3, and using space linear in its input length. Thus we may need a size $(n^3)^2 = n^6$ circuit for our prover. So the verifier takes time at least n^6 to even read the prover circuit, thus can't run in time n^3. See Item 2 in our open problems for further explanation.

1.2.2 Verifier Efficient PCP

Now we explain the **PCP** we actually use in the **MA** protocol. We start with a **PCP** similar to [HR18] and [BFL90] that we refer to as our base **PCP**. This **PCP** has a verifier that runs in time $\tilde{O}(n + \log(T))$ and uses $O(\log(T))$ queries. To reduce the number of queries, we use **PCP** composition [AS98; Ben+04; DR04; MR08; DH09].

To perform **PCP** composition, we need a robust **PCP**. Loosely, a robust **PCP** is a **PCP** so that when $x \notin L$, for any proof, most sets of queries to that proof return not only a rejected response, but a response that is far from any accepted response. To make our base **PCP** robust, we use the aggregation through curves technique [Aro+98]. Now we briefly explain how to use aggregation through curves to convert our base **PCP** into a robust **PCP**.

An honest proof for our base **PCP** is a single low degree polynomial in several variables. We call the number of variables the dimension of the **PCP**. Suppose our base **PCP** has q queries. To make our **PCP** robust, we first choose the randomness for the base **PCP**, and another random point in the **PCP** proof. Then we find the degree q curve that goes through all these points. Then we check if the proof, restricted to this curve, is a low degree polynomial, and whether the base **PCP** would have accepted on this input. Since a low degree polynomial is an error correcting code, this gives robustness.

One concern one might have with this robust **PCP** is that it actually requires $\Omega(\log(T)^2)$ queries. Since our verifier needs to run in $\tilde{O}(n + \log(T))$ time, the verifier cannot calculate all of these query locations. However, since we do **PCP** composition, we only need to calculate the query locations actually queried by the inner **PCP** of proximity, which are far fewer locations. Thus we only need to calculate any individual query location quickly. To find these query locations requires us to compute a point on the degree q curve going through each of our q points our base **PCP** queries plus a random point. In our base **PCP**, $q = O(\log(T))$ and our proof, when viewed as a polynomial, has dimension $O(\log(T))$. So the naive way to compute this curve is to calculate each coordinate independently, which would take time $\tilde{O}(\log(T)^2)$.

To efficiently compute low degree curves through points, or to extrapolate a function going through those points, we introduce the concept of time extrapolatable functions.

[4]This is a trichotomy in an asymptotic sense: for every constant a, either $X \in \mathbf{SIZE}[O(n^a)]$ or it is not. See Section 3.5 for details.

Definition 1.2.1 (Extrapolatability)**.** *For any $n, q, t > 0$, and field $\mathbb{F}$, we call $Q : [q] \to \mathbb{F}^n$ "t extrapolatable" (or time t extrapolatable) if there is a time t algorithm taking any $v \in \mathbb{F}^q$, that outputs*

$$\sum_{i \in [q]} v_i Q(i).$$

Equivalently, if we think of Q as outputting the columns of a matrix, then we say Q is time t extrapolatable if one can multiply a vector with it in time t. An important property of extrapolatable functions is that an extrapolation of an extrapolatable function can be computed efficiently. This is where it gets its name.

Our base **PCP** is just a sum check and a few point checks. Each of these are time $\tilde{O}(\log(T))$ extrapolatable. Our robust **PCP** only queries locations easily computable given the extrapolation of our base **PCP** query locations. Extrapolations of extrapolatable functions are easy to compute, so we can easily compute the query locations of the robust **PCP**.

We also introduce the concept an extrapolatable **PCP** (**ePCP**) as one where an honest proof is a low degree polynomial, and the query locations after fixing a choice of randomness are extrapolatable. We show that any **ePCP** can be extended into a robust **PCP** where the query locations of that robust **PCP** can be computed efficiently.

1.3 Generalization And Sharpness

We actually prove a stronger result than Theorem 1.1.1 that is sharp. First, our **MA** protocol is input oblivious: the message from Merlin is just a program for deciding a **PSPACE** complete language and doesn't depend on the specific input, just its length. Second, the hardness is against the model used in Merlin's message. We described our results so far with deterministic circuits, but Merlin can also give a randomized algorithm. By randomized algorithm, we can use probabilistic circuits or Turing machines with a randomness tape, since they are equivalent up to a polylog factor in the time/size [Fis74; Pip77; LW12; HS66; PF79]. The same result holds for standard RAM models of computation as well.

We define input oblivious Merlin-Arthur time, **OMATIME**, the same way as Fortnow, Santhanam, and Williams [FSW09]. Input oblivious Merlin-Arthur are languages solvable with untrusted advice, where the advice only depends on the input length. In our case, Merlin gets to send a long, untrusted message for every input length, and Arthur also gets a single bit of trusted advice. See Definition 2.1.4. Note that Santhanam's original proof implicitly also uses input oblivious **MA**.

The main property of circuits we use is that a randomized algorithm can efficiently simulate them. We can instead use **BPTIME**$[n^k]/n^k$, that is, randomized algorithms running in time n^k with description length n^k. This uses the same model of computation as our verifier, allowing it to more efficiently simulate **OMATIME**.

Using **OMATIME** instead of **MATIME** and **BPTIME** instead of **SIZE**, we can follow the same proof as our main result to show:

Theorem 1.3.1 (**OMATIME** Lower Bound Against **BPTIME**)**.** *There exists constant $a > 1$, such that for all $k < a$, for some $f(n) = o(1)$,*

$$\mathbf{OMATIME}[O(n^{k+f(n)})]/1 \not\subset \mathbf{BPTIME}[O(n^k)]/O(n^k).$$

This result is tight in the sense that for any function $f(n)$, we have

$$\mathbf{OMATIME}[f(n)]/1 \subseteq \mathbf{BPTIME}[O(f(n))]/(f(n)+1).$$

Thus only the constant a, the function f, and the one bit of advice in Theorem 1.3.1 can be improved. If one wants to show that **MATIME**$[n^a]/1 \not\subset$ **SIZE**$[n^b]$ for constants $a < b$, we need a proof technique that does not extend to **OMA** and **BPTIME**. That is, we provably need to use nondeterminism that depends on the input. Thus, our result is less about the power of nondeterminism, and more about the power of trusted versus untrusted advice. Specifically: trusting advice doesn't always buy (much) time in the randomized setting, as long as we have *some* trusted advice.

The main observation needed to make the circuits randomized rather than deterministic are:

1. Even if we allow our circuits to be randomized, for any fixed acceptable success probability, there is still a smallest circuit that computes any given function. This allows us to talk very concretely about what the exact circuit size for a problem is even for randomized circuits.
2. In our hard problem, we get our promise gap from our **PCP** system. So if $x \notin L$, no algorithm, not even a **PSPACE** algorithm, can make the verifier accept with high probability. And if $x \in L$ then there is a small randomized circuit to act as the prover.

In all cases of our proof, Arthur only asks Merlin for a program deciding some **SPACE**$[O(n)]$ complete problem. This advice does not depend on the specific input, only on the input size.

2 Preliminaries

We assume some familiarity with basic complexity theory. See Arora and Barak's book for background [AB09]. For our model of computation, we use multi-tape Turing machines. We assume the Turing machine has a read only input tape and we say it is space $S(n)$ if it only ever uses $S(n)$ cells in any of its working tapes. We assume that for any functions $S(n)$ and $T(n)$ used in this paper for space and time bounds that both S and T are computable in time $O(\log(T))$.

2.1 Complexity Classes

The most studied family of complexity classes are the time bounded Turing machines.

Definition 2.1.1 (**TIME**)**.** *For any function $f : \mathbb{N} \to \mathbb{N}$,* **TIME**$[f(n)]$ *is the class of languages, L, such that there is a time $f(n)$ deterministic Turing machine M deciding L. Similarly,* **NTIME**$[f(n)]$ *is the class of languages, L, such that there is a time $f(n)$ nondeterministic Turing machine M deciding L.*

A randomized algorithm is a deterministic algorithm with an extra input tape for randomness.

Now recall that **MA** is the complexity class of problems with polynomial sized certificates that can be verified with bounded error by a randomized, polynomial time algorithm. This is like **NP** with a randomized verifier.

Definition 2.1.2 (**MATIME**)**.** *For any function $f : \mathbb{N} \to \mathbb{N}$,* **MATIME**$[f(n)]$ *is the class of languages, L, such that there is a time $f(n)$ algorithm M taking three inputs, an input x, a random input r, and a witness w, so that*

Completeness *If $x \in L$ and $n = |x|$, then there exists w with $|w| \le f(n)$ such that*

$$\Pr_r[M(x, r, w) = 1] > 2/3.$$

Soundness *If $x \notin L$, then for every w,*

$$\Pr_r[M(x, r, w) = 1] < 1/3.$$

Remark (Perfect Completeness)**.** *We note that our circuit lower bounds for deterministic circuits have perfect completeness, that is when $x \in L$ we have $\Pr_r[M(x, r, w) = 1] = 1$. We only define* **MATIME** *with imperfect completeness for our randomized circuit results.*

An algorithm with trusted advice is an algorithm with an extra input for advice, where the advice is fixed for every input of a given length. Complexity class **MATIME**$[f(n)]/1$ is **MATIME**$[f(n)]$ with 1 bit of trusted advice.

Definition 2.1.3 (**MATIME**/1)**.** *For any function $f : \mathbb{N} \to \mathbb{N}$, define* **MATIME**$[f(n)]/1$ *as the set of languages, L, such that there is a function $b : \mathbb{N} \to \{0, 1\}$ and a time $f(n)$ randomized algorithm M taking four inputs, an input x, a random input r, a witness w, and an advice bit such that*

Completeness *If $x \in L$ and $n = |x|$, then there exists w with $|w| \le f(n)$ such that*

$$\Pr_r[M(x, r, w, b(n)) = 1] > 2/3.$$

Soundness *If $x \notin L$ and $n = |x|$, then for every w,*

$$\Pr_r[M(x, r, w, b(n)) = 1] < 1/3.$$

As described in Section 1.3 section, we also define input oblivious Merlin-Arthur.

Definition 2.1.4 (**OMATIME**/1)**.** *For function $f : \mathbb{N} \to \mathbb{N}$, define* **OMATIME**$[f(n)]/1$ *as the set of languages, L, such that there is a trusted advice function $b : \mathbb{N} \to \{0,1\}$, an untrusted advice function $w : \mathbb{N} \to \{0,1\}^*$ with $|w(n)| \le f(n)$ and a time $f(n)$ randomized algorithm M taking four inputs, an input x, a random input r, untrusted advice, and a trusted advice bit such that*

Completeness *If $x \in L$ and $n = |x|$, then*

$$\Pr_r[M(x, r, w(n), b(n)) = 1] > 2/3.$$

Soundness *If $x \notin L$ and $n = |x|$, then for every w',*

$$\Pr_r[M(x, r, w', b(n)) = 1] < 1/3.$$

We let **SIZE** denote the class of languages with circuits of a given size.

Definition 2.1.5 (**SIZE**)**.** *For any function $f : \mathbb{N} \to \mathbb{N}$,* **SIZE**$[f(n)]$ *is the class of languages, L, where for each input length n, there is a circuit of size $f(n)$ with n inputs deciding L for inputs of length n.*

Further, **SIZE**$[O(f(n))]$ *is the class of languages, L, such that for some $g(n) = O(f(n))$, we have $L \in$* **SIZE**$[g(n)]$*. Similarly for* **SIZE**$[o(f(n))]$*.*

A useful fact about circuits is that Turing machines can be efficiently converted to circuits [HS66; PF79], and Turing machines can efficiently simulate circuits [Fis74; Pip77] (see [LW12, Theorem 3.1]).

Lemma 2.1.6 (Turing machines Efficiently Compute Circuits)**.** *There is a Turing machine running in time $\tilde{O}(n)$ that takes as input a length n description of a circuit C which computes some function $f : \{0,1\}^n \to \{0,1\}^n$ and an input $x \in \{0,1\}^n$ and outputs $f(x)$.*

We use this fact to allow our algorithms to efficiently run the circuits provided in the **MA** proof.

To show that $L \notin$ **SIZE**$[o(f(n))]$, we will show that for some constant $c > 0$, for infinitely many n, language L on length n inputs requires circuits of size at least $cf(n)$. This implies that for any $g(n) = o(f(n))$, language L must have size greater than $g(n)$ infinitely often, because eventually, $g(n)$ must stay below $cf(n)$.

While super linear circuit lower bounds have been hard to prove, one can easily get linear circuit lower bounds for any language that depends on every bit in the input, for instance, the parity function.

Lemma 2.1.7 (Parity Requires Large Circuits)**.** *Let L be the language of strings with an odd number of 1s. Then $L \in$* **TIME**$[O(n)]$*, but L on length n inputs requires circuits of size $n/2$.*

This lower bound comes from the fact that parity as a function depends on every input, and since each gate only has fan in 2, we need at least $n/2$ gates to make the circuit a function of every input. Similarly, since **TIME**$[O(n)] \subseteq$ **MATIME**$[O(n)]$, we get a similar result for **MATIME**. Since we can run an algorithm that only computes parity on some specific subset of the input, we can extend this to sublinear time as well.

Corollary 2.1.8 (Sub-linear Circuit Lower Bounds Are Easy)**.** *For any time constructible $S(n) \le n/2$, there exists a language $L \in$* **TIME**$[O(S(n))]$ *but for every n, language L on length n inputs requires circuits of size $S(n)$.*

We assume a model of computation where n is provided to the algorithm in binary. Then the language is just parity on the first $2S(n)$ bits. By Lemma 2.1.7, this requires a size $S(n)$ circuit. Since $S(n)$ is time constructible, we can construct $S(n)$ then run parity on the first $S(n)$ bits in $O(S(n))$ time.

We use big O and little o notation extensively in this paper. We will use the result that sub-polynomial functions remain sub-polynomial when composed with polynomials.

Lemma 2.1.9 (Composing Sub-polynomials with Polynomials gives Sub-polynomials.)**.** *If $h(n) = o(1)$, and for some constant k, we have $D(n) = O(n^k)$, then for some $h'(n) = o(1)$,*

$$D(n)^{h(D(n))} = O(n^{h'(n)}).$$

Proof. Let $G(n) = n^{h(n)}$ so that $G(D(n)) = D(n)^{h(D(n))}$. Then we can bound $\log(G(n))$:

$$\log(G(n)) = h(n)\log(n) = o(\log(n)).$$

Using that $\log(n)$ is increasing and unbounded, we can bound $\log(G(D(n)))$.

$$\log(G(D(n))) = o(\log(D(n))) = o(\log(n)).$$

This is equivalent to, for some $h'(n) = o(1)$,

$$\log(G(D(n))) = h'(n)\log(n).$$

This gives the result.

$$D(n)^{h(D(n))} = 2^{\log(G(D(n)))} = n^{h'(n)}.$$

□

2.2 Randomized Circuits

Since we give a more general result for randomized circuits in Theorem 1.3.1, and the notation for randomized circuits are less well established, we establish notation here. First, we define unbounded error randomized circuits, then we will define bounded error circuits as a subset.

Definition 2.2.1 (Non-Uniform, Unbounded Error Randomized Circuits)**.** *We define $\mathcal{C}$ to be the set of boolean circuits such that each circuit has a pair of inputs, a regular input and a randomness input, and outputs a boolean value. Then for $C \in \mathcal{C}$ that takes a length n regular input and a length m randomness input, and a regular input x with $|x| = n$, define $C(x)$ to be the random variable of the output of C when given x as the regular input and a uniformly random m bit string for the randomness input.*

Remark (Using RAM Algorithms Instead of Circuits)**.** *Our upper and lower bounds are stated in terms of Turing machines, but we can extend this to RAM model of computation. One just needs to define a time bounded size, randomized program in that model and show that similar results as this section hold. For instance, if there is an efficient way to simulate other programs in that model, you can define $\mathcal{C}$ to be the description of a very long program which is simulated for a bounded amount of time.*

Since Turing machines and circuits efficiently simulate one another [Fis74; Pip77; LW12; HS66; PF79], we also have the following.

Lemma 2.2.2 (Simulating Unbounded Error, Randomized Circuits)**.** *Given $C \in \mathcal{C}$ with input length $n \leq |C|$, and an input x with $|x| = n$ there is a time $\tilde{O}(|C|)$, randomized multi-tape Turing machine outputting the random variable $C(x)$.*

This means that simulating an element $C \in \mathcal{C}$ only ever adds a polylogarithmic factor. We can also define a subset of $\mathcal{C}$ with bounded error.

Definition 2.2.3 (Non-Uniform, Bounded Error Randomized Circuits)**.** *We define $\mathcal{BPC} \subseteq \mathcal{C}$ to be the set of $C \in \mathcal{C}$ such that for all x whose lengths are the input length of C, either $\Pr[C(x) = 1] \geq \frac{2}{3}$ or $\Pr[C(x) = 1] \leq \frac{1}{3}$.*

Since that notation $\mathbf{BPTIME}[f(n)]/f(n)$ is a bit long, we introduce more compact notation for non-uniform, bounded error, randomized circuit size.

Definition 2.2.4 (Non-Uniform Randomized Circuit Complexity)**.** *For any function $T(n)$, define the complexity class $\mathbf{BPSIZE}[T(n)]$ as the set of languages L such that for every integer n there is a randomized circuit $C_n \in \mathcal{BPC}$ with $|C_n| \leq T(n)$ such that for all x with $|x| = n$ we have*

Completeness: *If $x \in L$, then $\Pr[C_n(x)$ accepts$] \geq 2/3$.*

Completeness: *If $x \notin L$, then $\Pr[C_n(x)$ accepts$] \leq 1/3$.*

The main feature we need from $\mathcal{BPC}$ is that there is a specific function T that is the best time for any specific language L. In our exposition, we will refer to **BPSIZE** as randomized circuits. Any result you see in this paper with **OMATIME** or **BPSIZE** also holds for **MATIME** or **SIZE**. We just state the result in its most general form, which has randomized circuits and input oblivious advice.

Of course, to get our final stated results we need that $\mathbf{BPTIME}[f(n)]/f(n)$ is loosely equivalent to **BPSIZE**. Indeed this is so.

Lemma 2.2.5 (Equivalence Between **BPSIZE** and **BPTIME** with Advice)**.** *For any function $T(n) = \Omega(n)$, we have $\mathbf{BPTIME}[\tilde{O}(T(n))]/\tilde{O}(T(n)) = \mathbf{BPSIZE}[\tilde{O}(T(n))]$.*

Proof. For $L \in \mathbf{BPTIME}[T(n)]/T(n)$, this means there is a deterministic multi-tape Turing machine that has an extra randomness tape and an extra advice tape that solves the problem. By [HS65; PF79] there is a circuit of size $\tilde{O}(T(n))$ that simulates this Turing machine. After hard-coding the advice, it becomes a randomized circuit in $\mathbf{BPSIZE}[\tilde{O}(T(n))]$.

For $L \in \mathbf{BPSIZE}[T(n)]$, there is a circuit that correctly decides L with high probability. By [Fis74; Pip77] there is a multi-tape Turing machine that can run that circuit in time $\tilde{O}(T(n))$. Just put the description of that circuit on the advice string, and we have an algorithm in $\mathbf{BPTIME}[\tilde{O}(T(n))]/\tilde{O}(T(n))$. □

To use our non-uniform randomized algorithms in our proof, we need the non-uniform hierarchy [AB09, Theorem 6.22], but for randomized circuits. Such a result follows from the fact that a bounded error randomized circuit gives an approximation of a function and even approximating most functions requires a large circuit size [ACR97], but also follows from the standard counting argument by Shannon [Sha49] and improved by Lupanov [Lup58]. See also [Weg87; FM05; SK12].

Lemma 2.2.6 (Non-Uniform, Bounded Error, Randomized Time Hierarchy)**.** *For every $S(n) < 2^n/n$, there is a language L in $\mathbf{SIZE}[S(n)]$ but not in $\mathbf{BPSIZE}[S(n)/10]$.*

2.3 PCPs and Composition

Two notations we will need to define **PCP**s are projection, and multi evaluation of a function.

Definition 2.3.1 (Projection)**.** *For any set Σ, naturals $n, m \in \mathbb{N}$, string $\pi = (\pi_1, \ldots, \pi_n) \in \Sigma^n$, and indices $I = (I_1, \ldots I_m) \in [n]^m$, we define the projection $\pi_I = (\pi_{I_1}, \ldots, \pi_{I_m})$. We may also write for $i \in [n]$, $\pi(i) = \pi_i$, and $\pi(I) = \pi_I$.*

Definition 2.3.2 (Multi-evaluation)**.** *For any integer n, alphabet Σ, and a function $Q : [n] \to \Sigma$, we define $Q(\cdot) = (Q(i))_{i \in [n]}$. Similarly if for some set Y we have that $Q : Y \times [n] \to \Sigma$, then for $y \in Y$ we define $Q(y, \cdot) = (Q(y, i))_{i \in [n]}$.*

In this paper, we will focus on time and space efficient, non-adaptive **PCP**s with perfect completeness. Because we need to pay close attention to the amount of time it takes to make a single query to the proof, we separate the algorithm for producing queries, Q, from the algorithm for verifying the response, V.

So at a high level, a **PCP** system does the following:

1. Chooses a common random string, r.
2. Runs query function Q with randomness r for $q(n)$ many times to get all query locations, $Q(r, \cdot)$.
3. Looks up all query locations, $Q(r, \cdot)$, into a provided proof, π, to get a proof window $\pi_{Q(r,\cdot)}$.

4. Runs verifier V with randomness r and proof window $\pi_{Q(r,\cdot)}$ and outputs if V accepts.

Then if the input is in a language L, we want some proof π to always make the verifier accept. But if an input is not in language L, we want for any proof π, the probability the verifier accepts to be small. We also want a prover, P, that can compute any symbol of the proof using low space.

Now we formally define a **PCP**.

Definition 2.3.3 (**PCP**). *We say that a language L has a non-adaptive* **PCP**, *A, with perfect completeness if there exists verifier V, prover P, and query function Q, such that, for some alphabet Σ, $\delta \in [0,1]$, and functions $r, l, q : \mathbb{N} \to \mathbb{N}$:*

1. *Q is an algorithm with three inputs, an input x of length n, randomness r of length $r(n)$, and an index $j \in [q(n)]$ and outputs an element of $[l(n)]$.*
2. *V is an algorithm with three inputs, an input of length n, randomness of length $r(n)$, and $q(n)$ symbols from Σ, and outputs either accept or reject.*
3. *P is an algorithm that takes two inputs, an input of length n, and an index $i \in [l(n)]$, and outputs a symbol from Σ.*

Completeness *If $x \in L$ and $n = |x|$, then there exists $\pi^x \in \Sigma^{l(n)}$ such that*

$$\Pr_r[V(x, r, \pi^x_{Q(x,r,\cdot)}) = 1] = 1,$$

and for every $i \in [l(n)]$, $P(x, i) = \pi^x_i$.

Soundness *If $x \notin L$ then for every π',*

$$\Pr_r[V(x, r, \pi'_{Q(x,r,\cdot)}) = 1] \leq \delta.$$

Then we also say:

1. *A has proof length $l(n)$.*
2. *A has alphabet Σ.*
3. *A has soundness δ.*
4. *A uses $q(n)$ queries.*
5. *A uses $r(n)$ bits of randomness.*
6. *If V runs in time $t(n)$, A has decision complexity $t(n)$.*
7. *If V runs in space $s(n)$, A has verifier space $s(n)$.*
8. *If P runs in space $s'(n)$, A has prover space $s'(n)$.*
9. *If Q is computable in time $t'(n)$, A has query time $t'(n)$.*

For convenience, we assume that any alphabet or field is always encoded with some canonical binary encoding. We generally will not worry too much about encoding as we switch models of computation and we will assume inputs are encoded in binary using a small power of two bits.

In this paper, any time we refer to a time T space S deterministic algorithm, we assume that both T and S can be computed in time $\tilde{O}(\log(T))$ and space $O(S)$. This is to simplify our **PCP** statements since the verifier needs to calculate these values efficiently. Importantly, the time bounds for our nonuniform algorithms, for instance **OMA** with one bit of advice, may *not* be efficiently computable. This is a side effect of our proof technique and we don't know how to avoid it while keeping our results tight.

To get our final results, we will use **PCP** composition of robust **PCP**s (**rPCP**) with **PCP**s of proximity (**PCPP**)[5]. An **rPCP** is a **PCP** so that when $x \notin L$, for any proof, most sets of queries to that proof return not only a rejected response, but a response that is far from any accepted response. And a **PCPP** of proximity is a **PCP** that makes few queries to part of its input, but still fails as long as that part is far from any accepted input. These two kinds of **PCP**s have a very natural composition, see [Ben+04].

To formally define a robust **PCP**s, we need to define relative Hamming distance.

Definition 2.3.4 (Distance)**.** *For $x, y \in \Sigma^n$, define distance by the function, Δ:*

$$\Delta(x, y) = \frac{\sum_{i\in[n]} 1_{x_i \neq y_i}}{n} = \Pr_{i\in[n]}[x_i \neq y_i].$$

For $Y \subseteq \Sigma^n$, define

$$\Delta(x, Y) = \min_{y\in Y} \Delta(x, y).$$

The only difference between a **PCP** (see Definition 2.3.3) and an **rPCP** is a strengthening of the of soundness to robust soundness. Robust soundness is that not only will $x \notin L$ not have any proof that makes it accept with high probability, but when the verifier looks at the proof, with high probability what they see will be far away from anything they would accept. Now we formally define an **rPCP**.

Definition 2.3.5 (Robust **PCP**)**.** *For language L with a non-adaptive* **PCP** *system, A, with verifier V, query function Q, and alphabet Σ, we say A is a robust* **PCP** *(***rPCP***) if:*

Robust Soundness *If $x \notin L$ then for every π', let $Y_r = \{\sigma : V(x, r, \sigma) = 1\}$ be the set of (local views of the) proofs that V would accept. Then*

$$\Pr_r[\Delta(\pi'_{Q(x,r,\cdot)}, Y_r) \leq \eta] < \delta.$$

Then we say A has robust soundness error δ and robustness parameter η.

Robustness is a standard property because many **PCP**s are sub-codes of locally testable codewords. In particular, they are Reed-Muller codes and query locations are curves or lines.

A **PCP** of proximity (**PCPP**) [Ben+04] verifies pairs of inputs together: an explicit input, and an implicit input.

- The explicit input is known to the verifier and contains the input x the **PCP** is trying to verify.
- The implicit input is not known by the verifier, it is only known by the prover. The verifier can access bits of the implicit input, but these count as queries. You should think of the implicit input as a partial witness for x, just one too large to read. In our application, it will be polynomially larger than x, so our verifier does not even have time to read it all.

Then the **PCPP** needs to verify that, conditioned on the first input, the second input is close to a valid implicit input. Now we define a **PCPP**.

Definition 2.3.6 (**PCP** of Proximity)**.** *Let L' be a language containing pairs, where if the first input is length n from alphabet Σ, the second input is $m(n)$ symbols from alphabet Σ'. We say L' has a* **PCP** *of proximity (***PCPP***), B, if for some verifier V, prover P, query function Q, alphabet Σ, constants $\delta, \eta \geq 0$, and functions $q, r, l : \mathbb{N} \to \mathbb{N}$:*

1. *Q is an algorithm with 4 inputs, an input x of length n, an index $i \in [m(n)]$, randomness r of length $r(n)$, and an index $j \in [q(n)]$, and outputs an element of $[m(n) + l(n)]$.*
2. *V is an algorithm that takes 4 inputs: an input of length n, randomness of length $r(n)$, and $q(n)$ symbols from Σ. The algorithm V either accepts or rejects.*

[5]A prior version of this paper uses decodable **PCP**s. A decodable **PCP** is a special case of a **PCPP** that decodes a random symbol of the input to check proximity. Our **PCP** *is* a decodable **PCP**, but we use **PCPP** since it is more standard.

3. *P is an algorithm that takes three inputs, an input of length n, some $m(n)$ symbols from Σ', and an index $i \in [l(n)]$, and outputs a symbol from Σ.*

Completeness: *For any $x \in \Sigma^n$ and $y \in \Sigma'^{m(n)}$ such that $(x, y) \in L'$, there exists a proof $\pi^{x,y}$ such that*

$$\Pr_r[V\left(x, r, (y, \pi^{x,y})_{Q(x,i,r,\cdot)}\right) = 1] = 1.$$

Further, for every $i \in [l(|x|)]$, we have $P(x, y, i) = \pi_i^{x,y}$.

Soundness: *For any x and y if for all y' such that $(x, y') \in L'$ we have that*

$$\Delta(y, y') > \eta,$$

then for any π' we have that

$$\Pr_r[[V\left(x, r, (y, \pi')_{Q(x,i,r,\cdot)}\right) = 1] \leq \delta.$$

Then we also say:

1. *B has proof length $l'(n)$.*
2. *B has alphabet Σ.*
3. *B has soundness δ.*
4. *B has proximity η.*
5. *B uses $q(n)$ queries.*
6. *B uses $r(n)$ bits of randomness.*
7. *If V runs in time $t(n)$, B has decoder time $t(n)$.*
8. *If P runs in space $s'(n)$, B has encoder space $s'(n)$.*
9. *If Q is computable in time $t'(n)$, B has query time $t'(n)$.*

This is also a natural property for **PCP**s to have since many **PCP**s encode their input as part of a Reed-Muller code, thus the implicit input can be locally decoded and compared to the provided input at random places. In fact, many **PCP**s are both robust and have proximity.

Definition 2.3.7 (Robust **PCP** of Proximity)**.** *For language L with a non-adaptive* **PCPP** *system, A, with verifier V, query function Q, alphabet Σ, and proximity η, we say A is a robust* **PCP** *of proximity (***rPCPP***) if:*

Completeness *If $(x, y) \in L$, then there exists $\pi^{x,y} \in \Sigma^{l(n)}$ such that*

$$\Pr_r[V(x, r, \pi^{x,y}_{Q(x,r,\cdot)}) = 1] = 1.$$

Robust Soundness *For any x, let $Y_r = \{\sigma : V(x, r, \sigma) = 1\}$ be the set of (local views of the) proofs that V would accept given explicit input x. If for some y, for all y' such $(x, y') \in L$ we have that that $\Delta(y, y') > \eta$ (where η is the proximity of A) then for every proof π', we have that*

$$\Pr_r[\Delta((y, \pi')_{Q(x,r,\cdot)}, Y_r) \leq \beta] < \delta.$$

Then we say A has robust soundness error δ and robustness parameter β.

Composing an **rPCP** and a **PCPP** gives a **PCP** with the number of queries of the **PCP** proximity. In our application, this allows us to reduce a **PCP** that uses $O(n)$ queries to one that uses $O(\log(n))$. This composition was proven in the same paper that introduced **PCPP** [Ben+04].

Theorem 2.3.8 (**PCP** Composition)**.** *Suppose L is a language with an* **rPCP***, A, with verifier V, prover P, and query function Q such that*

1. *Q runs in time $t(n)$.*
2. *P run in space $s(n)$.*
3. *V uses $r(n)$ bits of randomness.*
4. *A uses alphabet Σ.*
5. *A has robust soundness error δ.*
6. *A has robustness parameter η.*
7. *A has perfect completeness.*

Let $L' = \{((x,r),y) : V(x,r,y) = 1\}$. Let $n' = n + r(n)$. Suppose L' has a **PCPP** *system, B, with verifier V' and prover P' such that*

1. *P' runs in space $s'(n')$.*
2. *V' runs in time $t'(n')$.*
3. *B uses $q'(n')$ queries.*
4. *B has query time $t^*(n')$.*
5. *B uses alphabet Σ'.*
6. *B has soundness δ'.*
7. *B has proximity η.*
8. *B has perfect completeness.*
9. *B uses $r'(n)$ bits of randomness.*

Then L has a **PCP** *system, C, such that*

1. *C has decision complexity $O(t'(n'))$*
2. *C uses $O(q'(n'))$ queries.*
3. *C has prover space $O(s(n) + t(n) + s'(n'))$.*
4. *C uses alphabet $\Sigma' \cup \Sigma$.*
5. *C has query time $O(t(n) + t^*(n'))$.*
6. *C has soundness $\delta + \delta'$.*
7. *C has perfect completeness.*
8. *C uses $r(n) + r'(n')$ bits of randomness.*

Further, if A is a **rPCPP** *with proximity η', then so is C with the same proximity η'. Similarly, if B is an* **rPCPP** *with robust soundness δ' and robustness parameter β, then C also has robust soundness $\delta + \delta'$ and robustness parameter β.*

Proof. Most of these come from [Ben+04], notably the decision complexity, number of queries, randomness, completeness, and soundness. For details on those parameters, review the original paper. The alphabet is also very direct. New parameters we give not in the original proof include the prover space, and the query time[6]. Here we briefly justify them.

Recall that the composed **PCP** verifier, first chooses the randomness for the outer, robust **PCP**, A. Let σ be the local view of the outer proof queried for that choice of randomness. Then the verifier chooses the randomness for the inner **PCP** of proximity, B, for a proof that the **rPCP** verifier would accept σ. The implicit input to B is σ. Now we just show that the query locations can be computed efficiently, and the proof can be computed space efficiently.

For query time, the query function in B either queries its implicit input, or its proof. If it queries its proof, the query function of C only needs to evaluate the query function of B. If it chooses to query the implicit input, we translate that to the outer proof using the query function for A. This takes time $t^*(n') + t(n)$.

For prover space, see that the composed prover only needs to either output a symbol from the prover for A, or output a symbol from the prover for B. The space for a symbol from the prover for A is $s(n)$. To output a symbol from the prover for B, we need to simulate the prover for B. Unfortunately, the prover for B runs on input which is the proof for A at the query locations. Thus to access the input for B, we need the space both to compute the query locations of A, and to compute the proof in A, which requires space $s(n) + t(n) + s'(n')$. □

Another way to view a **PCP** is as a multi-prover interactive protocol (**MIP**). In an **MIP**, there is a weak verifier that can ask questions to multiple, powerful, independent provers. In **MIP**s we bound the time of the verifier, the number of provers, and the number of questions the verifier can ask per prover (called the number of rounds). A q query **PCP** is equivalent to a q prover, one round **MIP**. By convention, an **MIP** protocol needs to have a constant soundness less than 1.

Definition 2.3.9 (**MIP**)**.** *We say a language L has a two prover, one round* **MIP** *protocol for L with verifier time $T(n)$ if L has a* **PCP** *system with*

1. *Two queries,*
2. *Perfect completeness and constant soundness error $1 - \Omega(1)$.*
3. *For any choice of randomness, both queries and the verifier decision can be computed in time $T(n)$.*

We note that there are well known parallel repetition schemes that can boost the soundness from any constant to any other constant with only a constant factor overhead [Raz98; Hol07; Rao08]. Thus we consider any constant soundness sufficient.

3 Efficient PCP To Fine Grained Lower Bounds

Our analysis depends on the circuit complexity of some **PSPACE** complete problem. So we start by choosing a **SPACE**$[O(n)]$ complete problem. We use a version of `SPACE TMSAT` (on page 83 of [AB09]).

Definition 3.0.1 (Specific Problem)**.** *`SPACE TMSAT` is the language*

$$\{(M, x, 1^n, 0^*) : \text{Turing machine } M \text{ accepts } x \text{ using at most } n \text{ space.}\}$$

Note: `SPACE TMSAT` $\in$ **SPACE**$[O(n)]$ and `SPACE TMSAT` is **SPACE**$[O(n)]$ complete. The 0^* is just there to make it explicit the language is paddable. In particular, this means that the circuit complexity of `SPACE TMSAT` is non-decreasing.

Lemma 3.0.2 (`SPACE TMSAT` Circuit Complexity is Non-Decreasing)**.** *If $A'(n)$ is the size of the minimum circuit solving `SPACE TMSAT` for inputs of length n, then $A'(n)$ is non-decreasing.*

Proof. Let C be the circuit of size $A'(n+1)$ solving `SPACE TMSAT` for length $n+1$ inputs. Then to get a circuit for length n inputs, use C with an extra 0 hard coded into the last input. The resulting circuit will be at most the size of C and solve length n inputs. Thus $A'(n+1) \geq A'(n)$. □

[6]Query time was included in the original result as part of the time of V_{comp}, but with a very poor upper bound.

Then using Theorem 1.1.3, we can get a **PCP** for `SPACE TMSAT` by setting $T = 2^{O(n)}$ and $S = O(n)$. This can be turned into a **PCP** with a binary alphabet by replacing every query for a symbol in Σ with $O(\log(n))$ queries to the individual bits of that symbol.

Corollary 3.0.3 (**PCP** for `SPACE TMSAT`). *There is a* **PCP** *for* `SPACE TMSAT` *with:*

1. *Decision complexity* $\tilde{O}(n)$*.*
2. *Query time* $\tilde{O}(n)$*.*
3. $\mathsf{polylog}(n)$ *queries.*
4. *Binary alphabet.*
5. *Log of proof length* $\tilde{O}(n)$*.*
6. *Prover space* $\tilde{O}(n)$*.*
7. *Soundness* $1/2$ *and perfect completeness.*

We denote the set of languages recognized by families of size $f(n)$ randomized circuits as $\mathbf{BPSIZE}[f(n)]$. We emphasize that $\mathbf{BPSIZE}[\tilde{O}(f(n))]$ is equivalent to $\mathbf{BPTIME}[\tilde{O}(f(n))]/\tilde{O}(f(n))$, we just use **BPSIZE** since it is more convenient. For more details, see Section 2.2. Further, in this section, when we say circuit we mean randomized circuit, and we say Merlin Arthur, we mean oblivious Merlin Arthur.

We prove three different $\mathbf{OMATIME}/1$ lower bounds that are based on three different hard problems. Different ones work better in different parameter regimes. After constructing them all, we show we always fall into some range of parameters so that we can get the lower bounds of Theorem 1.1.1.

3.1 Implicitly Encoding Advice in Input Length

In each of our cases, we will use advice to find the size of some prover circuit. To do this, we implicitly encode a number in the input length. If that implicitly encoded number describes the size, our advice bit will be 1. Otherwise, the advice bit is 0.

For any input length $n \in \mathbb{N}$, for some $l \in \mathbb{N}$, we have $n \in [2^l, 2^{l+1})$. For such an l, there is some $m \in \mathbb{N}$ such that $n = 2^l + m$. This m, or equivalently this l, is our implicitly encoded number. Because we will use this decomposition a lot, we will explicitly define some functions that perform this decomposition.

Definition 3.1.1 (Implicit Encoding In Input)**.** *For natural* $n \geq 1$*, let* $l \geq 0$ *be an integer so that* $n \in [2^l, 2^{l+1})$*, and* $m \geq 0$ *be an integer so that* $n = 2^l + m$*. Then define* $\mu(n) = m$ *and* $\rho(n) = l$*.*

There is a simple interpretation of this $m = \mu(n)$ and $l = \rho(n)$ in terms of the binary representation of n. You can think of l as the length of the binary number, and m the binary number after the top bit is removed.

3.2 `SPACE TMSAT` $\notin \mathbf{P/poly}$

In this case, we follow the proof in the original work [San07] where $\mathbf{PSPACE} \not\subset \mathbf{P/poly}$. We present the same arguments here in more generality and with more precise parameters.

When $\mathbf{PSPACE} \not\subset \mathbf{P/poly}$, the circuit complexity of different input sizes for `SPACE TMSAT` could change drastically and in a way that may be hard to analyze. This is an issue because the **PCP** for `SPACE TMSAT` needs a prover with a longer input than the input being verified, thus might require a much larger circuit.

Instead, we use a randomly downward self reducible **PSPACE** complete language. Specifically, a language that has a sound interactive protocol with queries the same length as its input and whose prover is the language itself. We cite the result from Lemma 11 in [San07]:

Lemma 3.2.1 (Same Size, Self Proving **PSPACE** Complete Language)**.** *There is a* **PSPACE***-complete language* Y *and a probabilistic polynomial-time oracle Turing machine* M *such that for any input x:*

1. M *only asks its oracle queries of length* $|x|$*.*

2. *If M is given Y as oracle and $x \in Y$, then M accepts with probability* 1.

3. *If $x \notin Y$, then irrespective of the oracle given to M, M rejects with probability at least* $1/2$.

The important feature of language Y is that for an input x, the prover for x is the same language Y, and queries to the prover have the same length as x. This means Y, and the prover for Y, have the same circuit.

Now using Lemma 3.2.1, we can get the following bound.

Lemma 3.2.2 (Bound Using Padded Y as Hard Problem)**.** *Using Y from Lemma 3.2.1, if for some $g(n) = \omega(1)$ we have $Y \notin \mathbf{BPSIZE}[O(n^{g(n)})]$ then for any time constructible, non-decreasing, unbounded $S(n)$ such that $S(n) = o(n^{g(n)})$, for some*[7] *$f(n) = o(1)$:*

$$\mathbf{OMATIME}[O(S(n)^{1+f(n)})]/1 \not\subset \mathbf{BPSIZE}[o(S(n/4))].$$

Proof. Let $a > 0$ be the constant so that the verifier (M in Lemma 3.2.1) for Y's interactive protocol runs in time $O(n^a)$.

Now we define our language, W, in $\mathbf{OMATIME}[S(n)^{1+o(1)}]/1$ but not in $\mathbf{BPSIZE}[o(S(n))]$. For any input size, n, using Definition 3.1.1, let $m = \mu(n)$ and $l = \rho(n)$. Let our advice bit be 1 if

1. Y on length m inputs does not have circuits of size $m^{g(m)}$,

2. Y on length m inputs has circuits with size $S(n)$, and

3. for all integers l' with $l' < l$ and $2^{l'} > m$, Y on length m inputs does not have randomized circuits of size $S(2^{l'} + m)$.

   This condition requires the advice bit to only be 1 for a given m exactly once, whenever it can be used first. This simplifies the analysis, giving us a one to one function from n where the advice bit is 1, to m.

Then $x \in W$ for some x with $|x| = n$ if and only if the advice bit is 1 and for some $y \in Y$ with $|y| = m$ we have $x = y1^{n-m}$.

Now we will show that infinitely often the advice bit is 1 and W does not have randomized circuits with size $S(n/4)$.

Since $Y \notin \mathbf{BPSIZE}[O(n^{g(n)})]$, for some infinite set U', for $m \in U'$, the language Y on input length m does not have circuits of size $m^{g(m)}$. Since $S(m) = o(m^{g(m)})$, for some n', for all $m \geq n'$, $S(m) < m^{g(m)}$. So let $U = U' \cap [n', \infty)$. See that $|U| = \infty$.

For $m \in U$, since $S(n)$ is non-decreasing and unbounded, for large enough l, language Y on length m inputs has circuits of size at most $S(2^l + m)$. Then there is a smallest such l with $2^l > m$ and for $n = 2^l + m$, the language Y on length m inputs has circuits of size $S(n)$. For such n, the advice bit is 1.

Now either $2^{l-1} \leq m$, or $2^{l-1} > m$.

$2^{l-1} \leq m$ Then $2m \geq 2^l$, and $m > n/4$. Since $m \in U$, language Y on length m inputs does not have circuits of size $S(m)$. Since $S(n)$ is monotone, Y on length m inputs also doesn't have circuits of size $S(n/4)$.

$2^{l-1} > m$ Then by choice of l, Y on length m inputs does not have circuits of size $S(2^{l-1} + m)$. Since by definition of n, we have $2^{l-1} + m > n/2$ and $S(n)$ is monotone, Y on length m inputs does not have circuits of size $S(n/2)$.

So W does not have circuits with size less than $S(n/4)$.

Since U has infinitely many elements, and for every $m \in U$, there is an $n > m$ such that W on length n inputs does not have circuits of size $S(n/4)$, for infinitely many n, language W on length n inputs does not have circuits of size $S(n/4)$. So $W \notin \mathbf{BPSIZE}[o(S(n/4))]$.

Now we define $f(n)$. Let $\mu_1(n)$ be the partial function from n where the advice bit is 1, to $\mu(n)$. We claim $\mu_1(n) = \omega(1)$. This is because for any m, the advice bit can only be 1 once. Thus μ_1 is one to one.

[7]More generally, if $A(n)$ is the minimum circuit size for Y, the $\mathbf{MA}$ verifier will run in time similar to $S(n)\,\mathsf{poly}(A^{-1}(S(n)))$. Since $A^{-1}(n)$ is not simple, we avoid proving a more detailed result here.

Any one to one function into the naturals is $\omega(1)$, since for any b, there is a max n such that for some $m < b$, $\mu_1(n) = m$, and for all $i > n$, $\mu_1(i) \geq b$. Then let

$$D(n) := \begin{cases} \mu_1(n) & \text{Advice bit for } n \text{ is } 1 \\ D(n-1) & \text{Otherwise} \end{cases}.$$

Since $\mu_1(n) = \omega(1)$, we also have $D(n) = \omega(1)$. Then since $g(n) = \omega(1)$, we also have that for $f(n) = a/g(D(n))$, we have $f(n) = o(1)$.

Now we show that $W \in \mathbf{OMATIME}[O(S(n)^{1+f(n)})]/1$. If the advice bit is 0, this is true trivially. Suppose that the advice bit is 1.

For an n where the advice bit is 1, inputs of length $m = \mu(n)$ for Y have circuits of size $S(n)$, which can be guessed. Then from Lemma 3.2.1, there is a time m^a algorithm that can verify membership in Y with a circuit for Y. This gives an $\mathbf{MA}$ protocol for Y on length m that runs in time $O(S(n)m^a)$.

Then since the advice bit is 1, there are circuits for length m instances of Y with size $S(n)$, but not $m^{g(m)}$. Thus $S(n) > m^{g(m)}$, so $S(n)^{1/g(m)} > m$. Since we started with a randomized circuit for Y, we only know there is a size $S(n)$ circuit for Y that outputs Y with probability 2/3. So we actually need run this circuit $O(\log(m))$ times so that our randomized circuit for Y agrees with Y with probability $1 - \frac{1}{3m^a}$, and simulating the circuit may add a $\mathsf{polylog}(m)$ overhead. So the time of the $\mathbf{OMA}$ verifier is at most $O(S(n)\,\mathsf{polylog}(m)m^a) = O(S(n)S(n)^{a+1/g(m)}) = O(S(n)^{1+f(n)})$.

The $\mathbf{OMA}$ protocol is complete and sound since the protocol for Y is. That is, if $x \notin L$ and the advice bit is one, by the soundness of Y the protocol will not accept with probability greater than 1/3. If $x \in L$ and the advice bit is one, then there is a randomized circuit that will compute Y with probability $1 - \frac{1}{3m^a}$, thus will accept if all the queries to the circuit agree with Y, which happens with probability $\frac{2}{3}$. So $W \in \mathbf{OMATIME}[O(S(n)^{1+f(n)})]/1$.

Therefore

$$W \in \mathbf{OMATIME}[O(S(n)^{1+f(n)})]/1 \setminus \mathbf{BPSIZE}[o(S(n/4))].$$

□

We show that when $\mathbf{PSPACE} \not\subset \mathbf{P/poly}$, there is some $g(n) = \omega(1)$ such that $Y \notin \mathbf{BPSIZE}[n^{g(n)}]$. Thus we can apply Lemma 3.2.2.

Corollary 3.2.3 (Bound if **PSPACE** does not have Polynomial Sized Circuits). *If `SPACE TMSAT` $\notin \mathbf{P/poly}$, then for any $k > 0$, and some $f(n) = o(1)$:*

$$\mathbf{OMATIME}[O(n^{k+f(n)})]/1 \not\subset \mathbf{BPSIZE}[O(n^k)].$$

Proof. One can de randomize circuits with only a polynomial overhead. So `SPACE TMSAT` $\notin \mathbf{P/poly}$ also means that `SPACE TMSAT` also does not have randomized circuits. We want to use Lemma 3.2.2 with $S(n) = n^k \log(n)$ and some $g(n) = \omega(1)$. Let Y be the language from Lemma 3.2.1.

Since `SPACE TMSAT` is in **PSPACE**, `SPACE TMSAT` $\notin \mathbf{P/poly}$ and Y is **PSPACE** complete, $Y \notin \mathbf{P/poly}$. We will show, since $Y \notin \mathbf{P/poly}$, for some $g(n) = \omega(1)$, we have $Y \notin \mathbf{BPSIZE}[o(n^{g(n)})]$.

Let $A(n)$ be the size of the smallest circuit deciding Y on length n inputs. Let $g'(n) = \frac{\log(A(n))}{\log(n)}$. Suppose for contradiction that $g'(n)$ was bounded above by a constant, c. Then for all n, we have $g'(n) \leq c$ and $A(n) = n^{g'(n)} \leq n^c$. Thus Y has polynomial sized circuits. But Y doesn't, so $g'(n)$ is unbounded.

Let $g^*(n) = \max_{i \in [n]} g'(i)$. Since $g^*(n) \geq g'(n)$, we also know $g^*(n)$ is unbounded. By definition, $g^*(n)$ is non-decreasing. Thus $g^*(n) = \omega(1)$.

For infinitely many n, we know $g'(n) = g^*(n)$, since $g'(n)$ is unbounded. So for n such that $g'(n) = g^*(n)$, our problem Y does not have circuits of size less than $n^{g'(n)} = A(n)$. So infinitely often, Y does not have circuits of size $n^{g^*(n)}/2$. Thus $Y \notin \mathbf{BPSIZE}[o(n^{g^*(n)})]$.

Now let $g(n) = g^*(n) - 1$. Then $g(n) = \omega(1)$ and $n^{g(n)} = o(n^{g^*(n)})$, so $Y \notin \mathbf{BPSIZE}[O(n^{g(n)})]$.

Since $g(n) = \omega(1)$, see that $n^k \log(n) = o(n^{g(n)})$. Then using Lemma 3.2.2, we have that for some $f(n) = o(1)$,

$$\mathbf{OMATIME}[O((n^k \log(n))^{1+f(n)})]/1 \not\subset \mathbf{BPSIZE}[o((n/4)^k \log(n/4))].$$

Now to simplify this. Since k is a constant, we have that $n^k = o((n/4)^k \log(n/4))$. Thus

$$\mathbf{BPSIZE}[O(n^k)] \subset \mathbf{BPSIZE}[o((n/2)^k \log(n/2))].$$

Now for $f'(n) := kf(n) + (1 + f(n))\frac{\log(\log(n))}{\log(n)}$ we have $f'(n) = o(1)$ and $(n^k \log(n))^{1+f(n)} = n^{k+f'(n)}$. Thus

$$\mathbf{OMATIME}[O((n^k \log(n))^{1+f(n)})]/1 \subseteq \mathbf{OMATIME}[O(n^{k+f'(n)})]/1.$$

Together

$$\mathbf{OMATIME}[O(n^{k+f'(n)})]/1 \not\subset \mathbf{BPSIZE}[O(n^k)].$$

□

3.3 `SPACE TMSAT` $\in$ $\mathbf{BPSIZE}[n^{1+o(1)}]$

The idea in this case is to use a brute force, small space algorithm that finds a problem not in a fixed polynomial size. In particular, for circuit size $S(n)$, the brute force algorithm uses space $O(S(n))$ to compute some function with minimum circuit size $\Theta(S(n))$. Then we want to simulate the **PCP** from Corollary 3.0.3 to prove the output of this algorithm. Since the **PCP** is efficient, the prover for this algorithm does not use much more space than the brute force algorithm itself.

If `SPACE TMSAT` has almost linear sized circuits, the prover doesn't require much larger circuits than the space of the prover. Finally, our **PCP** is efficient, so the time of the **MA** verifier isn't much more than the size of the prover circuit. So the **MA** protocol doesn't require much more time than the size of the circuit it proves the output of.

If `SPACE TMSAT` requires larger circuits, say quadratic circuits, then the size of the prover circuits would be quadratically larger than the input length of the prover. That is, the prover circuit would be quadratically larger than the circuit it is trying to prove. This would give quadratic overhead for the **MA** verifier time over the size of the circuit it verifies. So this construction only works well enough when `SPACE TMSAT` has almost linear sized circuits.

Lemma 3.3.1 (Bound From Exhaustive Search As Hard Problem)**.** *If for some non-decreasing $A(n)$ we have `SPACE TMSAT` $\in \mathbf{BPSIZE}[O(A(n))]$, then there is some non-decreasing $B(n) = \tilde{\Theta}(n)$ such that for any time constructible, non-decreasing $S(n)$ with $S(n) < \frac{2^n}{n}$ and $S(n)2^n = \omega(A(B(S(n))))$:*

$$\mathbf{OMATIME}[\tilde{O}(A(B(S(n))))]/1 \not\subset \mathbf{BPSIZE}[o(S(n/2))].$$

We will use this theorem with $A(n) = n^{1+o(1)}$, so one should think of A and B as being almost linear. The $B(n)$ in this problem comes directly from the prover space and the log of the proof length[8] of our **PCP** given in Corollary 3.0.3. The outer polylogarithmic factors in the **MA** verifier time come from the number of queries made by the **PCP**, the query time, and the **PCP** decision complexity.

Remark (Why is There Advice?)**.** *It may be not clear on first inspection why this protocol needs advice. Indeed, if we said `SPACE TMSAT` $\in \mathbf{BPSIZE}[n^{1+\epsilon}]$ for some small constant ϵ, we wouldn't need advice at all. We could calculate A, B, and S explicitly.*

Unfortunately, we want to handle `SPACE TMSAT` $\in \mathbf{BPSIZE}[n^{1+o(1)}]$, and we can't guarantee that this $o(1)$ term is efficiently computible. This is also why we need advice: to tell us the circuit complexity of A. This is also why we need to pad our input length, to encode the advice in the input length, allowing us to reduce the number of advice bits.

Proof. One can show `SPACE TMSAT` requires circuits of size $\Omega(n)$ since it can compute parity and thus needs to read most of the bits in the input, so $A(n) = \Omega(n)$. If $S(n) = O(n)$, then we can prove the theorem statement using parity as the hard problem as in Corollary 2.1.8. Otherwise, we can assume $S(n) > 10n$.

The proof proceeds in five steps, which we outline first.

1. Find a language $L \in \mathbf{SPACE}[\tilde{O}(S(n))] \setminus \mathbf{BPSIZE}[S(n)/10]$. In particular, for every input length n, language L has circuits of size $S(n)$ but not $S(n)/10$.

[8]The log of the proof length of a **PCP** gives the length of a query to the prover.

2. Reduce L to `SPACE TMSAT` and use Corollary 3.0.3. In particular, find a circuit, C_n, for the prover in an **MA** protocol for L on length n inputs.

3. Define our advice bit to implicitly give an upper bound for the size of C_m for some m within a factor of 2 of n. Define W to be length m elements of L, padded to length n.

   Specifically, we think of m as a fixed power of 2, and the padding is the advice on how big C_m is. Said another way, the padding length is the real advice, and the advice bit is a hack to only run the algorithm when the advice in the padding length is right. Using a fixed m and letting n vary makes the argument slightly easier.

4. Show that infinitely often the advice bit is 1 and W does not have small circuits.

5. Show that W has an efficient **MA** protocol.

With that outline in mind, let us begin the proof.

1. Find a language $L \in \mathbf{SPACE}[\tilde{O}(S(n))] \setminus \mathbf{BPSIZE}[o(S(n))]$.

   By theorem premise $S(n) < 2^n/n$. So from the non-uniform, randomized algorithm time hierarchy (see Lemma 2.2.6), there is a language $L \in \mathbf{BPSIZE}[S(n)] \setminus \mathbf{BPSIZE}[S(n)/10]$. In particular, for every n, language L on length n has circuits size $S(n)$ but not size $S(n)/10$.

   Consider an algorithm, M, recognizing such an L which checks all circuits of size $S(n)$, and compares them with every circuit of size $S(n)/10$ on every input, and returns the output from the first circuit of size $S(n)$ that disagrees with every circuit of size $S(n)/10$ on some input. See that this is also enough space to check every randomness input to a circuit, so it can verify whether a circuit really is bounded error.

   Then M runs in space $\tilde{O}(S(n))$ (there may be some polylog factors between the size of the circuit, its description size, and the space of the simulation) and recognizes an $L \notin \mathbf{BPSIZE}[S(n)/10]$. So we have an $L \in \mathbf{SPACE}[\tilde{O}(S(n))] \setminus \mathbf{BPSIZE}[S(n)/10]$. In particular, for every n, language L on length n does not have circuits of size $S(n)/10$.

2. Reduce L to `SPACE TMSAT` and use Corollary 3.0.3.

   Since M only uses $g(n)$ space, for some $g(n) = \tilde{O}(S(n))$, we know $x \in L$ if and only if $(M, x, 1^{g(n)}, 0) \in$ `SPACE TMSAT`. We know `SPACE TMSAT` on length $\tilde{O}(S(n))$ inputs has a **PCP** system from Corollary 3.0.3 that uses $\mathsf{polylog}(S(n))$ many length $\tilde{O}(S(n))$ queries to a space $\tilde{O}(S(n))$ prover, P, where each query can be calculated by a time $\tilde{O}(S(n))$ algorithm, Q, and the results from P are verified by a time $\tilde{O}(S(n))$ verifier, V.

   Now we reduce the prover P to `SPACE TMSAT` so we can use that `SPACE TMSAT` $\in \mathbf{BPSIZE}[O(A(n))]$ to get a circuit for P.

   A length $\tilde{O}(S(n))$ query, q, to P can be converted into a length $\tilde{O}(S(n))$ input, q', for `SPACE TMSAT` by providing the algorithm for P and $\tilde{O}(S(n))$ 1s. In particular, for some $B(n) = \tilde{O}(n)$, proof input q' has length $B(S(n))$. We can also take $B(n) = \Omega(n)$. Call the circuit for `SPACE TMSAT` on length $|q'|$ inputs C_n. Since `SPACE TMSAT` $\in \mathbf{BPSIZE}[O(A(n))]$, we know C_n has size $O(A(B(S(n))))$.

3. Define our advice bit.

   Now an **MA** protocol can guess C_n, but we may not be able to compute how large C_n needs to be. The function $A(n)$ may be hard to compute. So we use advice.

   Let $l = \rho(n)$, $m = 2^l$ and $t = \mu(n)$ so that $n = m + t$. Observe that m and t can be inferred from n. Then let the advice bit be 1 if

   (a) Circuit C_m has size $S(m)2^t$.

   (b) For any natural t' less than t, circuit C_m does not have size $S(m)2^{t'}$.
   This condition allows us to use the smallest t possible for a given m.

Now define language W so that $x \in W$ for some x with $|x| = n$ if and only if the advice bit is 1 and for some $y \in \texttt{SPACE TMSAT}$ with $|y| = m$ we have $x = y1^{n-m}$.

4. Show W does not have small circuits.

   First we show that for every large enough l, for $m = 2^l$, there will be one t such that this advice bit is 1. To show this, we will show that for some t, C_m has size $S(m)2^t$. Then for the minimum such t, the advice bit will be one.

   For $t = m - 1$, by premise of the theorem, we have

   $$S(m)2^t = S(m)2^m/2 = \omega(A(B(S(m)))).$$

   This is eventually larger than C_m since C_m has size $O(A(B(S(m))))$. Then for large enough l with $m = 2^l$, there will be a smallest t so that C_m has size $S(m)2^t$ circuits, since it will for $t = m - 1$. The advice bit for such an $n = m + t$ must be 1. So infinitely often, the advice bit will be 1.

   When the advice bit is 1, the language W on length $n = m + t$ inputs is equal to L on length m inputs. Language L on length m inputs does not have circuits of size $S(m)/10$. See by choice of m that $2m > n$, and $S(n)$ is monotone, so $S(m)/10 > S(n/2)/10$. Thus infinitely often, W does not have size $S(n/2)/10$ circuits. Thus $W \notin \mathbf{BPSIZE}[o(S(n/2))]$.

5. Show W has an efficient **MA** protocol.

   If the advice bit is 0, this is trivially true. For $n = 2^l + t$ so that the advice bit is 1 and $m = 2^l$, either

   $t = 0$: Then C_m has size $S(m)$. Since $A(n) = \Omega(n)$ and $B(n) = \Omega(n)$, we know C_m has size $O(A(B(S(m))))$.

   $t \geq 1$: Then C_m has size $S(m)2^t$ but not $S(m)2^{t-1}$. Since C_m does have circuits of size $O(A(B(S(m))))$:

   $$S(m)2^t = O(A(B(S(m)))).$$

   In either case, an **MA** protocol can guess C_m with a circuit with size $O(A(B(S(m))))$.

   Then an **MA** protocol for $x = y1^{n-m}$ and an advice bit of 1 can verify if $y \in L$ by first guessing a circuit for C_m, then using it as the prover in the **PCP** system from Corollary 3.0.3. Unfortunately, C_m may fail with probability 1/3 on each query, so instead of running C_m directly on each query, we run it $O(\log(\log(S(m))))$ times and take a majority vote so it fails on any of the $\mathsf{polylog}(S(m))$ queries with probability at most 1/3.

   The **MA** verifier needs to calculate $\mathsf{polylog}(S(m))$ queries with Q, run C_m for $O(\log(\log(S)))$ many times on each of those queries, and run V on those results. Since C_m has size $O(A(B(S(m))))$, and Q and V run in time $\tilde{O}(S(m))$, calculating all query locations, running C_m on each of those locations, and V on those outputs takes time

   $$\begin{aligned} &\mathsf{polylog}(S(m))(\tilde{O}(S(m)) + \tilde{O}(A(B(S(m))))) + \tilde{O}(S(m)) \\ =&\tilde{O}(A(B(S(m))) + S(m)) \\ =&\tilde{O}(A(B(S(m)))). \end{aligned}$$

   The last equality comes from the fact $A(n) = \Omega(n)$ and $B(n) = \Omega(n)$. Finally, since A, B and S are non-decreasing and $m < n$, the **MA** verifier runs in time $\tilde{O}(A(B(S(n))))$.

   The **MA** protocol is complete and sound since the **PCP** is. That is, if $x \in L$ then the verifier accepts as long as all queries to the amplified C_m succeed, which they do with probability 2/3, and if $x \notin L$ then no prover will make the verifier accept with probability more than 1/3. Thus $W \in \mathbf{OMATIME}[\tilde{O}(A(B(S(n))))]/1$.

Therefore

$$W \in \mathbf{OMATIME}[\tilde{O}(A(B(S(n))))]/1 \setminus \mathbf{BPSIZE}[o(S(n/2))].$$

□

And in the special case where `SPACE TMSAT` has almost linear sized circuits, we get:

Corollary 3.3.2 (Bound if `SPACE TMSAT` has Size $n^{1+o(1)}$)**.** *If for some $g(n) = o(1)$ and some non-decreasing function $A(n) = n^{1+g(n)}$ we have `SPACE TMSAT` $\in$ $\mathbf{BPSIZE}[O(A(n))]$, then for any $k > 0$, there is an $f(n) = o(1)$ such that:*

$$\mathbf{OMATIME}[O(n^{k+f(n)})]/1 \not\subset \mathbf{BPSIZE}[O(n^k)].$$

Proof. We want to use Lemma 3.3.1 with $S(n) = n^k \log(n)$. The size upper bound on $S(n)$ is clear: $S(n) = o(2^n/n)$. We need to show $S(n)2^n = \omega(A(B(S(n))))$. Well for any $B(n) = \tilde{O}(n)$,

$$\begin{aligned} A(B(S(n))) =& B(n^k \log(n))^{1+g(n)} \\ =& \tilde{O}(n^{k+kg(n)}) \\ =& o(2^n) \\ S(n)2^n =& \omega(A(B(S(n)))). \end{aligned}$$

So by Lemma 3.3.1, for some $B(n) = \tilde{O}(n)$,

$$\mathbf{OMATIME}[\tilde{O}(A(B(S(n))))]/1 \not\subset \mathbf{BPSIZE}[o(S(n/2))].$$

See that for some $f(n) = o(1)$,

$$\tilde{O}(A(B(S(n)))) = \tilde{O}(n^{k+kg(n)}) = O(n^{k+f(n)}).$$

Similarly $n^k = o(S(n/2))$, so we also have

$$\mathbf{OMATIME}[O(n^{k+f(n)})]/1 \not\subset \mathbf{BPSIZE}[O(n^k)].$$

□

3.4 `SPACE TMSAT` $\in \mathbf{BPSIZE}[n^{a+o(1)}] \setminus \mathbf{BPSIZE}[n^{a-o(1)}]$ for $a > 1$

This is the "bad" case, where we can't prove the result for every constant k, only for $k < a$. This is the most complicated case, requiring us to both pad the input to get the correct problem difficulty, and use advice to get the size of the circuits for the prover.

Lemma 3.4.1 (Bound from `SPACE TMSAT` as Hard Problem)**.** *If for some non-decreasing $A(n)$ we have `SPACE TMSAT` $\in \mathbf{BPSIZE}[O(A(n))] \setminus \mathbf{BPSIZE}[o(A(n))]$, then there is some non-decreasing $B(n) = \tilde{\Theta}(n)$ and $D(n) = O(n)$ such that if for some time constructible, non-decreasing $S(n)$ with $S(2n) = o(A(n))$ and $S(n)2^n = \omega(A(B(n)))$, then we have:*

$$\mathbf{OMATIME}\left[\tilde{O}\left(S(n)\frac{A(B(D(n)))}{A(D(n)/2)}\right)\right]/1 \not\subset \mathbf{BPSIZE}[o(S(n/4))].$$

We will use this problem with $A(n)$ being close to n^k for some constant k so the $\frac{A(B(D(n)))}{A(D(n)/2)}$ term is $n^{o(1)}$. Before we give the proof, we explain the parameters in this result. In this problem, you can think of $D(n)$ as being similar to $A^{-1}(S(n))$, though to simplify the analysis, we use a more trivial bound of $D(n) = O(n)$. The $B(n)$ comes from the prover space and log of the proof length of Corollary 3.0.3. Then this fraction term in the **MA** verifier time, loosely, accounts for the increase in circuit size for `SPACE TMSAT` on length n inputs versus length $B(n)$ inputs.

If `SPACE TMSAT` $\in \mathbf{P/poly}$, the difference between the size of circuits for `SPACE TMSAT` on length n inputs and length $B(n)$ inputs will be small (at least for n where `SPACE TMSAT` requires circuits with size near the polynomial that upper bounds the size of `SPACE TMSAT`). But if `SPACE TMSAT` requires larger than polynomial sized circuits, then the difference in circuit size between length n inputs and length $B(n)$ may become large. This is why we only use this argument in the case that `SPACE TMSAT` $\in \mathbf{P/poly}$, and we use a different **PSPACE** complete language with a interactive different protocol when `SPACE TMSAT` $\notin \mathbf{P/poly}$.

So the idea is to solve `SPACE TMSAT` on a padded version of the input using our **PCP**. So we need the advice to tell us three things:

1. Some m so that `SPACE TMSAT` on length m inputs requires circuits of size $S(n/4)$.
2. Further, we need `SPACE TMSAT` on length m inputs to require circuits of size near $A(m/2)$. This keeps the prover from requiring circuits too much larger than `SPACE TMSAT` on length m inputs does.
3. How big the circuit for the prover in Corollary 3.0.3 needs to be.

Similar to the previous cases, this advice will come implicitly from the input length, and the single advice bit will be 1 if and only if the input length encodes valid advice. Since the input length needs to encode three things, the problem input length, the amount of padding, and the prover circuit size, we need to decompose the input length into three components this time, not just two.

Proof. If $S(n) = O(n)$, then we use Corollary 2.1.8. Otherwise, we want to solve a smaller instance of `SPACE TMSAT` that requires circuits of size $S(n/4)$, and we also need advice to tell us the size of circuits needed to prove `SPACE TMSAT`. The advice for this will come implicitly from the input length.

For input x of length n, (using ρ and μ from Definition 3.1.1) let $l = \rho(n)$, $l' = \rho(\mu(n))$, and $t = \mu(\mu(n))$ so that $n = 2^l + 2^{l'} + t$. We want to solve `SPACE TMSAT` on length $2^{l'}$ inputs, so we let $m := 2^{l'}$. Let $D(n) := m$. Then we can write n as $n = 2^l + m + t$ and our language will solve length m inputs for `SPACE TMSAT` using prover circuits of size $S(2^l)2^t$. Then the advice bit will only be 1 only when this advice is good.

So then m is the input length to `SPACE TMSAT` we want to solve, 2^l is how much padding is needed to make length m problems the right difficulty, and $S(2^l)2^t$ is the size of the circuits needed for our **PCP** prover.

The proof proceeds in 4 steps.

1. Define circuits C_m that prove `SPACE TMSAT` for length m inputs using our **PCP** and our theorem assumptions on circuits for `SPACE TMSAT`.
2. Define when the advice bit should be 1.
3. Show infinitely often the advice bit is 1 and $W \notin \mathbf{BPSIZE}[o(S(n/4))]$.
4. Show that
$$W \in \mathbf{OMATIME}\left[\tilde{O}\left(S(n)\frac{A(B(D(n)))}{A(D(n)/2)}\right)\right]/1.$$

Now following this outline:

1. Define circuits C_m that prove `SPACE TMSAT` for length m inputs.

   We know `SPACE TMSAT` on length m inputs has a **PCP** system with decision complexity $\tilde{O}(m)$, log of proof length $\tilde{O}(m)$, and prover space $\tilde{O}(m)$. Then for some strictly increasing $B(m) = \tilde{O}(m)$, the prover for `SPACE TMSAT` on length m inputs can be reduced to a circuit for `SPACE TMSAT` with length $B(m)$ inputs. Then `SPACE TMSAT` on length $B(m)$ inputs has a circuit, C_m, of size at most $O(A(B(m)))$. We can also take $B(m) = \Omega(m)$ so that $B(m) = \tilde{\Theta}(m)$.

2. Define when the advice bit should be 1.

   Since `SPACE TMSAT` $\notin \mathbf{BPSIZE}[o(A(n))]$, for some $c_1 > 0$, for some infinite set, U', for all $n' \in U'$, language `SPACE TMSAT` on length n' inputs does not have circuits with size $c_1 A(n')$.

   Let the advice bit be 1 if and only if each of the following hold:

   (a) `SPACE TMSAT` on length m inputs does not have circuits with size at most $c_1 A(m/2)$.
   This restricts us to m where the circuits for `SPACE TMSAT` require size near our upper bound. This limits how much bigger C_m needs to be than the circuits for `SPACE TMSAT` on length m inputs.

   (b) `SPACE TMSAT` on length m inputs does not have a circuit with size $S(2^{l-1})$. Note $S(2^{l-1}) \geq S(n/4)$.

   (c) `SPACE TMSAT` on length m inputs does have a circuit with size $S(2^l)$.

   (d) Circuit C_m has size at most $S(2^l)2^t$.

   (e) Either $t = 0$, or C_m has size at least $S(2^l)2^{t-1}$.

Then $x \in W$ for some x with $|x| = n$ if and only the advice bit is 1 and for some $y \in$ `SPACE TMSAT` with $|y| = m$ we have $x = y1^{n-m}$.

3. Now we will argue that infinitely often the advice bit is 1 and W does not have circuits with size $S(n/4)$. We do this in a few steps:

   - First restrict our focus to m large enough and where `SPACE TMSAT` on length m inputs has size near $A(m/2)$. This will be the set of input lengths, U, which is the power of 2 integers m where m is large enough *and* `SPACE TMSAT` does not have circuits of size $S(m)$ or $c_1A(m/2)$. Now we show such a U exists and is infinite.

     Recall that U' is the set of n' such that language `SPACE TMSAT` on length n' inputs does not have circuits with size $c_1A(n')$. Since, by theorem premise, $S(2n) = o(A(n))$, for some n_2, for all $n' > n_2 : S(2n') < c_1A(n')$.

     Since, by theorem premise, $S(n)2^n = \omega(A(B(n)))$, and C_n has size at most $O(A(B(n)))$, we have $|C_n| = o(S(n)2^n)$. So for some n_3, for all $n' > n_3$, circuit $C_{n'}$ has size at most $S(n')2^{n'-1}$.

     Take U^* to be the $n' \in U'$ larger than $\max\{n_1, n_2, n_3\}$. See that U^* is still an infinite set. For each length $n' \in U^*$, we will find a length $n > n'$ so the advice bit is 1.

     For $n' \in U^*$, let $m = 2^{l'}$ be the smallest power of 2 greater than n'. That is, $m > n'$, but $2n' \geq m$. By choice of $U^* \subseteq U'$, language `SPACE TMSAT` on length n' inputs does not have circuits of size $c_1A(n')$. Recall that the min circuit length for `SPACE TMSAT` is monotone (see Lemma 3.0.2), so since $m > n'$, language `SPACE TMSAT` on length m inputs does not have circuits of size $c_1A(n')$.

     Since A is monotone and $m \leq 2n'$, we know $c_1A(m/2) \leq c_1A(n')$. Since $n' > n_2$, we know $S(2n') < c_1A(n')$. Since S is monotone and $m \leq 2n'$, we have $S(m) \leq S(2n')$. So we know $S(m) \leq c_1A(n')$. Then since `SPACE TMSAT` on length m inputs does not have circuits of size $c_1A(n')$, we also have `SPACE TMSAT` on length m inputs does not have circuits of size $S(m)$. Similarly, since $n' \geq m/2$ and A is monotone, `SPACE TMSAT` on length m inputs does not have circuits of size $c_1A(m/2)$.

     Let U be the set of m from each $n' \in U^*$. See that U is an infinite set since for each $n' \in U^*$, there is an $m \in U$ greater than n', and U^* is an infinite set. Then for $m \in U$, language `SPACE TMSAT` on length m inputs does not have circuits of size $S(m)$ or $c_1A(m/2)$ and $m > \max\{n_1, n_2, n_3\}$.

   - For each $m \in U$, find appropriate l and t (that is, show they exist as previously defined). Here, think of m as being fixed, and we are looking for l and t to define the corresponding n.

     Take the smallest l so that `SPACE TMSAT` on length m inputs does have a circuit of size $S(2^l)$. Note that $l > l' = \log(m)$, since `SPACE TMSAT` on length m inputs does not have circuits with size $S(m)$.

     Let t be the smallest t such that C_m has size $S(2^l)2^t$. Since $m > n_3$, we know C_m has size at most $S(m)2^{m-1} < S(2^l)2^{m-1}$. Thus $t \leq m - 1 < m$.

     Finally, see that since m is a power of two, 2^l is a power of two bigger than m and t is less than m, we have that these are the proper l, m, t for $n = 2^l + m + t$.

   - Now for $n = 2^l + m + t$, we show the advice bit is 1 and language `SPACE TMSAT` on length n inputs does not have circuits with size $S(n/4)$.

     First, see that $t < m$, so $m + t < 2^{l'+1}$. As noted before, $l' < l$, so $2^{l'+1} \leq 2^l$. Thus $2^l > m + t$ and $\rho(n) = l$. Similarly $l' = \rho(\mu(n))$, $m = 2^{l'}$, and $t = \mu(\mu(n))$. Then

     (a) By choice of U, language `SPACE TMSAT` on length m inputs does not have circuits of size $c_1A(m/2)$.

     (b,c) `SPACE TMSAT` on length m inputs does not have a circuit with size $S(2^{l-1})$, since we chose the smallest l so that `SPACE TMSAT` on length m inputs has a circuit with size $S(2^l)$.

     (d) By choice of t, circuit C_m has size $S(2^l)2^t$.

     (e) Specifically, t is the smallest such that C_m has size $S(2^l)2^t$. So either $t = 0$, or C_m does not have size $S(2^l)2^{t-1}$.

So for that n, the advice bit is 1.

Since for every $m \in U$ for some $n > m$ the advice bit is 1, and since U is an infinite set, the advice bit is one infinitely often. For input lengths where the advice bit is 1, `SPACE TMSAT` does not have circuits of size $S(2^{l-1}) \geq S(n/4)$. So `SPACE TMSAT` does not have circuits of size $S(n/4)$ infinitely often. Therefore

$$W \notin \mathbf{BPSIZE}[o(S(n/4))].$$

4. Show that

$$W \in \mathbf{OMATIME}\left[\tilde{O}\left(S(n)\frac{A(B(D(n)))}{A(D(n)/2)}\right)\right]/1.$$

If the advice bit is 0, this is trivial. Otherwise, assume for n the advice bit is 1.

When the advice bit is 1, we know C_m has size at most $S(2^l)2^t$ and either

$t = 0$: Then C_m has size $S(2^l) = O(S(n))$ since S is monotone and $2^l < n$.

$t \geq 1$: Then C_m does not have size $S(2^l)2^{t-1}$ by choice of t. Circuit C_m has size $A(B(m))$. Thus

$$S(2^l)2^{t-1} < A(B(m)).$$

Further, `SPACE TMSAT` on length m inputs does not have circuits of size $c_1A(m/2)$ since the advice bit is 1, but it does have circuits of size $S(2^l)$. We emphasize that this is the circuit size of `SPACE TMSAT`, not C_m. We specifically chose l and m so this would be true, see Item 2a and Item 2c. Thus

$$c_1A(m/2) < S(2^l).$$

Together

$$\begin{aligned} S(2^l)2^{t-1} &< A(B(m)) \\ c_1A(m/2)2^{t-1} &< A(B(m)) \\ 2^t &< \frac{2}{c_1}\frac{A(B(m))}{A(m/2)}. \end{aligned}$$

Thus C_m has size

$$S(2^l)2^t = O\left(S(n)\frac{A(B(D(n)))}{A(D(n)/2)}\right).$$

Similar to the other results, each query needs to use $O(\log(\log(m)))$ queries to the circuit C_m to simulate one call to the prover with high probability. Thus, the verifier for `SPACE TMSAT` can be simulated in time $\tilde{O}(m)$, and the $\mathsf{polylog}(m)$ queries to the prover can be simulated in time

$$\mathsf{poly}(\log(m))S(2^l)2^t = \tilde{O}\left(S(n)\frac{A(B(D(n)))}{A(D(n)/2)}\right).$$

This gives a total **MA** time of $\tilde{O}\left(S(n)\frac{A(B(D(n)))}{A(D(n)/2)}\right)$ for W. Thus

$$W \in \mathbf{OMATIME}\left[\tilde{O}\left(S(n)\frac{A(B(D(n)))}{A(D(n)/2)}\right)\right]/1.$$

Therefore

$$W \in \mathbf{OMATIME}\left[\tilde{O}\left(S(n)\frac{A(B(D(n)))}{A(D(n)/2)}\right)\right]/1 \setminus \mathbf{BPSIZE}[o(S(n/4))].$$

□

Now for the special case where `SPACE TMSAT` almost has some fixed polynomial size.

Corollary 3.4.2 (Bound if `SPACE TMSAT` has size $n^{a+o(1)}$). *Suppose for some function $h(n)$ with $|h(n)| = o(1)$ and for some constant $a > 1$, for some function $A(n)$ we have $A(n) = n^{a+h(n)}$. Then if $A(n)$ is non-decreasing and we have* `SPACE TMSAT` $\in \mathbf{BPSIZE}[O(A(n))] \setminus \mathbf{BPSIZE}[o(A(n))]$, *then for any $k < a$, for some $f(n) = o(1)$,*

$$\mathbf{OMATIME}[O(n^{k+f(n)})]/1 \not\subset \mathbf{BPSIZE}[O(n^k)].$$

Proof. If $k < 1$, we use Corollary 2.1.8. Otherwise, let $S(n) = n^k \log(n)$. Since $k < a$, we have $S(2n) = o(A(n))$.

To apply Lemma 3.4.1, we need to show that for any $B(n) = \tilde{\Theta}(n)$, we have $S(n)2^n = \omega(A(B(n)))$. But since $A(n)$ and $B(n)$ are both polynomials, they are smaller than 2^n. That is

$$\begin{aligned} A(B(n)) =& o(B(n)^{k+1/2}) \\ =& o(2^n) \\ =& o(S(n)2^n). \end{aligned}$$

This is equivalent to $S(n)2^n = \omega(A(B(n)))$.

Now we can apply Lemma 3.4.1 to get a language W such that

$$W \in \mathbf{OMATIME}\left[\tilde{O}\left(n^k \log(n)\frac{A(B(D(n)))}{A(D(n)/2)}\right)\right]/1 \setminus \mathbf{BPSIZE}[o(n^k \log(n))].$$

Now let's simplify this a bit. Since $W \notin \mathbf{BPSIZE}[o(n^k \log(n))]$ and $n^k = o(n^k \log(n))$, we have $W \notin \mathbf{BPSIZE}[O(n^k)]$.

Now we want to bound that fraction:

$$\frac{A(B(D(n)))}{A(D(n)/2)} = \frac{(B(D(n)))^{a+h(B(D(n)))}}{(D(n)/2)^{a+h(D(n)/2)}}.$$

We start by letting $D(n) = m$ and bounding this in terms of m first. Then

$$\begin{aligned} \frac{A(B(D(n)))}{A(D(n)/2)} =& \frac{(B(m))^{a+h(B(m))}}{(m/2)^{a+h(m/2)}} \\ =& \tilde{O}\left(\frac{m^{a+h(B(m))}}{m^{a+h(m/2)}}\right) \\ =& \tilde{O}\left(m^{h(B(m))-h(m/2)}\right). \end{aligned}$$

Since $B(m) = \omega(1)$, and $|h(m)| = o(1)$, we know $|h(B(m))| = o(1)$. So for some $h^*(m)$ with $|h^*(m)| = o(1)$, we have

$$\begin{aligned} \frac{A(B(D(n)))}{A(D(n)/2)} =& O(m^{h^*(m)}) \\ =& O(D(n)^{h^*(D(n))}). \end{aligned}$$

Note that since A and B are both non-decreasing, this fraction is at least 1. So in particular $h^*(n) \geq 0$, and $h^*(n) = o(1)$.

Now using Lemma 2.1.9, since $D(n) = O(n)$, for some $h'(n) = o(1)$, we have

$$\frac{A(B(D(n)))}{A(D(n)/2)} = O(n^{h'(n)}).$$

Thus for some $f(n) = o(1)$, we have

$$\begin{aligned} \tilde{O}\left(n^k \log(n)\frac{A(B(D(n)))}{A(D(n)/2)}\right) =& \tilde{O}\left(n^k n^{h'(n)}\right) \\ =& O(n^{k+f(n)}). \end{aligned}$$

So $W \in \mathbf{BPSIZE}[O(n^{k+f(n)})]$. Thus we conclude:

$$\mathbf{OMATIME}[O(n^{k+f(n)})]/1 \not\subset \mathbf{BPSIZE}[O(n^k)].$$

□

3.5 Altogether

Altogether, these three cases imply Theorem 1.1.1 and the following generalization.

Theorem 1.3.1 (**OMATIME** Lower Bound Against **BPTIME**)**.** *There exists constant $a > 1$, such that for all $k < a$, for some $f(n) = o(1)$,*

$$\mathbf{OMATIME}[O(n^{k+f(n)})]/1 \not\subset \mathbf{BPTIME}[O(n^k)]/O(n^k).$$

Proof. First, we will find the best polynomial approximation of the circuit complexity of `SPACE TMSAT`. So define set

$$S = \{a \in \mathbb{R} : \texttt{SPACE TMSAT} \in \mathbf{BPSIZE}[O(n^a)]\}.$$

If $S = \emptyset$, then there is no constant a such that `SPACE TMSAT` $\in \mathbf{BPSIZE}[O(n^a)]$. Then `SPACE TMSAT` $\notin$ **P/poly**, so we use Corollary 3.2.3. Then Corollary 3.2.3 gives: for any $k > 0$, and some $f(n) = o(1)$:

$$\mathbf{OMATIME}[O(n^{k+f(n)})]/1 \not\subset \mathbf{BPSIZE}[O(n^k)].$$

So suppose $S \neq \emptyset$. Now see that `SPACE TMSAT` requires circuits of size $\Omega(n)$ since we can reduce parity to it and parity requires circuits of size $\Omega(n)$ (see Lemma 2.1.7). Thus for any $a < 1$, we know that `SPACE TMSAT` $\notin \mathbf{BPSIZE}[O(n^a)]$, that is $a \notin S$. So 1 is a lower bound for S.

Then the set S is nonempty and has a lower bound. So S has an infimum, a, so that for any constant $\epsilon > 0$, we have `SPACE TMSAT` $\in \mathbf{BPSIZE}[O(n^{a+\epsilon})]$, but `SPACE TMSAT` $\notin \mathbf{BPSIZE}[O(n^{a-\epsilon})]$.

Before we use Corollary 3.3.2 and Corollary 3.4.2, we need to find an $A(n)$ such that for some $h(n)$, we have $A(n) = n^{a+h(n)}$ and

1. $A(n)$ is non-decreasing.
2. `SPACE TMSAT` $\in \mathbf{BPSIZE}[O(A(n))] \setminus \mathbf{BPSIZE}[o(A(n))]$.
3. $|h(n)| = o(1)$.

Let $A'(n)$ be the minimum circuit size of `SPACE TMSAT` on length n inputs. One might hope $A'(n)$ would work for $A(n)$, but the difficulty of `SPACE TMSAT` may not increase smoothly. It may remain near linear for many consecutive n, and only occasionally increase near n^a. So instead, we want a smoother function for $A(n)$ that never drops too far below n^a, but infinitely often is equal to $A'(n)$.

So the idea is just to have $A(n)$ be the maximum of $A'(n)$ and some polynomial just smaller than n^a, say $n^{a-\epsilon}$. Then $A(n)$ won't get far away from n^a if $A'(n)$ becomes small. But we can't use a constant ϵ, or we could get $|h(n)| = \Omega(1)$. So instead, we make ϵ smaller each time $A'(n)$ is larger than $n^{a-\epsilon}$.

Define $m(n)$ so that $m(0) = 0$ and

$$m(n+1) = \begin{cases} m(n)+1 & A'(n) \geq n^{a-2^{-m(n)}} \\ m(n) & \text{otherwise} \end{cases}.$$

Then $\epsilon(n) = 2^{-m(n)}$.

Now we define $A(n) = \max\{A'(n), n^{a-\epsilon(n)}\}$. Then for $h(n) = \frac{\log(A(n))}{\log(n)} - a$, we have $A(n) = n^{a+h(n)}$. Now we show the three conditions.

1. $A(n)$ is non-decreasing.

   $A(n)$ is the maximum of two non-decreasing sequences: $A'(n)$ and $n^{a-\epsilon(n)}$, so is also non-decreasing.

2. `SPACE TMSAT` $\in \mathbf{BPSIZE}[O(A(n))] \setminus \mathbf{BPSIZE}[o(A(n))]$.

   By choice of $A(n)$, for all n, $A(n) \geq A'(n)$, the minimum circuit size of `SPACE TMSAT`, so `SPACE TMSAT` $\in \mathbf{BPSIZE}[O(A(n))]$.

   Now we will argue that infinitely often, $A(n) = A'(n)$. Otherwise, for some n', for all $n \geq n'$, $A'(n) < n^{a-\epsilon(n)}$. If this were true, then for any $n \geq n'$, $m(n) = m(n')$ since for none of these n will $m(n)$ increase. Thus for all $n > n'$, $A'(n) < n^{a-\epsilon(n')}$. Then `SPACE TMSAT` $\in \mathbf{BPSIZE}[O(n^{a-\epsilon(n')})]$. But since $\epsilon(n') > 0$, by choice of a, this cannot happen. Contradiction. So infinitely often, $A(n) = A'(n)$.

   Since infinitely often, `SPACE TMSAT` requires circuits of size $A(n)$, `SPACE TMSAT` $\notin \mathbf{BPSIZE}[o(A(n))]$.

3. $|h(n)| = o(1)$.

   From the last section, infinitely often, $A'(n) \geq n^{a-\epsilon(n)}$, so $m(n) \to \infty$, and for $h_1(n) = \epsilon(n) = o(1)$, we have a lower bound on $A(n)$ of $A(n) \geq n^{a-h_1(n)}$.

   Let $h_2(n) = \max\{0, \frac{\log(A'(n))}{\log(n)} - a\}$. See that $n^{a+h_2(n)}$ is always at least $n^{a-\epsilon(n)}$ and $A'(n)$, so $A(n) \leq n^{a+h_2(n)}$. Next we show that $h_2(n) = o(1)$.

   Suppose otherwise. Then for some $c > 0$, for infinitely many n, we have $h_2(n) > c$. But for such n, we have $h_2(n) = \frac{\log(A'(n))}{\log(n)} - a$, so
$$A'(n) = n^{a+h_2(n)} > n^{a+c}.$$
   Then infinitely often, `SPACE TMSAT` does not have size n^{a+c} circuits. This means `SPACE TMSAT` $\notin$ **BPSIZE**$[o(n^{a+c})]$. Specifically, for $c/2 > 0$, we have `SPACE TMSAT` $\notin$ **BPSIZE**$[O(n^{a+c/2})]$. But choice of a, this cannot happen. Contradiction. So $h_2(n) = o(1)$.

   Thus
$$\begin{aligned}
n^{a-h_1(n)} \leq && A(n) \leq && n^{a+h_2(n)} \\
a - h_1(n) \leq && \frac{\log(A(n))}{\log(n)} \leq && a + h_2(n) \\
-h_1(n) \leq && h(n) \leq && h_2(n) \\
&& |h(n)| \leq && \max\{h_1(n), h_2(n)\} \\
&& = && o(1).
\end{aligned}$$

If $a = 1$, we use Corollary 3.3.2. See that $|h(n)| = o(1)$ and `SPACE TMSAT` $\in$ **BPSIZE**$[O(n^{1+|h(n)|})]$. Thus for any k, for some $f(n) = o(1)$, we have
$$\mathbf{OMATIME}[O(n^{k+f(n)})]/1 \not\subset \mathbf{BPSIZE}[O(n^k)].$$

If $a > 1$, we use Corollary 3.4.2. See that A was specifically constructed to satisfy the theorem requirements. Then for any $k < a$, for some $f(n) = o(1)$, we have
$$\mathbf{OMATIME}[O(n^{k+f(n)})]/1 \not\subset \mathbf{BPSIZE}[O(n^k)].$$

Now to turn this into a statement about about **BPTIME**, we use Lemma 2.2.5. □

Next we prove a generalization of Theorem 1.1.2. We use the case of $a > 1$ in the proof of Theorem 1.3.1 and apply Lemma 3.3.1 similar to Corollary 3.3.2.

Theorem 3.5.1 (**MA** Lower Bound for Small a)**.** *If the a from Theorem 1.3.1 is finite, then for all $k > 0$, for some $f(n) = o(1)$,*
$$\mathbf{OMATIME}[O(n^{ak+f(n)})]/1 \not\subset \mathbf{BPSIZE}[O(n^k)].$$

Proof. We want to use Lemma 3.3.1 with $S(n) = n^k \log(n)$, $A(n) = n^{a+h(n)}$ from the proof of Theorem 1.3.1, and $B(n) = \tilde{\Theta}(n)$ from Lemma 3.3.1.

The size upper bound on $S(n)$ is clear: $S(n) = o(2^n/n)$. We need to show $S(n)2^n = \omega(A(B(S(n))))$. Well for any $B(n) = \tilde{O}(n)$,
$$\begin{aligned}
A(B(S(n))) =& B(n^k \log(n))^{a+h(n)} \\
=& \tilde{O}(n^{ak+kh(n)}) \\
=& o(2^n) \\
S(n)2^n =& \omega(A(B(S(n)))).
\end{aligned}$$

So by Lemma 3.3.1, for some $B(n) = \tilde{O}(n)$,
$$\mathbf{OMATIME}[\tilde{O}(A(B(S(n))))]/1 \not\subset \mathbf{BPSIZE}[o(S(n/2))].$$

See that for some $f(n) = o(1)$,

$$\tilde{O}(A(B(S(n)))) = \tilde{O}(n^{ak+kh(n)}) = O(n^{ak+f(n)}).$$

Similarly $n^k = o(S(n/2))$, so we also have

$$\mathbf{OMATIME}[O(n^{ak+f(n)})]/1 \not\subset \mathbf{BPSIZE}[O(n^k)].$$

$\square$

4 Extrapolatable PCPs

To get our efficient **PCP**s, we use a technique called **PCP** composition to compose a robust outer **PCP** (**rPCP**) with an inner **PCP** of proximity (**PCPP**). See Section 2.3 for more information. We introduce extrapolatable **PCP**s (**ePCP**s) as an intermediate **PCP** in constructing an efficient **rPCP** and **PCPP**. We will later use **rPCP**s in **PCP** composition to reduce the number of queries. Before defining an **ePCP**, we start by introducing several useful properties about extrapolatability.

We emphasize that many prior PCP constructions were extrapolatable, they only lacked analysis to show that they are. Similarly, our PCP construction is not new. Indeed, it is a standard instantiation of the BFL PCP [BFL90] made robust via an aggregation through curves argument [Aro+98] composed with itself. The composition soundness comes from a PCP of proximity type argument [Ben+04]. The main innovation here is that the query time can be significantly less than previously shown leading to a more time efficient verifier and a more space efficient prover. **ePCP** is just a convenient primitive for analysis, and in particular the BFL **PCP** is an **ePCP**.

4.1 Extrapolatable Functions

We introduce extrapolatable functions as a tool to efficiently compute low degree extrapolations. For any $Q : [q] \to \mathbb{F}^n$, we say its low degree extrapolation is the unique degree $q-1$ function that agrees with Q on its first q values. Any Q computable in time $n \,\mathsf{polylog}(|\mathbb{F}|)$, we can compute the extrapolation of Q in time $qn \,\mathsf{polylog}(|\mathbb{F}|)$. But we want to compute the extrapolation of Q in time $(q+n)\,\mathsf{polylog}(|\mathbb{F}|)$. Recall the definition of an extrapolatable function.

Definition 1.2.1 (Extrapolatability)**.** *For any $n, q, t > 0$, and field $\mathbb{F}$, we call $Q : [q] \to \mathbb{F}^n$ "t extrapolatable" (or time t extrapolatable) if there is a time t algorithm taking any $v \in \mathbb{F}^q$, that outputs*

$$\sum_{i\in[q]} v_i Q(i).$$

Remark (Embedding in $\mathbb{F}$)**.** *Here, we embed $[q]$ in $\mathbb{F}$ in any convenient way. For our results, it does not matter as long as the embedding is efficiently computable.*

We will use Q where $t \le (q+n)\,\mathsf{polylog}(|\mathbb{F}|)$.

Various basic combinations of extrapolatable functions give extrapolatable functions. The function that outputs one extrapolatable function for its first m inputs, and then a second extrapolatable function for its last m' inputs is also extrapolatable.

Lemma 4.1.1 (Extrapolatability Combination 1)**.** *For integers $n, q, q', t, t' > 0$, and field $\mathbb{F}$, if $p : [q] \to \mathbb{F}^n$ is t extrapolatable, and $p' : [q'] \to \mathbb{F}^n$ is t' extrapolatable, then $g : [q+q'] \to \mathbb{F}^n$ is $O(t + t' + n\log(\mathbb{F}))$ extrapolatable where*

$$g(i) = \begin{cases} p(i) & i \le q \\ p'(i-q) & i > q \end{cases}.$$

Proof. To prove $g(i)$ is extrapolatable, we need an algorithm that takes $v \in \mathbb{F}^{q+q'}$ and outputs

$$\sum_{i\in[q+q']} v_i g(i).$$

We can write this sum as

$$\begin{aligned}\sum_{i\in[q+q']} v_i g(i) &= \sum_{i\in[q]} v_i g(i) + \sum_{i\in[q']} v_{q+i} g(q+i) \\ &= \sum_{i\in[q]} v_i p(i) + \sum_{i\in[q']} v_{q+i} p'(i).\end{aligned}$$

Then use extrapolatability of p to calculate

$$\sum_{i\in[q]} v_i p(i)$$

in time t, and use extrapolatability of p' to calculate

$$\sum_{i\in[q']} v_{q+i} p'(i)$$

in time t'. Then their sum is the answer, and addition takes time $O(n \log(\mathbb{F}))$. $\square$

Similarly, a function that outputs pairs of values from extrapolatable functions is extrapolatable.

Lemma 4.1.2 (Extrapolatability Combination 2)**.** *For integers $n, n', q, t, t' > 0$, and field $\mathbb{F}$, if $p : [q] \to \mathbb{F}^n$ is t extrapolatable, and $p' : [q] \to \mathbb{F}^{n'}$ is t' extrapolatable, then $g : [q] \to \mathbb{F}^{n+n'}$ is $O(t + t')$ extrapolatable where*

$$g(i) = (p(i), p'(i)).$$

Proof. Given $v \in \mathbb{F}^q$, we need to calculate

$$\begin{aligned}\sum_{i\in[q]} v_i g(i) &= \sum_{i\in[q]} v_i (p(i), p'(i)) \\ &= \left(\sum_{i\in[q]} v_i p(i), \sum_{i\in[q]} v_i p'(i) \right).\end{aligned}$$

We use extrapolatability of p to calculate

$$\sum_{i\in[q]} v_i p(i)$$

in time t, then use extrapolatability of p' to calculate

$$\sum_{i\in[q]} v_i p'(i)$$

in time t'. Then concatenate the results. $\square$

As an example of extrapolatable functions, see that any function outputting an arithmetic progression is extrapolatable.

Lemma 4.1.3 (Arithmetic Progressions are Extrapolatable)**.** *For integers $n, q > 0$, and field $\mathbb{F}$, for any $x \in \mathbb{F}^n$ and $y \in \mathbb{F}^n$, the function $f : [q] \to \mathbb{F}^n$ defined by*

$$f(i) = x + iy$$

is time $O((n + q)\,\mathsf{polylog}(|\mathbb{F}|))$ extrapolatable.

Proof. Given $v \in \mathbb{F}^q$, we need to calculate

$$\begin{aligned}\sum_{i\in[q]} v_i f(i) &= \sum_{i\in[q]} v_i(x+iy)\\ &= \left(\sum_{i\in[q]} v_i\right) x + \left(\sum_{i\in[q]} v_i i\right) y.\end{aligned}$$

Then one can calculate $\alpha = \sum_{i\in[q]} v_i$ using just q field additions, which takes time $O(q\log(|\mathbb{F}|))$. Similarly, one can calculate $\beta = \sum_{i\in[q]} v_i i$ using q multiplications and additions, which takes time $O(q\,\mathsf{polylog}(|\mathbb{F}|))$.

Now we need to calculate $\alpha x + \beta y$. Since $x \in \mathbb{F}^n$, it only n field operations to multiply x by α, so αx only takes time $O(n\,\mathsf{polylog}(|\mathbb{F}|))$ to calculate. Similar for βy and the sum of of αx with βy.

So altogether, this algorithm only takes time $O((q+n)\,\mathsf{polylog}(|\mathbb{F}|)$ to compute $\sum_{i\in[q]} v_i f(i)$. $\square$

Now we show that we can efficiently extrapolate (compute the low degree extrapolation of) an extrapolatable function.

Lemma 4.1.4 (Efficient Polynomials From Extrapolatability)**.** *For any $n, q, t > 0$, field $\mathbb{F}$ where $|\mathbb{F}| > q$, and t extrapolatable $Q : [q] \to \mathbb{F}^n$, let g be the degree $q-1$ polynomial such that for all $i \in [q], g(i) = Q(i)$. Then there is a time*

$$O(t + q\,\mathsf{polylog}(|\mathbb{F}|))$$

algorithm computing taking any $x \in \mathbb{F}$ and outputting $g(x)$,

Proof. We use Lagrange interpolation. For a given q, and $i \in [q]$, the ith Lagrange basis polynomial is:

$$l_i^q(x) = \prod_{j\in[q]\setminus\{i\}} \frac{x-j}{i-j}.$$

This is the degree $q-1$ polynomial that is 1 at $x = i$, but 0 for all other $x \in [q] \setminus \{i\}$.

Then we can easily write our desired g in terms of the Lagrange basis polynomials:

$$g(x) = \sum_{i\in[q]} l_i^q(x) Q(i).$$

A naive, straightforward evaluation of this sum takes time $O(nq\,\mathsf{polylog}(|\mathbb{F}|))$. But since Q is t extrapolatable, if we can calculate $l_1^q(x), \dots, l_q^q(x)$, we can use these to calculate g in time t.

For a fixed x, and i, we can define

$$\begin{aligned}\alpha_i &= \prod_{j\in[i-1]} (x-j)\\ \alpha_i' &= \prod_{j\in[q]\setminus[i]} (x-j)\\ \beta_i &= \prod_{j\in[i-1]} j\\ \beta_i' &= \prod_{j\in[q-i]} (-j)\end{aligned}$$

so that

$$\begin{aligned}l_i^q(x) &= \prod_{j\in[q]\setminus\{i\}} \frac{x-j}{i-j}\\ &= \frac{\alpha_i\alpha_i'}{\beta_i\beta_i'}.\end{aligned}$$

Each one of these sequences $(\alpha, \alpha', \beta, \beta')$ can be entirely computed in time $O(q\,\mathsf{polylog}(|\mathbb{F}|))$. For instance, see that for $i < q$, $\alpha_{i+1} = (x-i)\alpha_i$, which can be computed with two field operations. So all q of the α_i can be computed in time $O(q\,\mathsf{polylog}(|\mathbb{F}|))$. Similarly for α', β, and β'.

Now given each of α, α', β, and β' have already been calculated, we can calculate $l_i^q(x)$ in four field operations. Thus, every $l_1^q(x), \dots, l_q^q(x)$ can be calculated in time $O(q\,\mathsf{polylog}(|\mathbb{F}|))$.

Finally, since Q is time t extrapolatable, we can calculate $g(x)$ in time t, giving a total time of $O(t + q\,\mathsf{polylog}(|\mathbb{F}|))$. □

4.2 Low Degree Testing

Low degree testing checks if there is a global low degree polynomial a proof is close to. This will be important for making our **PCP** robust. We start with the "line versus point" low degree test. For a function $f : \mathbb{F}^m \to \mathbb{F}$, the line versus point test chooses a random line through $\mathbb{F}^m$ and a claimed polynomial of what f should be on that line. Then it checks f at a random point on that line against the polynomial. The line versus point test has been extensively studied [PS94; FS95; AS97; Har+24]. We will use this to show that the test of checking if a random line is low degree is a robust low degree test, similar to [Ben+04].

Definition 4.2.1 (Line versus Point test)**.** *Let $\mathbb{F}$ be a field, f be a function $f : \mathbb{F}^m \to \mathbb{F}$, and degree d be an integer. For each line given by $l : \mathbb{F} \to \mathbb{F}^m$, let there be a degree d polynomial $g_l : \mathbb{F} \to \mathbb{F}$.*

The degree d line vs point test uniformly samples a line given by, $l : \mathbb{F} \to \mathbb{F}^m$ and a uniform $t \in \mathbb{F}$ then accepts if and only if

$$f(l(t)) = g_l(t).$$

We say that f passes the degree d line vs point test with probability ϵ if there is some set of functions g such that f and g pass the test with probability ϵ.

The failure probability of the line versus point test is related to the distance of f to a low degree polynomial [PS94; FS95; AS97]. To make our robust **PCP** as simple and efficient as possible, we will use a more recent, high error regime low degree test by Harsha, Kumar, Saptharishi, and Sudan [Har+24].

Lemma 4.2.2 (Line vs Point Test Measures Distance to Degree)**.** *There exists some $\tau \in (0, 1/3]$ such that for any finite field $\mathbb{F}$, integer m, and d, the following holds.*

Suppose some function $f : \mathbb{F}^m \to \mathbb{F}$ passes the degree d line vs point test with probability $\mu + \left(\frac{d}{|\mathbb{F}|}\right)^\tau$. Then there is a degree d polynomial $p : \mathbb{F}^m \to \mathbb{F}$ that agrees with f on μ fraction of points. That is:

$$\Pr_{x \in \mathbb{F}^m}[f(x) = p(x)] \geq \mu.$$

The line versus point test was used by [Ben+04] to create a robust line versus point test in order to construct their **PCP**. Here we give an even more robust version of this low degree test using the new line versus point test in combination with the sampling properties of lines. The result of [Ben+04] only gives robustness with large robust soundness error, while we give robustness with very small robust soundness error.

Lemma 4.2.3 (Robust Line vs Point Test)**.** *Take any field $\mathbb{F}$, integer dimension m, integer degree d, and constant $\beta \geq \left(\frac{d}{|\mathbb{F}|}\right)^\tau$ where τ is the constant from Definition 4.2.1. Take any $f : \mathbb{F}^m \to \mathbb{F}$ whose relative distance to the nearest polynomial is at least $\beta + \left(\frac{d}{|\mathbb{F}|}\right)^\tau$. Then for a random line ℓ we have*

$$\Pr_\ell[\textit{The relative distance of } f \circ \ell \textit{ to the closest degree } d \textit{ polynomial} \leq \beta] \leq \frac{\beta + 3\left(\frac{d}{|\mathbb{F}|}\right)^\tau}{1-\beta}.$$

Proof. There are two cases. Either f is far from all low degree polynomials, then we can use Lemma 4.2.2 to conclude the line versus point test fails often, and can get our result through a Markov inequality. Otherwise, we can use that lines are good samplers to say that with high probability, we will sample both the nearest polynomial and symbols that are not the nearest polynomial with high probability. Either many of the symbols close to the near polynomial will need to be changed to get a different polynomial, or the ones far will need to be changed to get a nearby polynomial.

So let $p : \mathbb{F}^m \to \mathbb{F}$ be nearest degree d polynomial to f. Define the set of points they agree as $A = \{x \in \mathbb{F}^m : p(x) = f(x)\}$, how much they agree as $\mu = \frac{|A|}{|\mathbb{F}^m|}$. Since the distance of f to p is at least $\beta + \left(\frac{d}{|\mathbb{F}|}\right)^\tau$, we have $\mu \leq 1 - \beta - \left(\frac{d}{|\mathbb{F}|}\right)^\tau$.

Let $\alpha = \sqrt{\frac{1}{|\mathbb{F}|\beta}}$. Since $\beta \geq \left(\frac{1}{|\mathbb{F}|}\right)^\tau$ and $\tau \leq 1/3$, we can bound α as

$$\alpha \leq \sqrt{\frac{|\mathbb{F}|^\tau}{|\mathbb{F}|}} \leq \frac{1}{|\mathbb{F}|^\tau} \leq \left(\frac{d}{|\mathbb{F}|}\right)^\tau.$$

Thus $\mu \leq 1 - \beta - \alpha$. Now we handle two cases.

$\mu \leq \beta + \alpha + \frac{d}{|\mathbb{F}|}$: Then we know that the degree d line vs point test must pass with probability at most $\beta + \alpha + \frac{d}{|\mathbb{F}|} + \left(\frac{d}{|\mathbb{F}|}\right)^\tau$, otherwise by Lemma 4.2.2, μ would be larger than this case. Thus the expected agreement between a line the nearest degree d polynomial is at most $\beta + \alpha + \frac{d}{|\mathbb{F}|} + \left(\frac{d}{|\mathbb{F}|}\right)^\tau$. So by a Markov inequality, the probability that the agreement is more than $1 - \beta$ is at most

$$\frac{\beta + \alpha + \frac{d}{|\mathbb{F}|} + \left(\frac{d}{|\mathbb{F}|}\right)^\tau}{1 - \beta}.$$

See that $\frac{d}{|\mathbb{F}|} \leq \left(\frac{d}{|\mathbb{F}|}\right)^\tau$. So we can bound the probability we are closer than β by $\frac{\beta + 3\left(\frac{d}{|\mathbb{F}|}\right)^\tau}{1-\beta}$.

$\beta + \alpha + \frac{d}{|\mathbb{F}|} \leq \mu \leq 1 - \beta - \alpha$: One can use pairwise independence of the points in random lines and Chebyshev bounds to show that

$$\Pr_\ell \left[|\mathbf{E}_{x \in \mathbb{F}}[\ell(x) \in A] - \mu| \geq \alpha \right] \leq \frac{1}{\alpha^2 |\mathbb{F}|}.$$

See that if ℓ intersects with A on more than $\beta + \frac{d}{|\mathbb{F}|}$ fraction of symbols, then $f \circ \ell$ must have distance β from any other degree d polynomials. If ℓ intersects with A on less than $1 - \beta$ fraction of inputs than $f \circ \ell$ has distance β from $p \circ \ell$. Thus with probability at all but $\frac{1}{\alpha^2|\mathbb{F}|} = \beta$, both of these hold and so $f \circ \ell$ is β away from any degree d polynomial.

□

Remark (Strength of this Robust Line vs Point Test)**.** *We note that this line vs point test is better than one gets by the standard reduction. The standard reduction says that if the function is about ϵ away from a polynomial, then the line versus point test fails with probability about ϵ, so there must be about $\epsilon/2$ fraction of lines that fail with distance at least $\epsilon/2$. This comes by a simple averaging argument. But we show that about $1 - \epsilon$ fraction of lines have distance about ϵ. We get this by exploiting the sampling property of lines.*

4.3 Extrapolatable PCPs

The purpose of an **ePCP** is to give an easy, efficient way to construct a robust **PCP** (**rPCP**). Since we will construct a robust **PCP** of proximity (**rPCPP**), we describe our **ePCP** with an implicit input as well. An **ePCP** is a **PCP** where:

1. An honest **PCP** proof is a low degree polynomial: $\pi : \mathbb{F}^m \to \mathbb{F}$. This allows us to make the **PCP** robust using an aggregation through curves type technique.

2. We relax soundness to only be against low degree proofs. This makes proving soundness of **ePCP**s easier.

3. The query function is extrapolatable (see Definition 1.2.1). This makes individual query locations of the robust **PCP** efficient to compute.

4. The implicit input must be contained in the proof. Ideally, it would be literal direct symbols in the proof, but sometimes the alphabet of the implicit input may be larger than that of the proof. In this case, we allow several symbols of the proof together to correspond to a symbol of the implicit input.

Now we formally define an extrapolatable **PCP** (see standard **PCP**, Definition 2.3.3, for reference).

Definition 4.3.1 (Extrapolatable **PCP**). *We say a non-adaptive* **PCP***, A, for a pair language L with verifier V, prover P, and query function Q is an extrapolatable* **PCP** *(***ePCP***) if for some m and d (both functions of n):*

1. *For some field $\mathbb{F}$, A uses alphabet $\mathbb{F}$.*

2. *The proof length is $|\mathbb{F}|^m$. That is a proof π together can be viewed as a function $\pi : \mathbb{F}^m \to \mathbb{F}$.*

Low Degree Completeness: *If $(x, y) \in L$, then there exists a proof $\pi^{x,y}$ such that $\pi^{x,y} : \mathbb{F}^m \to \mathbb{F}$ is a polynomial of degree at most d such that*

$$\Pr_r[V(x, r, \pi^{x,y}_{Q(x,r,\cdot)}) = 1] = 1,$$

and for every $i \in [l(n)]$, we have $P(x, y, i) = \pi^{x,y}_i$.

Decoding: *Suppose that the implicit input to L has alphabet Σ and length n'. Then for some integer ρ where $|\Sigma| \leq |\mathbb{F}^\rho|$, there is a $O(m\rho\,\mathsf{polylog}(|\mathbb{F}|))$ time, $O(m\rho \log(|\mathbb{F}|))$ space algorithm taking an index of $i \in [n']$ and outputting ρ indexes $j^i_1, \ldots, j^i_\rho \in \mathbb{F}^m$ such that there is a time $\rho\,\mathsf{polylog}(|\mathbb{F}|)$ and space $O(\rho \log(|\mathbb{F}|))$ computable bijection f between a subset of $\mathbb{F}^\rho$ and Σ such that for any $(x, y) \in L$ we have $f(\pi^{x,y}_{j^i_1,\ldots,j^i_\rho}) = y_i$. Further, if one is given any of $j^i_1, \ldots, j^i_\rho$, then in the same time and space one can determine the corresponding i.*

We say that π encodes implicit input y if for all $i \in [n']$ the corresponding $j^i_1, \ldots, j^i_\rho$ has $f(\pi_{j^i_1,\ldots,j^i_\rho}) = y_i$. Here, we allow y to contain symbols $\perp \notin \Sigma$ so that f is always defined.

Low Degree Soundness: *For any $(x, y) \notin L$ and any degree at most d polynomial proof $\pi' : \mathbb{F}^n \to \mathbb{F}$ that encodes y we have*

$$\Pr_r[V(x, r, \pi'_{Q(x,r,\cdot)}) = 1] \leq \delta.$$

Further, we say A has:

1. *Extrapolation time $t(n)$ if for any x, r, the function $Q_{x,r}(i) = Q(x, r, i)$ is time $t(n)$ extrapolatable.*

2. *Degree d and m variables.*

3. *Low degree soundness δ.*

4. *Perfect low degree completeness.*

5. *ρ proof symbols per implicit symbol.*

To make our **rPCP** from an **ePCP**, the idea is to take an **ePCP**, and instead of querying points, query all the points along a curve that goes through those points. Since low degree functions are an error correcting code, restricting low degree proofs to a low degree curve gives an error correcting code. So by querying entire curves, we can make the set of accepted query values for our **PCP** verifier an error correcting code.

Querying along a line and checking if it is low degree performs a low degree test. A low degree test only succeeds with high probability if a proof is close to a global, low degree polynomial. Then since low degree polynomials are error correcting codes, if the query values are close to both the global low degree polynomial and an accepted proof, the accepted proof is the global low degree polynomial. If the query values for a proof are close to being accepted often, we show a global low degree proof for the original **ePCP** succeeds often.

Then the idea of the protocol is to choose the randomness for the **ePCP**, take a curve through all the query points of the **ePCP**, query all the locations along this curve and check if the the curve is low degree, and the **ePCP** accepts the proof on this curve. This is almost what the **rPCP** does, with a few caveats:

1. We also perform a robust line versus point test. This gives our line versus point test robustness since low degree polynomials are an error correcting code.

2. To guarantee the curve is consistent with the global low degree polynomial with high probability, we need a random point on the curve to be approximately uniform over $\mathbb{F}^m$. So we also choose another random point in the proof, and use a curve that goes through the **ePCP** queries and that point.

   From Lemma 4.1.1, the function going through all these points is still extrapolatable, and so by Lemma 4.1.4 we can efficiently compute the curve going through them.

We can also give any **ePCP** proximity by adding extra queries to the implicit input. Since **ePCP**s only have to handle soundness against low degree polynomials, to change a symbol in the implicit input requires changing most of the proof. Thus checking random points in the implicit input input is enough to verify the implicit input is (close to) the input used in the proof. The same techniques for making an **ePCP** a robust **PCP** also make this **ePCP** of proximity a robust **PCP** of proximity (**rPCPP**).

We note that this is a standard technique, the only extra precaution is that we need to keep the verifier space, and query time low. Keeping verifier space low requires a space efficient low degree test. This space efficient low degree test requires time $|\mathbb{F}|^3 \operatorname{\mathsf{polylog}}(|\mathbb{F}|)$, whereas a standard less space efficient low degree test requires time $|\mathbb{F}| \operatorname{\mathsf{polylog}}(|\mathbb{F}|)$. This increases the decision complexity somewhat, but we will do **PCP** composition to reduce decision complexity so it is not a problem. Keeping query time low is handled by the fact that **ePCP** queries are extrapolatable.

Theorem 4.3.2 (**ePCP** gives efficient **rPCPP**)**.** *For any pair language L with an* **ePCP***, A, with*

1. *Decision complexity $t(n)$.*
2. *Verifier space $s(n)$.*
3. *Extrapolation time $t'(n)$.*
4. *Randomness $r(n)$.*
5. *Degree $d(n)$ and $m(n)$ variables.*
6. *$q(n)$ queries.*
7. *Alphabet $\mathbb{F}$.*
8. *Prover P.*
9. *Low degree soundness δ_1.*
10. *Perfect low degree completeness.*
11. *Degree d.*
12. *ρ implicit symbols per proof symbol.*

Take any positive η, β such that $|\mathbb{F}| > 10d(n)\left(q(n) + \lceil \frac{10\rho \log(1/\delta_1)}{\eta} \rceil\right)$, and $3\left(\frac{d}{|\mathbb{F}|}\right)^\tau \leq \beta \leq \min\{\frac{\eta}{2}, \frac{1}{10}\}$ holds (where τ is the universal constant from Lemma 4.2.2). Then the language L has an **rPCPP***, B, with*

1. *Decision complexity $O(t(n) + |\mathbb{F}|^3 \rho \operatorname{\mathsf{polylog}}(|\mathbb{F}|))$.*
2. *Verifier space $O(s(n) + \rho \log(|\mathbb{F}|))$.*
3. *Randomness $r(n) + O\left(\frac{m(n)\log(1/\delta_1)}{\eta} \log(|\mathbb{F}|)\right)$.*

4. *Query time* $O\left(t'(n) + \left(q(n) + \frac{m(n)\rho\log(1/\delta_1)}{\eta}\right)\mathsf{polylog}(|\mathbb{F}|)\right)$.

5. $O(|\mathbb{F}|)$ *queries.*

6. *Prover* P*, alphabet* $\mathbb{F} \cup \Sigma$*, and perfect completeness.*

7. *Proximity* η.

8. *Robust soundness error at most* $\delta_1 + 3\beta$.

9. *Robustness parameter* $\beta/3$.

Remark (Efficiency of the Prover)**.** *We emphasize that the prover of the* **rPCPP** *is the same as that of the* **ePCP***. So if the prover of the* **ePCP** *is efficient, so is the prover of the* **rPCPP***.*

Proof. Let V be the verifier, Q the query function, and P the prover from **ePCP**, A. We construct a new **rPCPP**, B, that expects the same low degree polynomial as proof as A. Our new verifier will be V' and our new query function Q'. We will start by describing our protocol from a high level, pointing out which parts are done by a new query function Q' and V' later.

First, on explicit input x, implicit input y, and proof π, our **rPCPP** system B will choose the randomness for A, call it r. This determines the query points for A, which are $Q(x, r, \cdot)$. By assumption, $Q_{x,r}(i) = Q(x, r, i)$ is time $t'(n)$ extrapolatable.

Let $c = \lceil \frac{10\log(1/\delta_1)}{\eta} \rceil$. Then B chooses c random indexes in the implicit input to check. These correspond to $c' = c\rho$ locations in the low degree polynomial, $w_1, \ldots, w_{c'} \in \mathbb{F}^m$ that we will use to check the implicit input. We also choose some random index into the whole proof, $w_{c'+1} \in \mathbb{F}^m$. The function $w : [c'+1] \to \mathbb{F}^m$ that takes i and outputs w_i is time $O(c'm\,\mathsf{polylog}(|\mathbb{F}|))$ extrapolatable. Let $g : \mathbb{F} \to \mathbb{F}^m$ be the degree $q(n) + c'$ function such that, for each $t \in [q(n)]$, we have $g(t) = Q(x, r, t)$, and for every $i \in [c'+1]$ we have $g(q(n) + i) = w_i$.

Then $\pi \circ g$ is a degree $d' = d(q + c')$ polynomial if π is actually a degree d polynomial. Our new **rPCPP** verifier V' will check every point along $\pi \circ g$ and verify it is a degree d' polynomial that would cause our **ePCP** verifier V to accept.

Next the verifier chooses a random $z \in \mathbb{F}^m$ to run a robust line versus point test with the line ℓ defined by $\ell(i) = w_{c+1} + i \cdot z$. Altogether, B uses randomness $r(n) + (c+2)m(n)\log(|\mathbb{F}|)$. Let r' be the choice of randomness. Then the query function $Q' : [2|\mathbb{F}|] \to \mathbb{F}^m$ is defined by

$$Q'(x, r', i) = \begin{cases} g(i) & i \le |\mathbb{F}| \\ \ell(i - |\mathbb{F}|) & i > |\mathbb{F}| \end{cases}$$

(where we define $g(|\mathbb{F}|) = g(0)$ and $\ell(|\mathbb{F}|) = l(0)$) with one exception. When Q would query one of the inputs corresponding to y, it queries y instead. This is efficient because **ePCP**s have efficient decoding.

We call the first $|\mathbb{F}|$ queries the curve queries and the rest of the queries the line queries. Similarly, we call π evaluated on the first $\mathbb{F}$ queries the curve values and π on the rest of the queries the line values.

Finally, as an accounting trick, we then repeat the queries into y, (which would be the locations at $w_1, \ldots, w_{c'}$, except the verifier knows these actually come from y so it queries y instead) until we also query indexes in y approximately $|\mathbb{F}|$ times and make sure the values don't change. Call these the y queries. This is just to make sure distance on these indexes cost more when calculating robustness. Equivalently, one could also use a more fine grained notion of robustness that measured distance on π and y differently.

The verifier V' first checks if the **ePCP** would accept the curve values, that is, if $V(x, r, \pi_{Q(x,r,\cdot)}) = 1$. It can do this since the first q queries of Q' are the same as Q. Then the verifier checks if the curve values are a degree d' polynomial. Finally, it checks if the proof restricted to the line is a degree d polynomial. Our new verifier V' accepts only if all of these checks pass.

Now to keep verifier space down, we need to be a little careful how we implement our low degree test, so we describe that first. Let $f := \pi \circ g$ so that f is a function outputting the curve values. Using the degree d' interpolating polynomials,

$$l_i^{d'}(x) = \prod_{j \in [d'] \setminus \{i\}} \frac{x - j}{i - j},$$

we can write a degree d' polynomial, h:

$$h(x) = \sum_{i \in [d']} l_i^{d'}(x) f(i).$$

If f is a degree d' polynomial, then $f = h$. To see if $f = h$, we calculate h at each point and compare to f.

Each $l_i^{d'}$ can be computed directly by simply looping through each terms in the sums and products, calculating them from the definition, and reusing the space each time. Notably, we do NOT calculate the interpolating polynomials the same way we computed them in Lemma 4.1.4. That version uses more memory, but less time, and in this case we need less memory but allow for more time. Instead, we use the naive algorithm following the definition directly. We do a similar thing for the line versus point test.

Now to argue we achieve the stated performance.

1. Now we show the decision complexity is $O(t(n) + (|\mathbb{F}|^3 + |\mathbb{F}|\rho) \,\mathsf{polylog}(|\mathbb{F}|))$.

   The decision complexity is just the time to simulate V, which is $t(n)$, plus the time it takes to perform the low degree tests. To test the low degree of f takes $O(|\mathbb{F}|)$ calculations of $h(x)$. Each $h(x)$ only takes $O(d')$ calculations of $l_i^{d'}(x)$. Each $l_i^{d'}(x)$ only takes time $O(d' \,\mathsf{polylog}(|\mathbb{F}|))$. Thus the total time for the low degree test of f is

   $$O(|\mathbb{F}| d' \,\mathsf{polylog}(|\mathbb{F}|)) = O(|\mathbb{F}|^3 \,\mathsf{polylog}(|\mathbb{F}|)).$$

   There is an additional $\rho \,\mathsf{polylog}(|\mathbb{F}|)$ time from decoding the **ePCP** which we do at most $|\mathbb{F}|^3$ times. The check takes at most this long, so the overall time is

   $$O(t(n) + |\mathbb{F}|^3 \rho \,\mathsf{polylog}(|\mathbb{F}|)).$$

2. Now we show the verifier space is $O(s(n) + \rho \log(|\mathbb{F}|))$.

   Calculating a single $l_i^{d'}$ only requires keeping track of a constant number of field elements and a pointer for j. Then given that, $h(x)$ only needs the additional space for another counter for i and another field element. Finally, comparing all of the $h(x)$ to the $f(x)$ only takes space for another pointer for the x and another field element. So it only requires a constant number of pointers and field elements. The line tests are similar.

   So the total space of V' is the space used to run V plus $O(\log(|\mathbb{F}|))$, plus a $\rho \log(|\mathbb{F}|)$ overhead to decode. So the total space is $O(s(n) + \rho \log(|\mathbb{F}|))$.

3. As already shown, B uses randomness $r(n) + (2 + c)m(n) \log(|\mathbb{F}|)$.

4. Next, we show the query time of the robust **PCP**.

   By assumption, the query locations of Q are time $t'(n)$ extrapolatable. And by Lemma 4.1.1, adding w gives a $O(t' + c'm \log(|\mathbb{F}|))$ extrapolatable function. And g is the low degree extrapolation of this sequence.

   By Lemma 4.1.4, we can calculate g in time $O(t'(n) + (c'm + q) \,\mathsf{polylog}(|\mathbb{F}|))$. This handles the curve queries, as these are just evaluations of g.

   The line queries just return a point in $\ell(i) = w_{c'+1} + i \cdot z$. These can be calculated in $O(m \,\mathsf{polylog}(|\mathbb{F}|))$ time. To check if any of the query locations need to index into the implicit input y only takes an additional $m\rho \,\mathsf{polylog}(|\mathbb{F}|)$ time from decoding of the **ePCP**.

   In either case, we calculate Q' in time

   $$O(t'(n) + ((c' + \rho)m + q) \,\mathsf{polylog}(|\mathbb{F}|)).$$

5. The number of queries are at most $3|\mathbb{F}| = O(|\mathbb{F}|)$.

6. These hold by construction, and since the **ePCP** has perfect completeness.

7,8,9. Now we need to show proximity η, robust soundness error $\delta_1 + 3\beta$, and robustness parameter $\beta/3$. Let $\delta_2 = 3\beta$.

Suppose that x, y is such that for all y' with $(x, y') \in L$ we have that $\Delta(y, y') \geq \eta$. Take any proof π. Let π' be the degree d polynomial closest to π and let y' be the implicit input that π' encodes. Suppose the relative distance between π and π' is ϵ. We divide the proof into two cases depending on the size of ϵ.

$\epsilon < \frac{9\delta_2}{10}$: In this case, the proof is basically a low degree polynomial. Since $1 - (q + c)/|\mathbb{F}| \geq 1 - d'/|\mathbb{F}|$ fraction of the curve queries are uniformly random, the probability we choose more than $1 - \beta - d'/|\mathbb{F}|$ fraction of curve query locations that π and π' differ is at most $\frac{\epsilon}{1-\beta} \leq \frac{10\epsilon}{9} \leq \delta_2$. Notice the relative distance of a degree d' polynomial over $\mathbb{F}$ is at least $1 - \frac{d'}{|\mathbb{F}|}$.

We further break this case down into whether y' is close to y.

$\Delta(y, y') < \eta$: Then (x, y') is not in L and we know that the verifier only accepts π' with probability at most δ_1. If for a choice of randomness the **ePCP** verifier would reject π' and on the curve queries π' and π disagree on at most $1 - \beta - d'/|\mathbb{F}|$ fraction of places, then the relative distance of the assignments on the curve queries to one the prover would accept is at least β. Thus the probability that the proof is within $\beta/3$ of an accepting proof is at most $\delta_1 + \delta_2$.

$\Delta(y, y') \geq \eta$: In this case, y and y' disagree on many locations, so the y queries should find this. In particular, the probability a particular query to y is where y and y' disagree is at least $\frac{1}{\eta}$. So by a Chernoff bound the probability we don't make an $\frac{\eta}{2} \geq \beta$ fraction of queries where they disagree is at most δ_1.

If for a choice of randomness the curve queries on π' and π disagree on at most $1 - \beta - d'/|\mathbb{F}|$ fraction of places, and this includes locations where y and y' differ, then to make the verifier accept, one either needs to change the y queries to make them agree with y', or change a β fraction of the curve queries to make it a different low degree polynomial. If y and y' differ on β fraction of y queries, than the distance to an accepting proof is at least $\beta/3$. Thus the probability of being within $\beta/3$ of an accepting proof is at most $\delta_1 + \delta_2$.

$\frac{9\delta_2}{10} \leq \epsilon$: Then $\epsilon \geq \frac{9 \cdot 3\beta}{10} > 2\beta \geq \beta + \left(\frac{d}{|\mathbb{F}|}\right)^{\tau}$. So by Lemma 4.2.3,

$$\begin{aligned}\Pr_{\ell}[\text{The relative distance of } f \circ \ell \text{ to the closest degree } d \text{ polynomial} \leq \beta] \leq & \frac{\beta + 3\left(\frac{d}{|\mathbb{F}|}\right)^{\tau}}{1 - \beta} \\ \leq & \frac{2\beta 10}{9} \\ < & \delta_2.\end{aligned}$$

Thus the probability that the line queries are within β relative distance of a degree d polynomial is δ_2. Thus the probability of all of the queries being within $\beta/3$ of being accepted is at most δ_2.

□

Remark (Better **ePCP** to **rPCP** Reduction)**.** *We note that this* **PCP** *reduction is not optimized and the specific parameters were chosen for convenience. Other constructions achieve better parameters in many regards, but not necessarily verifier space (which translates into prover space after composition) and query time. Prover space and query time will not improve under composition, so these parameters are important to optimize in our application. Our contribution is showing these parameters can be kept small.*

5 Constructing our ePCP

We use a BFL style base **PCP**. This is one of the earliest and most common kinds of **PCP**s. Our contribution is showing that this **PCP** is an **ePCP** which in particular means it has very small query time. The **ePCP** can be thought of as having three parts. The first part is a multilinear extension of the computation history. The second is a low degree polynomial that is some consistency check of the computation history. And the final part is a series of low degree polynomials that are proofs that the consistency check pass. Finally, we add another variable to package all of these parts into a single low degree polynomial. More explicitly, the proof the checks the verifier perform are the following.

1. The verifier makes sure the computation history starts at a valid start state. We can do this by computing the multilinear extension of the input at a random point and comparing it to the corresponding symbol in what should be a multilinear extension of the computation history.

   The computation history is just the work tape of the Turing machine at every time step (with an indicator of whether the work head is on that cell). One subtle issue is that the Turing machine is *not* a single tape Turing machine, it is a two tape Turing machine where the work tape is much shorter than the input tape. It is well known that we can convert a two tape Turing machine to a single tape Turing machine, but we need to further show that the prover can space efficiently compute the computation history of this single tape Turing machine.

2. Since Turing machines are local, the local check formula of whether the computation history is correct is a constant degree polynomial of computation history. This can be used to give a low degree polynomial that is zero at the binary index of a cell if that cell is derived through the rules of Turing machine from the previous time step. The prover can compute this space efficiently if it can compute the computation history itself efficiently. The verifier can check if this consistency polynomial is from the computation history by just computing it at a random point.

3. Finally, a sum check [Lun+92] is used to verify that consistency polynomial is zero on all binary inputs (these inputs correspond to checking if the entries in the computation history are all consistent).

We emphasize that all of these steps are standard, except that:

1. We need to show computing this computation history can be done space efficiently. Since this is *not* standard, we need to explain what the computation history is and how we simulate it efficiently in more detail.

2. We need to show that the locations the verifier queries are efficiently extrapolatable.

5.1 Arithmetization

This paper frequently uses arithmetizations of boolean functions. We say that a function $f : \mathbb{F}^n \to \mathbb{F}$ is consistent with a boolean function $g : \{0,1\}^n \to \{0,1\}$ if f agrees with g when restricted to boolean inputs. If further f is a low degree polynomial, f is often called an arithmetization of g.

An example of an arithmetization is the multilinear extension of a boolean function. That is just the unique multilinear function, f, that agrees with g on boolean inputs. These can often be constructed very efficiently. For instance, the multilinear extension of the equality function has a simple and efficient to evaluate description.

Definition 5.1.1 (Equality Arithmetization)**.** *For field $\mathbb{F}$, and $l \geq 1$, define $equ : \mathbb{F}^l \times \mathbb{F}^l \to \mathbb{F}$ as:*

$$equ(u, v) = \prod_{i \in [l]} u_i \cdot v_i + (1 - u_i) \cdot (1 - v_i).$$

Observe that equ is the multilinear extension of the Boolean equality function.

But even for boolean functions whose multilinear extensions can't be computed time efficiently, there is a space efficient, brute force way to compute it.

Lemma 5.1.2 (Multilinear Extensions Can Be Calculated in Low Space)**.** *Suppose function $G : \{0,1\}^n \to \{0,1\}$ is computable in space S and time T. Then the multilinear function g consistent with G on Boolean inputs is computable in space $O(n + \log(|\mathbb{F}|) + S)$ and time $2^n(T + \mathsf{polylog}(|\mathbb{F}|))$.*

Proof. This follows from the fact g can be written as

$$g(x) = \sum_{y \in \{0,1\}^n} G(y)\mathrm{equ}(y, x).$$

Then this can be evaluated using only a pointer for y, a small amount of space for equ, $O(n)$ field elements, and the space to evaluate G. □

Thus if we can compute the computation history space efficiently, we can compute its multilinear extension space efficiently.

5.2 Space Efficient Simulation of Single Tape Simulation

First, we need to translate from the two tape Turing machine model to the single tape Turing machine model. This is important because a single tape Turing machine has better locality than a two tape Turing machine, which we need when arithmetizing the local consistency checks. But since our prover is low space, it cannot directly simulate the single tape Turing machine, as this would require at least as much space as the input, which is much longer than the working space. So we also need to describe how a small space algorithm can still output what would be in this single tape Turing machine at any step. Ultimately, this is just the Cook-Levin reduction, but with more attention on how space efficient this reduction is for the prover.

Everything in this result is standard, except the final note that the one tape TM also has an efficient simulation by a two tape TM, although it is not a difficult result.

Lemma 5.2.1 (Two Tape TMs Have Simple One Tape TMs). *Let A be a two tape nondeterministic Turing maching recognizing L, running in time T (where $T = \Omega(S)$) with a space S (where $S = \Omega(\log(n))$ read/write worktape, and a length n read-only input tape.*

Then there is a 1 tape TM B such that:

1. *B runs in time $T' = \mathsf{poly}(T, n)$, and space $S' = O(n + S)$.*
2. *B has a constant size alphabet, Σ, where for some k, we have $|\Sigma| = 2^{2^k}$. That is, Σ is represented by a power of 2 number of bits. Further, the alphabet Σ includes whether the head is currently on that cell in the work tape.*
3. *For any input x for A, there is a corresponding input for B, w_x, of length S'. Specifically we have that $w_x = (w^1, w_x^2, w^3)$ where*
   - *(a) w^1 has length $O(\log(S'))$ and is independent of the specific x, only the length of x, and w^1 is computable in time $O(|w^1|)$.*
   - *(b) w_x^2 is exactly n symbols where for some $f : \{0,1\} \to \Sigma$, for each $i \in [n]$, $(w_x^2)_i = f(x_i)$, where f is computable in constant time.*
   - *(c) w^3 is exactly S copies of a specific symbol in Σ.*
4. *Not all transitions for B will be defined. In particular, the transition from a reject state will be undefined (so no valid transitions) and a transition from an accept state will be a self loop. Thus A accepts on x if and only if after time T' starting on w_x, B reaches a steady state. Similarly, A rejects on x if and only if there is no sequence of T' valid transitions in B starting from w_x.*
5. *If B has a starting state that is (w^1, z) for any z that is not (w_x^2, w^3) for some $x \in L$, then B will not have T' valid transitions.*

If A is a deterministic, then B is deterministic and the following holds. Let $x \in L$ be an input for A, with transformed input for B, w_x. Given a time $t \in [T']$ and a memory location $s \in [S']$, there is a two tape TM C that can compute the symbol in cell s at time t in B's computation history on y_x in time $O(\mathsf{poly}(T))$ given read only access to x and a work tape of size $O(S)$.

Remark (Specific Input Format)**.** *The exact structure of the transformed input may seem overly specific, but we need this extra structure to get proximity. In particular, our* **PCPP** *will need to know which cells encode a known, explicit, first input, and which cells encode an unknown, implicit, second input.*

Remark (Two Tape TM vs RAM algorithm)**.** *The uniform circuit model (which is equivalent to one dimensional Turing machine) and RAM model of computation are two common models of computation. RAM algorithms can simulate Turing machines with very little time and space overhead. Turing machines can simulate RAM algorithms with a polynomial time and very little space overhead. Thus a similar theorem to Lemma 5.2.1 holds for both RAM algorithms and Turing machine algorithms.*

We state the reduction for two tape Turing machines since that is the standard model and to keep it clear that our PCP verifiers are also Turing machines. However, our actual proof Lemma A.0.1 shows the reduction for more general RAM algorithms, and the two tape Turing machine case is a corollary.

Standard two tape to one tape TM reductions satisfy all these requirements. The only trick to this analysis is realizing that there is a direct correspondence between the states of the two tap Turing machines, and the state of one tape Turing at regular intervals. Between the regular intervals, the state of the one tape Turing Machine changes very little, and those changes are simple and predictable. More details are given in Appendix A.

For the purpose of analyses, it will be useful to look at a multilinear extension associated with a low degree polynomial. So we define the following purely to simplify the analysis of our **PCP**.

Definition 5.2.2 (Multilinear Extension of Binarized function (MLB))**.** *For any function $f : \mathbb{F}^n \to \mathbb{F}$, there is a unique, multilinear function, $g : \mathbb{F}^n \to \mathbb{F}$, such that for all binary $x \in \{0,1\}^n$,*

$$g(x) = \begin{cases} 0 & f(x) = 0 \\ 1 & f(x) \neq 0 \end{cases}.$$

Then we say $MLB(f) = g$.

We can construct our inconsistency check from a claimed multilinear extension of a computation history. This comes from arithmetizing the transition rules of a Turing machine. The construction is straightforward, but details can be found in Appendix A.

Lemma 5.2.3 (Inconsistency Function)**.** *Let k be a constant, s, t be integers, and $\mathbb{F}$ be a field. Let B be a Turing machine with 2^{2^k} different states per cell running in $S = 2^s$ cells, and time $T = 2^t$. Then there is a function Γ_B taking any function $X : \mathbb{F}^s \times \mathbb{F}^k \times \mathbb{F}^t \to \mathbb{F}$ and returning a function $Y : \mathbb{F}^{3s+2t+4(2^k)} \to \mathbb{F}$ such that:*

1. *If X is Boolean on Boolean inputs, Y is Boolean on Boolean inputs.*
2. *If X is Boolean on Boolean inputs, then Y is 0 on all Boolean inputs if and only if X on Boolean inputs encodes a valid computation history for B. That is, X describes a sequence of T different states of B and they are each derived by a valid transition rule.*
3. *If X is degree d, then Y is degree $O(s + t + d)$. If X is degree d in every variable individually, Y is degree $O(d)$ in every variable individually.*
4. *Given oracle access to X, Y can be computed in time $O((t + s)\,\mathsf{polylog}(|\mathbb{F}|))$ with a constant number of calls to X.*
5. *If $Y = \Gamma_B(X)$ is 0 on all Boolean inputs, then $\Gamma_B(MLB(X))$ is also 0 on all Boolean inputs.*

5.3 Sum Check Protocols

Our **PCP** uses a variation (similar to those used in [Ben+04; KRR21; HR18]) of the sum check protocol [Lun+92]. We assume some familiarity with the protocol. The sum check is a standard element of **PCP**s, and all we add is a small bit of analysis that sum check is extrapolatable. We use sum check in the **PCP** or **MIP** setting where the prover can only depend on the current query. We use V as the verifier (the "decision" part of the verifier), Q as the query locations, f as the prover, and g as the low degree polynomial we want to check the sum of. Our only new contribution is showing that Q is extrapolatable.

Lemma 5.3.1 (Sum Check Protocol)**.** *Let $n, d \in \mathbb{N}$, and $\mathbb{F}$ be a field with $|\mathbb{F}| > (d+1)n$. Then there is some protocol, A, so that for any $f : \mathbb{F}^n \times \mathbb{F} \to \mathbb{F}$:*

1. *For some $m = O(nd)$ and $R = 2n$, there is a verifier $V : \mathbb{F}^m \to \{0,1\}$ and query function $Q : \mathbb{F}^R \times [m] \to \mathbb{F}^n \times \mathbb{F}$ so that*
$$A(f, r) = V(f(Q(r,1)), f(Q(r,2)), \dots, f(Q(r,m))).$$
2. *V runs in time $O(nd\,\mathsf{polylog}(|\mathbb{F}|))$ and space $O(nd\log(|\mathbb{F}|))$.*
3. *For any $r \in \mathbb{F}^R$, for $Q_r(i) = Q(r,i)$, Q_r is time $O(nd\,\mathsf{polylog}(|\mathbb{F}|))$ extrapolatable.*
4. *For any $r \in \mathbb{F}^R$, the last coordinate of Q is always an element of[9] $[n+1]$. That is, for all $i \in [m]$, $Q(r,i)_{n+1} \in [n+1]$*

   Further, the last coordinate of Q is only equal to $n+1$ at most $O(d)$ times.

Completeness *For any $g : \mathbb{F}^n \to \mathbb{F}$ where g has max degree d in any individual variable, if for all $x \in \{0,1\}^n, g(x) = 0$, then there is some $f : \mathbb{F}^n \times \mathbb{F} \to \mathbb{F}$ so that:*

- *For all $x \in \mathbb{F}^n$ we have $g(x) = f(x, n+1)$.*
- *Sum check succeeds on f:*
$$\Pr_r[A(f,r) = 1] = 1.$$
- *Function f has degree at most d in each of its first n variables.*
- *If function g is computable in space S, then function f is computable in space $O(n\log(|\mathbb{F}|) + S)$.*

Soundness *for any $g : \mathbb{F}^n \to \mathbb{F}$ where g has max individual degree d', if there exists $x \in \{0,1\}^n$ such that $g(x) \neq 0$, then for any $f : \mathbb{F}^n \times \mathbb{F} \to \mathbb{F}$ so that for all $x \in \mathbb{F}^n, g(x) = f(x, n+1)$, sum check fails with high probability:*
$$\Pr_r[A(f,r) = 1] \leq \frac{(d'+1)n}{|\mathbb{F}|}.$$

To prove extrapolatability, we first formally define the sum check algorithm. We note that we use a variation of the standard sum check protocol. In the standard sum check protocol, you verify the value of a sum where the sum is performed in a finite field. If the field is of a small characteristic (less than 2^n), then this sum may be zero even if there is a non-zero value in the sum. To get around this, our protocol verifies a related multilinear extension of a boolean function is zero at a random point. A similar variation has been used in multiple places [Ben+04; HR18; KRR21].

We will prove just the extrapolatable property in the body of this paper. For completeness, we prove the rest of the properties in Appendix B.

Definition 5.3.2 (Sum Check Protocol Definition)**.** *Let $n, d \in \mathbb{N}$, and $\mathbb{F}$ be a field with $|\mathbb{F}| > \max\{d, n\} + 1$. Suppose $f : \mathbb{F}^n \times \mathbb{F} \to \mathbb{F}$. Then the degree d Sum Check Protocol on f is the following randomized algorithm.*

1. *Get $2n$ random field elements, $R = (r_1, \dots, r_n$ and $r'_1, \dots, r'_n)$.*
2. *Reject if $f((r_1, \dots, r_n), 1) \neq 0$.*
3. *For i from 1 to n:*

   (a) *For $j \in [d+1]$, query*
$$a_i^j = f((r'_1, \dots, r'_{i-1}, j, r_{i+1}, \dots r_n), i+1).$$
   Using these, let $g_i : \mathbb{F} \to \mathbb{F}$ be the degree d polynomial so that for all $j \in [d+1]$, $g_i(j) = a_i^j$.

[9]Any reasonable embedding of $[n+1]$ in $\mathbb{F}$ works, just as in the definition of extrapolatability from Section 4.1.

(b) If

$$f((r'_1,\dots,r'_{i-1},r_i,\dots r_n),i)\neq(1-r_i)g_i(0)+r_ig_i(1)$$

reject.

(c) If

$$f((r'_1,\dots,r'_i,r_{i+1},\dots r_n),i+1)\neq g_i(r'_i)$$

reject.

4. If all checks pass, accept.

Remark (Randomness Efficiency)**.** *The sum check protocol as defined here has soundness even in interactive proofs where there is a single prover that can remember prior queries. However, if one only requires soundness in multi-interactive proofs or* **PCP***s, then one can reduce the amount of randomness to just n field elements. This uses the same protocol, except that $r'_i = r_i$ for all i. Thus even the more randomness efficient protocol is extrapolatable.*

Note one can actually skip querying the points in (a) and (b). Where we query f in (b) in one iteration is where we queried f in (c) during the last iteration. We can skip (c) as well by assuming equality holds. Then step (b) just compares g_{i-1} to g_i. We use the test as described because it is easier to describe.

At a high level, this protocol expects a sequence of polynomials where $f_i(x) = f(x,i)$. Think of f_{n+1} as some degree d arithmetization of a boolean function and f_1 as the multilinear extension of a boolean function. Each f_i reduces the degree of f_{i+1} in one of the variables from d to 1. Step 2 of the sum check verifies that f_1 is the zero function. Step 3 (b) verifies that f_i is linear in variable i and relates correctly to f_{i+1}. Step 3 (c) is to make sure f_{i+1} is of degree d in variable i.

The correctness of this protocol is standard. Here, we only show that this protocol is extrapolatable.

Lemma 5.3.3 (Sum Check Queries Are Extrapolatable)**.** *For $n,d\in\mathbb{N}$, field $\mathbb{F}$ with $|\mathbb{F}| > \max\{d,n\}+1$, and $r=(r_1,\dots,r_n,r'_1,\dots r'_n)\in\mathbb{F}^{2n}$, the degree d sum check query locations, Q_r, used in Definition 5.3.2, are $O(nd\,\mathsf{polylog}(|\mathbb{F}|))$ extrapolatable.*

Proof. First we define the degree d sum check query location function, $Q_r:[(d+3)n]\to\mathbb{F}^n\times\mathbb{F}$. We will do this by indicating what is queried for each i in the loop of step 3. For any $i\in[n]$ in the loop the next $d+3$ queries are defined by, for $l\in[d+3]$:

$$Q_r((d+3)(i-1)+l)=\begin{cases}\big((r'_1,\dots,r'_{i-1},r_i,r_{i+1},\dots r_n),i\big) & l=1\\ \big((r'_1,\dots,r'_{i-1},1,r_{i+1},\dots r_n),i+1\big) & l=2\\ \vdots & \\ \big((r'_1,\dots,r'_{i-1},d+1,r_{i+1},\dots r_n),i+1\big) & l=d+2\\ \big((r'_1,\dots,r'_{i-1},r'_i,r_{i+1},\dots r_n),i+1\big) & l=d+3\end{cases}.$$

These are the queries made to f by the sum check protocol for randomness $r=(r_1,\dots,r_n,r'_1,\dots,r'_n)$. For each i, the first query is to sample the point used in step (b) of Definition 5.3.2. The next $d+1$ queries are to calculate g_i in step (a). And the last query is made to get the point in step (c). We give them in this order because it makes the extrapolation formula simpler. And note that the test in step 2 also queries $((r_1,\dots,r_n),1)$, which is the same location as $Q_r(1)$.

To show Q_r is extrapolatable, take $v_1,\dots,v_{(d+3)n}$. We need a time $O(nd\,\mathsf{polylog}(|\mathbb{F}|))$ algorithm computing

$$u=\sum_{i\in[(d+3)n]}v_iQ_r(i).$$

Note u is an $n+1$ component vector. See that component j of Q is fixed to r_j until we get to query $(d+3)(j-1)+1$. Then after that query $(d+3)j$ component j is fixed to r'_j. Thus for any give $j\in[n]$,

$$u_j=\left(\sum_{i=1}^{(d+3)(j-1)+1}v_ir_j\right)+\left(\sum_{i=1}^{d+1}iv_{((d+3)(j-1)+1+i)}\right)+\left(\sum_{i=(d+3)j}^{(d+3)n}v_ir'_j\right).$$

We will handle u_{n+1} at the end.

First, look at the first and last terms. For $j \in [n]$, let

$$\alpha_j = \sum_{i=1}^{(d+3)(j-1)+1} v_i$$
$$\beta_j = \sum_{i=(d+3)j}^{(d+3)n} v_i.$$

Then we can rewrite u_j as

$$u_j = \alpha_j r_j + \left(\sum_{i=1}^{d+1} i v_{((d+3)(j-1)+1+i)} \right) + \beta_j r'_j.$$

Then an iterative algorithm can calculate every α_j and β_j in $O(nd\,\mathsf{polylog}(|\mathbb{F}|))$ time, since there are only $O(nd)$ terms in each sum and the only difference between α_j and α_{j+1} is $d+3$ more terms in the sum. Similarly for β_{j+1} and β_j. The middle sum only has $d+1$ terms in it, which we can calculate in time $d\,\mathsf{polylog}(|\mathbb{F}|)$. So given α_j and β_j, u_j can be calculated in $O(d\,\mathsf{polylog}(|\mathbb{F}|))$ time. So all u_j for $j \in [n]$ can be calculated in $O(nd\,\mathsf{polylog}(|\mathbb{F}|))$ time.

Now u_{n+1}, as a single component, can just straightforwardly be evaluated in time $O(nd\,\mathsf{polylog}(|\mathbb{F}|))$ from the definition. Thus u can be calculated in $O(nd\,\mathsf{polylog}(|\mathbb{F}|))$ time. □

For completeness, we prove the rest of Lemma 5.3.1 in Appendix B.

5.4 Constructing our ePCP

The idea of the **PCP** is to ask the prover for the function that takes a time t and a bit of memory s and returns the value of cell s at time t in the Turing machine. Of course, we need this to be error corrected, so we ask for the multilinear extension of this function. We check if this is consistent with the input. Then, given this function, we can compute an arithmetization of whether cell s at time t has an improper transition. Finally, we run a sum check on this arithmetization to see if it is 0 on all Boolean inputs.

This **PCP** is actually an **ePCP**, which can be converted into an **rPCPP**.

Lemma 5.4.1 (BFL style **ePCP**)**.** *Let L be any pair language with alphabet Σ where $\log(|\Sigma|)$ is a power of 2, first input length n and second input length n'. Let time $T = \Omega((n+n')\log(|\Sigma|))$ and space $S = \Omega(\log(T))$ be such that L is decided by a simultaneous time T and space S nondeterministic algorithm, A.*

Take any field $\mathbb{F}$. Then we have an **ePCP** *with:*

1. *Decision complexity $O((\log(T) + n)\,\mathsf{polylog}(|\mathbb{F}|))$.*
2. *Verifier space $O(\log(T)\log(|\mathbb{F}|))$.*
3. *Randomness $O(\log(T)\log(|\mathbb{F}|))$.*
4. *$O(\log(T))$ queries.*
5. *Alphabet $\mathbb{F}$.*
6. *Extrapolation time $O(\log(T)\,\mathsf{polylog}(|\mathbb{F}|))$.*
7. *Degree $O(\log(T))$ and $O(\log(T))$ variables.*
8. *Perfect completeness.*
9. *Low degree soundness $O\left(\frac{\log(T)^2}{|\mathbb{F}|}\right)$.*
10. *Log of proof length $O(\log(T)\log(|\mathbb{F}|))$.*
11. *$\rho = O(\log(|\Sigma|))$ proof symbols per implicit symbol.*

Further, if A is deterministic, then the prover space is $O(\log(T)\log(|\mathbb{F}|)+S)$.

Proof. First make sure the language is paddable. Since it will not effect the result, we assume we start with such a language, L, and such an explicit input x. Suppose L is decided by a two tape TM A running in time T and space S. By Lemma 5.2.1, A has a simulation by a one tape TM B with time T' polynomial in T and space S' linear in S.

Let x be our length n explicit input, let y be our implicit input, and w be (x, y) encoded for B. Recall that w can be written in three parts, $w^1, w^2_{x,y}, w^3$. Here w^1 is $O(\log(n))$ bits that don't depend on the specific input, just its length, $w^2_{x,y}$ is the actual encoding of x, y, and w^3 is the working space. Since $w^2_{x,y}$ is a simple encoding, we can further separate $w^2_{x,y}$ into w^2_x and w^2_y. Now pad x until $|(w^1, w^2_x)|$ is a power of two. We will need this so we can separate x into its own sub code. Now we will describe an honest prover for $(x, y) \in L$.

Let $K = 2^k$ be a constant power of two such that the states in B are encoded in $\{0,1\}^K$. Let s and t be the smallest integers so that $S' \le 2^s$ and $T' \le 2^t$. And note that if A is deterministic, then there is also a RAM algorithm C that can compute the bits of the computation history of B in time $O(T)$ and space $O(S)$.

If $(x, y) \in L$, then let $X' : \{0,1\}^s \times \{0,1\}^k \times \{0,1\}^t \to \{0,1\}$ be the function that outputs a valid computation history of B with the starting input being w. Then by Lemma 5.1.2 the multilinear extension of X', X, can be computed in space $O(\log(T)\log(|\mathbb{F}|)+S)$.

Then by Lemma 5.2.3, using X, we can compute $Y = \Gamma_B(X)$ that is constant degree, $d = O(1)$, in each variable individually, uses constantly many queries to X, and Y is 0 on all binary inputs (as well as the other properties listed in Lemma 5.2.3, which we later use to prove soundness). Let $m = 3s+2t+4K$. Then Y is a function $\mathbb{F}^m \to \mathbb{F}$. Abusing notation slightly, let $X : \mathbb{F}^m \to \mathbb{F}$ be X applied to the first $s+k+t$ variables.

Then by the completeness case of Lemma 5.3.1, if A is deterministic, then in space $O(\log(T)\log(|\mathbb{F}|)+S)$ the prover can compute the proof, f, for the sum check for Y. Let $f_i(x) = f(x, i)$.

For all $j \in [m+2]$, let $l^{m+2}_j : \mathbb{F} \to \mathbb{F}$ be the unique $m+1$ degree polynomial that is 1 at j, and 0 for all other $i \in [m+2]$. Then the proof for our **PCP** is supposed to be $\pi : \mathbb{F}^m \times \mathbb{F} \to \mathbb{F}$ where

$$\pi(z, i) = \left(\sum_{j \in [m+1]} l^{m+2}_j(i) f_j(z) \right) + l^{m+2}_{m+2}(i) X(z).$$

See that restricting $i \in [m+1]$ gives f_i, and for $i = m+2$ gives X.

Then we already showed how to compute X and f in space $O(\log(T)\log(|\mathbb{F}|)+S)$. Then we only need additional space to store a pointer to j (which requires only $O(\log(T))$ space), and to compute the interpolating polynomial l^{m+2}_j. Recall that

$$l^{m+2}_j(i) = \prod_{h \in [m+2] \setminus \{j\}} \frac{i-h}{j-h}.$$

Which can be straightforwardly computed with space for a constant number of field elements. So any symbol in π is computable in space $O(\log(T)\log(|\mathbb{F}|)+S)$.

Finally, π has constant degree in each of the first m variables, since X and each of the f_i do. And π has degree $O(m)$ in the last variable since each l^{m+2}_j has degree $O(m)$. This gives π a final degree of $d = O(m)$.

Now we describe the verifier. For a provided proof, π, we will infer the provided X, and f in the obvious way. The verifier runs a few checks for input x.

1. Sum check of f.

   Follow the verifier in the sum check protocol in Lemma 5.3.1.

2. Consistency of X with f_{m+1}.

   Let W be the set of w so that the sum check queries $f(w, m'+1)$. Sum check will only query constantly many of such w, since the degree of Y in individual variables is constant. So W has constant size.

   Then for $w \in W$, use Lemma 5.2.3 to calculate $Y(w) = \Gamma_B(X)(w)$. Then check if $f(w, m'+1) = Y(w)$.

3. Consistency of X with x.

   For input (x, y), it has transformed input $w = (w^1, w_x^2, w_y^2, w^3)$ as described in Lemma 5.2.1. As previously described, we padded x such that $|w^1| + |w_x^2|$ is a power of two. Now we realize that the multilinear extension of (w^1, w_x^2) (as a truth table) is a subset of X (as a truth table). So just compute the multilinear extension of (w^1, w_x^2) at a random point and compare it to X.

Now we show this has the desired properties. For reference, recall that $m = O(s + t + K) = O(\log(T))$.

1. Decision complexity $O((\log(T) + n)\,\mathsf{polylog}(|\mathbb{F}|))$ and verifier space $O(\log(T)\log(|\mathbb{F}|))$.

   The sum check protocol runs in time $O(md\,\mathsf{polylog}(|\mathbb{F}|)) = O(\log(T)\,\mathsf{polylog}(|\mathbb{F}|))$ and uses space $O(md\log(|\mathbb{F}|)) = O(\log(T)\log(|\mathbb{F}|))$ since the individual degree of the correct proof is $d = O(1)$.

   Checking consistency of X with f_{m+1} is only constantly many calculations of Y, which only takes time $O(\mathsf{polylog}(|\mathbb{F}|))$ and space $O(\log(|\mathbb{F}|))$.

   When checking the consistency of X with the input, we need to calculate the multilinear extension of (w^1, w_x^2). These can be thought of as a function from $\{0,1\}^{O(\log(n)}$ to $\{0,1\}$ that is very efficiently computable. By Lemma 5.1.2 this can be done in time $O(n\,\mathsf{polylog}(|\mathbb{F}|))$ and space $O(\log(n)+\log(|\mathbb{F}|))$.

2. Randomness $O(\log(T)\log(|\mathbb{F}|))$.

   The verifier needs to use $O(\log(T)\log(|\mathbb{F}|))$ bits to run the sum check, and to choose v.

3. $O(\log(T))$ queries.

   The sum check only takes $O(\log(T))$ queries, and there are only constantly many other queries.

4. Alphabet $\mathbb{F}$.

   From how we defined our **ePCP**.

5. Extrapolation time $O(\log(T)\,\mathsf{polylog}(|\mathbb{F}|))$.

   From Lemma 5.3.1, the sum check is time $O(\log(T)\,\mathsf{polylog}(|\mathbb{F}|))$ extrapolatable. There are only constantly many other queries, so all the other queries are trivially time $O(\log(T)\,\mathsf{polylog}(|\mathbb{F}|))$ extrapolatable. Since the query locations are just constantly many extrapolatable query locations, by Lemma 4.1.1, all together they are time $O(\log(T)\,\mathsf{polylog}(|\mathbb{F}|))$ extrapolatable.

6. Degree $O(\log(T))$ and $O(\log(T))$ variables. We already showed when describing an honest prover that we have degree $O(m)$ which is $O(\log(T))$ and by definition of the proof, it has $O(\log(T))$ variables.

7. Perfect completeness.

   Follows for $(x, y) \in L$ with an honest prover. Since $(x, y) \in L$, the prover provides the X that is the multilinear extension of the computation history from the proper w, so consistency with input passes. Similarly, f is honestly given to be consistent with Y. So the consistency between X, Y, and f passes. Finally, since X is a valid computation history, Y is 0 on all Boolean inputs, and degree $d = O(1)$ in each individual variable, so the sum check succeeds.

8. Low degree soundness $\frac{O(\log(T)^2)}{|\mathbb{F}|}$.

   Suppose $(x, y) \notin L$ and we are given a proof, π, such that π encodes y and is a polynomial with max total degree $d' = O(\log(T))$. Then X is a degree d function, and y is implicitly encoded in whatever w_y^2 is provided by the computation history.

   Now let $\hat{X} = \mathrm{MLB}(X)$, and X' be $\hat{X}$ restricted to binary inputs. Since $(x, y) \notin L$, either X' is an invalid computation history, or it does not start with state $w^1, w_{x,y}^2, w^3$.

   If X' does not start with state (w^1, w_x^2, z) for some z, then $\hat{X}$ restricted to time 0 and 0 for everything but the first N spaces is not the multilinear extension of (w^1, w_x^2). Thus neither is X. Then the probability they agree on a random point is at most $\frac{d}{|\mathbb{F}|}$. So the **ePCP** accepts with probability only $\frac{d}{|\mathbb{F}|}$.

If X' does start with state (w^1, w_x^2, z'), see that it must *either* actually start with $(w^1, w_{x,y}^2, z)$ for some z since y is implicitly defined by w_y^2 *or* z' does not start with a valid encoding of $w_{y'}^2$ for any y'. If z' does not start with a valid encoding of any $w_{y'}^2$, or if $z \neq w^3$, then X' must be an invalid computation history, by Lemma 5.2.1. On the other hand, if $z = w^3$, the computation history must be invalid since $(x, y) \notin L$. So then all we have left is the case that X' is not a valid computation history.

Suppose X' is an invalid computation history. Then $\Gamma_B(\hat{X})$ must not be 0 on all Boolean inputs. Then from Lemma 5.2.3, by contrapositive, $\Gamma_B(X)$ must not be 0 on all binary inputs.

Then by the soundness in Lemma 5.3.1, the sum check for $\Gamma_B(X)$ passes with probability at most

$$\frac{(d'+1)m}{|\mathbb{F}|} = O\left(\frac{\log(T)^2}{|\mathbb{F}|}\right).$$

So with probability at most $O\left(\frac{\log(T)^2}{|\mathbb{F}|}\right)$ do we accept.

9. Log of proof length $O(\log(T)\log(|\mathbb{F}|))$.

   This comes from the fact that the proof is a function with domain $\mathbb{F}^{m+1}$, and

$$\log(|\mathbb{F}^{m+1}|) = (m+1)\log(|\mathbb{F}|) = O(\log(T)\log(|\mathbb{F}|)).$$

10. Proof symbols per implicit symbol.

    See that the individual bits representing y are in w_y^2. This is simply because the symbol encoded for a bit of y must be different whether it is a 0 or a 1.

□

6 PCP of Proximity and Composition

Our **PCP** uses the standard technique of **PCP** composition [AS98; Ben+04; DR04; MR08; DH09] to reduce the number of queries. Recall that in **PCP** composition, we start with an outer **PCP** that is efficient, but makes too many queries. Now instead of running these queries and verifying them, we want a short proof that the outer **PCP** would accept those queries, using a proof that itself requires even fewer queries. That is, we want an "inner" **PCP** to prove we would accept the outer **PCP** without actually running the outer **PCP**. This is **PCP** composition is. In particular, we use composition between a "robust" outer **PCP** and an inner **PCP** of "proximity". See the preliminaries, Section 2.3, for details.

First, we use our **ePCP** to construct a **rPCPP**.

Corollary 6.0.1 (Our Robust **PCP** of Proximity)**.** *Let L be any pair language with alphabet Σ where $\log(|\Sigma|)$ is a power of 2, first input length n and second input length n'. Let time $T = \Omega((n+n')\log(|\Sigma|))$ and space $S = \Omega(\log(T))$ be such that L is decided by a simultaneous time T and space S nondeterministic algorithm, A.*

*For any constant robust soundness $\delta \leq \frac{1}{3}$, proximity η, and robustness parameter $\beta \leq \min\{\frac{\delta}{10}, \frac{\eta}{6}\}$ there is an **rPCPP** for L with*

1. *Decision complexity $\tilde{O}(n + \mathsf{poly}(\log(T|\Sigma|)/\beta))$.*
2. *Verifier space $O\left((\log(T|\Sigma|))\log\left(\frac{\log(T|\Sigma|)}{\beta}\right)\right)$.*
3. *Randomness $O\left(\frac{\log(T)\log(1/\delta)}{\eta}\log\left(\frac{\log(T|\Sigma|)}{\beta}\right)\right)$.*
4. *Query time $\tilde{O}\left(\frac{\log(T)}{\eta}\mathsf{polylog}(\log(|\Sigma|)/\beta)\right)$.*
5. *$O\left(\frac{\log(T)\log(T|\Sigma|)}{\beta^{O(1)}}\right)$ queries.*

6. *Alphabet* $\mathbb{F} \cup \Sigma$ *with* $|\mathbb{F}| = O\left(\frac{\log(T)\log(T|\Sigma|)}{\beta^{O(1)}}\right)$.

7. *Perfect completeness.*

8. *Proximity* η.

9. *Robust soundness error at most* δ.

10. *Robustness parameter* β.

Further, if A *is deterministic then the prover space is* $O\left(S + \log(T)\log\left(\frac{\log(T|\Sigma|)}{\beta}\right)\right)$.

Proof. Let τ be the constant from Lemma 4.2.2. Let c be a sufficiently large constant. Then choose some field $\mathbb{F}$ so that

$$|\mathbb{F}| \geq \max c\{\frac{\log(T)}{\delta}\left(\log(T) + \lceil\frac{\log(1/\delta)\log(|\Sigma|)}{\eta}\rceil\right), \log(T)\beta^{-1/\tau}\}$$

and $|\mathbb{F}| = O\left(\frac{\log(T)\log(T|\Sigma|)}{\beta^{O(1)}}\right)$ (in particular then $\log(|\mathbb{F}|) = O\left(\log\left(\frac{\log(T|\Sigma|)}{\beta}\right)\right)$.)

Then by Lemma 5.4.1 there is an **ePCP** with

1. Decision complexity $t = O((\log(T) + n)\,\mathsf{polylog}(|\mathbb{F}|))$.
2. Verifier space $s = O(\log(T)\log(|\mathbb{F}|))$.
3. Randomness $r = O(\log(T)\log(|\mathbb{F}|))$.
4. $q = O(\log(T))$ queries.
5. Alphabet $\mathbb{F}$.
6. Extrapolation time $t' = O(\log(T)\,\mathsf{polylog}(|\mathbb{F}|))$.
7. Degree $d = O(\log(T))$ and $m = O(\log(T))$ variables.
8. Perfect completeness.
9. Low degree soundness $\delta_1 = O\left(\frac{\log(T)^2}{|\mathbb{F}|}\right) \leq \frac{\delta}{10}$.
10. Log of proof length $O(\log(T)\log(|\mathbb{F}|))$.
11. $\rho = O(\log(|\Sigma|))$ proof symbols per implicit input symbol.

Further, if A is deterministic, then the prover space is $O(\log(T)\log(|\mathbb{F}|) + S)$.

Then by Theorem 4.3.2, setting (setting the β in Theorem 4.3.2 to 3β), there is an **rPCPP** for L with

1. Decision complexity $O(t + |\mathbb{F}|^3\rho\,\mathsf{polylog}(|\mathbb{F}|)) = \tilde{O}(n + \mathsf{poly}(\log(T|\Sigma|)/\beta))$.
2. Verifier space $O(s + \rho\log(|\mathbb{F}|)) = O\left((\log(T|\Sigma|))\log\left(\frac{\log(T|\Sigma|)}{\beta}\right)\right)$.
3. Randomness $r + O\left(\frac{m\log(1/\delta_1)}{\eta}\log(|\mathbb{F}|)\right) = O\left(\frac{\log(T)\log(1/\delta)}{\eta}\log\left(\frac{\log(T|\Sigma|)}{\beta}\right)\right)$.
4. Query time $O\left(t' + \left(q + \frac{m\log(1/\delta_1)}{\eta}\right)\mathsf{polylog}(|\mathbb{F}|)\right) = \tilde{O}\left(\frac{\log(T)}{\eta}\,\mathsf{polylog}(\log(|\Sigma|)/\beta)\right)$.
5. $O(|\mathbb{F}|) = O\left(\frac{\log(T)\log(T|\Sigma|)}{\beta^{O(1)}}\right)$ queries.
6. Alphabet $\mathbb{F}$, and perfect completeness.
7. Proximity η.
8. Robust soundness error at most $\delta_1 + 9\beta \leq \delta$.

9. Robustness parameter β.

Further the prover is the same as the prover for the **ePCP**, so if A is deterministic then the prover space is $O\left(S + \log(T)\log\left(\frac{\log(T|\Sigma|)}{\beta}\right)\right)$. $\square$

Now we just compose our **PCP** with itself to get our main theorem.

Theorem 6.0.2 (Verifier Time Efficient **rPCPP**)**.** *Let L be any pair language with alphabet Σ where $\log(|\Sigma|)$ is a power of 2, first input length n and second input length n^*. Let time $T = \Omega((n+n^*)\log(|\Sigma|))$ and space $S = \Omega(\log(T))$ be such that L is decided by a simultaneous time T and space S nondeterministic algorithm, A.*

*For any constant robust soundness $\delta \leq \frac{2}{3}$, proximity η, and robustness parameter $\beta \leq \min\{\frac{\delta}{120}, \frac{\eta}{36}\}$ there is an **rPCPP** for L with*

1. *decision complexity $\tilde{O}\left(n + \log(T)\log(1/\delta)\,\mathsf{poly}\left(\frac{\log(|\Sigma|)}{\beta}\right)\right)$.*
2. *$O\left(\frac{\log(n\log(T)|\Sigma|)^2}{\beta^{O(1)}}\right)$ queries.*
3. *alphabet size $O\left(\frac{\log(T|\Sigma|)^2}{\beta^{O(1)}} + |\Sigma|\right)$.*
4. *query time $\tilde{O}\left(\frac{\log(T)}{\beta}\,\mathsf{polylog}(\log(|\Sigma|)/\beta)\right)$.*
5. *robust soundness δ.*
6. *perfect completeness.*
7. *uses $O\left(\frac{\log(T)\log(\log(T))\log(1/\delta)}{\beta}\,\mathsf{polylog}\left(\frac{\log(|\Sigma|)}{\beta}\right)\right)$ bits of randomness.*
8. *has proximity η.*
9. *robustness parameter β.*

*Further, if A is deterministic, the **rPCPP** has prover space $\tilde{O}\left(S + \frac{\log(T)}{\eta}\,\mathsf{polylog}(\log(|\Sigma|)/\beta)\right)$.*

Proof. By Corollary 6.0.1, using δ as $\delta/2$, β as 6β and η as η, there is an **rPCPP** for L, call it C_1, with

1. Decision complexity $\tilde{O}(n + \mathsf{poly}(\log(T|\Sigma|)/\beta))$.
2. Verifier space $O\left((\log(T|\Sigma|))\log\left(\frac{\log(T|\Sigma|)}{\beta}\right)\right)$.
3. Randomness $r = O\left(\frac{\log(T)\log(1/\delta)}{\eta}\log\left(\frac{\log(T|\Sigma|)}{\beta}\right)\right)$.
4. Query time $t = \tilde{O}\left(\frac{\log(T)}{\eta}\,\mathsf{polylog}(\log(|\Sigma|)/\beta)\right)$.
5. $O\left(\frac{\log(T)\log(T|\Sigma|)}{\beta^{O(1)}}\right)$ queries.
6. Alphabet $\Sigma' = \Sigma \cup \mathbb{F}$ with $|\mathbb{F}| = O\left(\frac{\log(T)\log(T|\Sigma|)}{\beta^{O(1)}}\right)$.
7. Perfect completeness.
8. Proximity η.
9. Robust soundness error at most $\delta/2$.
10. Robustness parameter 6β.

Further, if A is deterministic then the prover space is $s = O\left(S + \log(T)\log\left(\frac{\log(T|\Sigma|)}{\beta}\right)\right)$.

Let L_2 be the pair language recognized by the verifier of C_1. See that the explicit input length for L_2 is the length of the initial input, plus the length of the randomness, which is $n' = n + r = O\left(n + \frac{\log(T)\log(1/\delta)}{\eta}\log\left(\frac{\log(T|\Sigma|)}{\beta}\right)\right)$. The implicit input length is the number of queries. Finally the algorithm recognizing L_2 is the verifier, which runs in time $T' = \tilde{O}(n + \mathsf{poly}(\log(T|\Sigma|)/\beta))$ and space $O\left((\log(T|\Sigma|))\log\left(\frac{\log(T|\Sigma|)}{\beta}\right)\right)$. See we can also bound the log of the time to recognize L_2 as $\log(T') = O\left(\log\left(n\frac{\log(T|\Sigma|)}{\beta}\right)\right)$, and the log of the alphabet size of L_2 as $\log(|\Sigma'|) = O(\log(|\Sigma|) + \log(\log(T|\Sigma|)/\beta)) = O(\log(|\Sigma|) + \log(T'))$.

By Corollary 6.0.1, using δ as $\delta/4$, β as β and η as 6β, there is an **rPCPP** for L_2, call it C_2 with

1. Decision complexity

$$\begin{aligned} t' =& \tilde{O}(n' + \mathsf{poly}(\log(T'|\Sigma'|)/\beta)) \\ =& \tilde{O}\left(n + \frac{\log(T)\log(1/\delta)}{\eta}\log\left(\frac{\log(T|\Sigma|)}{\beta}\right) + \mathsf{poly}\left(\frac{\log(n\log(T)\log(|\Sigma|)|\Sigma|/\beta)}{\beta}\right)\right) \\ =& \tilde{O}\left(n + \log(T)\log(1/\delta)\,\mathsf{poly}\left(\frac{\log(|\Sigma|)}{\beta}\right)\right). \end{aligned}$$

2. Verifier space

$$\begin{aligned} & O\left((\log(T'|\Sigma'|))\log\left(\frac{\log(T'|\Sigma'|)}{\beta}\right)\right) \\ =& O\left(\left(\log\left(n\frac{\log(T)}{\beta}|\Sigma|\right)\right)\log\left(\log\left(n\frac{\log(T)}{\beta}|\Sigma|\right)\right)\right). \end{aligned}$$

3. Randomness

$$\begin{aligned} r' =& O\left(\frac{\log(T')\log(1/\delta)}{\eta}\log\left(\frac{\log(T'|\Sigma'|)}{\beta}\right)\right) \\ =& O\left(\frac{\log(1/\delta)}{\beta}\log\left(n\frac{\log(T)}{\beta}\right)\log\left(\log\left(n\frac{\log(T)}{\beta}|\Sigma|\right)\right)\right) \\ =& O\left(\frac{\log(1/\delta)\log(T)}{\beta}\,\mathsf{polylog}\left(\frac{\log(|\Sigma|)}{\beta}\right)\right). \end{aligned}$$

4. Query time

$$\begin{aligned} t^* =& \tilde{O}\left(\frac{\log(T')}{\eta}\,\mathsf{polylog}(\log(|\Sigma'|)/\beta)\right) \\ =& \tilde{O}\left(\frac{\log(n\log(T))}{\beta}\,\mathsf{polylog}(\log(|\Sigma|))\right). \end{aligned}$$

5.

$$\begin{aligned} q' =& O\left(\frac{\log(T')\log(T'|\Sigma'|)}{\beta^{O(1)}}\right) \\ =& O\left(\frac{\log(n\log(T)\log(|\Sigma|))\log(n\log(T)|\Sigma|)}{\beta^{O(1)}}\right) \\ =& O\left(\frac{\log(n\log(T)|\Sigma|)^2}{\beta^{O(1)}}\right) \end{aligned}$$

queries.

6. Alphabet $\Sigma \cup \mathbb{F}'$ with

$$\begin{aligned}|\mathbb{F}'| =& O\left(\frac{\log(T')\log(T'|\Sigma'|)}{\beta^{O(1)}}\right)\\ =& O\left(\frac{\log(n\log(T)\log(|\Sigma|))\log(n\log(T)|\Sigma|)}{\beta^{O(1)}}\right)\\ =& O\left(\frac{\log(n\log(T)|\Sigma|)^2}{\beta^{O(1)}}\right).\end{aligned}$$

7. Perfect completeness.
8. Proximity 6β.
9. Robust soundness error at most $\delta/2$.
10. Robustness parameter β.

Further, since the verifier for C_1 is deterministic the prover space of C_2 is

$$\begin{aligned}s' =& O\left((\log(T'|\Sigma'|))\log\left(\frac{\log(T'|\Sigma'|)}{\beta}\right)\right)+\\ & O\left(\log\left(\frac{n\log(T)|\Sigma|}{\beta}\right)\log\left(\frac{\log(n\log(T)\log(|\Sigma|)/\beta)\log(|\Sigma|)}{\beta}\right)\right)\\ =& O\left(\log(nT)\log\left(\frac{\log(nT)}{\beta}\right)+\mathsf{polylog}(|\Sigma|/\beta)\right).\end{aligned}$$

Then by Theorem 2.3.8 L has an **rPCPP** system for L, call it C_3, such that C_3 has:

1. the decision complexity of C_2: $O(t') = \tilde{O}\left(n+\log(T)\log(1/\delta)\,\mathsf{poly}\left(\frac{\log(|\Sigma|)}{\beta}\right)\right)$.
2. the number of queries as C_2: $O(q') = O\left(\frac{\log(n\log(T)|\Sigma|)^2}{\beta^{O(1)}}\right)$.
3. the alphabet size of C_2, C_1, and Σ:

$$\begin{aligned}& O\left(\frac{\log(T)\log(T|\Sigma|)}{\beta^{O(1)}}+\frac{\log(n\log(T)|\Sigma|)^2}{\beta^{O(1)}}+|\Sigma|\right)\\ =& O\left(\frac{\log(T|\Sigma|)^2}{\beta^{O(1)}}+|\Sigma|\right).\end{aligned}$$

4. the query time of C_1 plus C_2:

$$\begin{aligned}t+t^* =& \tilde{O}\left(\frac{\log(T)}{\eta}\,\mathsf{polylog}(\log(|\Sigma|)/\beta)\right)+\tilde{O}\left(\frac{\log(n\log(T))}{\beta}\,\mathsf{polylog}(\log(|\Sigma|))\right)\\ =& \tilde{O}\left(\frac{\log(T)}{\beta}\,\mathsf{polylog}(\log(|\Sigma|)/\beta)\right).\end{aligned}$$

5. the robust soundness of C_1 plus C_2: $\delta/2+\delta/2=\delta$.
6. perfect completeness.
7. the number of bits of randomness as C_1 plus C_2:

$$\begin{aligned}r+r' =& O\left(\frac{\log(T)\log(1/\delta)}{\eta}\log\left(\frac{\log(T)\log(|\Sigma|)}{\beta}\right)\right)+O\left(\frac{\log(1/\delta)\log(T)}{\beta}\,\mathsf{polylog}\left(\frac{\log(|\Sigma|)}{\beta}\right)\right)\\ =& O\left(\frac{\log(T)\log(\log(T))\log(1/\delta)}{\beta}\,\mathsf{polylog}\left(\frac{\log(|\Sigma|)}{\beta}\right)\right).\end{aligned}$$

8. the proximity of C_1: η.

9. the robustness parameter of C_2: β.

Further, if A is deterministic, then C_3 has prover space of C_1 plus query time of C_1 plus prover space of C_2:

$$\begin{aligned}s + t + s' =& O\left(S + \log(T)\log\left(\frac{\log(T|\Sigma|)}{\beta}\right)\right) + \\ & \tilde{O}\left(\frac{\log(T)}{\eta}\,\mathsf{polylog}(\log(|\Sigma|)/\beta)\right) + \\ & O\left(\log(nT)\log\left(\frac{\log(nT)}{\beta}\right) + \mathsf{polylog}(|\Sigma|/\beta)\right) \\ =& \tilde{O}\left(S + \frac{\log(T)}{\eta}\,\mathsf{polylog}(\log(|\Sigma|)/\beta)\right).\end{aligned}$$

□

Then our main PCP results, Theorem 1.1.3 and Theorem 1.1.4, come as special cases where $\eta = 1$, β is small constant, and the implicit input is empty (thus its alphabet is constant).

Remark (Better PCPs and More Composition)**.** *We believe other* **PCP***s, such as those of [Ben+04; DR04; MR08; DH09], could also be given tighter analysis and modified slightly to be similarly efficient. We used the simplest* **PCP** *that could achieve our parameters. One could also achieve even fewer queries by doing more rounds of composition, but one needs a more fine grained analysis of the decision complexity.*

A prior version of this paper achieved fewer queries than this version. This is due to it using different constructions for the **PCP** *of proximity and robust* **PCP***. In particular the* **PCP** *of proximity had a non robust construction that allowed it to be more time and query efficient. However, to make the paper simpler, this version of the paper uses a single* **PCP** *construction.*

6.1 Fine Grained MIP = NEXP

For completeness, we also prove our fine grained equivalence between **MIP** and **NEXP**: Corollary 1.1.5. A two prover, one round **MIP** is equivalent to a two query **PCP**. The main difference is the conventional parameter regime. For **PCP**s one is often concerned about proof length and alphabet size, and for **MIP**s one is often concerned about verifier time or prover time.

There is a standard way to convert a robust **PCP** to a two prover, one round **MIP**. That is to choose the randomness for the **PCP**, ask one prover for everywhere the proof would be queried on that choice of randomness, and the other prover for a single, random location the proof would be queried.

Corollary 1.1.5 (Fine Grained Equivalence of **MIP** = **NEXP**)**.** *For any time constructible function $p(n) = \Omega(n)$, language $L \in$* **NTIME**$[2^{\tilde{O}(p(n))}]$ *if and only if there is a two prover, one round* **MIP** *protocol for L whose verifier runs in time $\tilde{O}(p(n))$.*

Proof. First, there is a simple nondeterministic algorithm for a language with an **MIP** protocol: nondeterministically guess the entire prover strategy, then run the verifier for every choice of randomness on that strategy. So any language L with an **MIP** protocol with a time $\tilde{O}(p(n))$ verifier has a nondeterministic algorithm running in time $2^{\tilde{O}(p(n))}$.

For a language $L \in$ **NTIME**$[2^{\tilde{O}(p(n))}]$, we want to use Theorem 6.0.2, so we first convert L to a pair language where we just force the second input to be a single 0 (just so we can apply the theorem). Then set $\delta = \frac{2}{3}$ and $\eta = 1$ and $\beta = \frac{1}{180}$. Then by Theorem 6.0.2, we get a **PCP** for L with

- decision complexity $\tilde{O}(n + \mathsf{polylog}(p(n))) = \tilde{O}(p(n))$.
- $O(\log(p(n)))$ queries.
- Query time $\tilde{O}(p(n))$.

- Robust Soundness $\delta = \frac{2}{3}$ and robustness parameter $\beta = \frac{1}{180}$.
- Perfect completeness.
- $\tilde{O}(p(n))$ bits of randomness.

Instead of running this **PCP**, our verifier in our **MIP** protocol chooses the randomness, r, for the **PCP** and sends r to the first prover. The **MIP** verifier expects the prover to send what the **PCP** prover would send for all queries from r. The **MIP** verifier then checks that the **PCP** verifier would accept what it was sent. Next the **MIP** verifier chooses a random one of the queries the **PCP** verifier would have made for randomness r and sends that query to the second prover. Then the **MIP** verifier checks if the second prover gave a symbol that agrees with the first prover. The **MIP** verifier accepts if both tests pass.

The completeness is clear by construction. For soundness, see that there must be some optimal strategy for player 2, and that strategy corresponds to some proof. Suppose that $x \notin L$. Then by robustness, with probability $\frac{1}{3}$, the verifier will choose a randomness that either the first prover's proof will be rejected or the second provers proof will fail with probability $\frac{1}{180}$. Thus the verifier will reject with probability at least $\frac{1}{3}\frac{1}{180} = \frac{1}{540}$. So the **MIP** has constant soundness $\frac{539}{540} < 1$.

The verifier only needs time $\tilde{O}(p(n))$ to choose randomness and send it to the first prover, time $\tilde{O}(p(n))$ to decide if the proof is valid, and $\tilde{O}(p(n))$ time to make the query to the second prover. Thus this **MIP** verifier runs in time $\tilde{O}(p(n))$. □

Unfortunately, our **PCP** has only a small constant robustness, so this protocol alone has a soundness error close to one. However we note that to improve the soundness error, one can use a technique from **MIP**s called parallel repetitions [Raz98; Hol07; Rao08].

7 Open Problems

There are several ways we would like to improve the circuit lower bounds.

1. Remove the advice bit.

   We still had to use advice, a limitation from the original Santhanam result. It would be nice if we could get lower bounds on **MA** with no non-uniformity.

2. Prove tight bounds for all k.

   Another limitation of our circuit lower bound is that it does not prove this tight bound for all $k > 1$, just for some k.

   The major barrier is in the case that $\mathbf{SPACE}[n]$ algorithms may require super linear, but polynomial, sized circuits. Then the circuit size required for any given space may change in a strange way. For example, suppose for some $a > 1$

   $$\mathbf{SPACE}[n] \subseteq \mathbf{SIZE}[O(n^a)] \setminus \mathbf{SIZE}[o(n^a)].$$

   What we would like, but this does not obviously imply, is that for all $b > 1$:

   $$\mathbf{SPACE}[n^b] \subseteq \mathbf{SIZE}[O(n^{ab})] \setminus \mathbf{SIZE}[o(n^{ab})].$$

   While a padding argument gives $\mathbf{SPACE}[n^b] \subseteq \mathbf{SIZE}[O(n^{ab})]$, it does not give $\mathbf{SPACE}[n^b] \not\subseteq \mathbf{SIZE}[o(n^{ab})]$. We may even have something weird, like

   $$\mathbf{SPACE}[n^a] \subseteq \mathbf{SIZE}[O(n^a)] \setminus \mathbf{SIZE}[o(n^a)].$$

   That is, even if space n algorithms require circuit size n^a, we may not need larger circuits until our algorithms use more space than n^a.

   In this case, to get circuit lower bounds greater than n^a, we need to use an algorithm with space greater than n^a. Unfortunately, our verifier uses queries to the prover of the same length as the space of the algorithm being verified. It seems like the prover needs to use the space of the original problem, n^a, which is linear in its input length, n^a. Now since the prover has a longer, length n^a input, it may require size $(n^a)^a = n^{a^2}$ circuits.

One way to try to solve this problem is to show that if

$$\mathbf{SPACE}[n] \subseteq \mathbf{SIZE}[O(n^a)] \setminus \mathbf{SIZE}[o(n^a)]$$

for some $a > 1$, then for all $b > 1$:

$$\mathbf{SPACE}[n^b] \subseteq \mathbf{SIZE}[O(n^{ab})] \setminus \mathbf{SIZE}[o(n^{ab})].$$

This seems plausible, but hard to prove.

Another direction is to find an efficient **PCP** for $\mathbf{SPACE}[n^b]$ with prover queries shorter than n^b (or equivalently, proof length less than 2^{n^b}). But this seems hard as shorter **PCP** proofs imply more efficient algorithms.

For instance, for constant c, if L has a **PCP** with polynomial time verifier and proof length 2^{n^c}, then $L \in \mathbf{MATIME}[O(2^{n^c})]$ just by guessing the whole proof string, and verifying it. So if every language in $\mathbf{NTIME}[O(2^{n^b})]$ had a **PCP** with proof length $O(2^{n^c})$, then we would have

$$\mathbf{NTIME}[O(2^{n^b})] \subseteq \mathbf{MATIME}[O(2^{n^c})].$$

If $c < b$, this would contradict a derandomization conjecture that

$$\mathbf{MATIME}[f(n)] \subseteq \mathbf{NTIME}[\mathsf{poly}(f(n))].$$

Thus any more efficient **PCP** either must not apply to nondeterministic algorithms (ours does), or **MA** cannot be efficiently derandomized. This does not rule out this approach, but is a major challenge.

3. Make lower bound more frequent.

   Another direction is improving the infinitely often separation to a more frequently often separation. Murray and Williams [MW18] gave a refinement of the Santhanam circuit lower bounds that is incomparable to ours. In it they proved that for some $L \in \mathbf{MA}/O(\log(n))$ and constant c, for almost every n, either L on length n inputs wouldn't have circuits with size n^k, or L on length n^{ck} inputs wouldn't have circuits with size $n^{c^2k^2}$. One might want to strengthen their results.

   We conjecture that there exists some constant $k > 1$, function $f(n) = o(1)$, gap function $g(n) = \mathsf{poly}(n)$, and language $L' \in \mathbf{MATIME}[O(n^{k+f(n)})]/O(\log(n))$ such that for all n there is some $m \in [n, g(n)]$ such that language L' on length m inputs does not have circuits of size m^k.

   The Murray and Williams result produces a language L that for every input length n will either be the downward self reducible language from Santhanam's result (Y in Lemma 3.2.1), or a circuit found with exhaustive search (like in Lemma 3.3.1). If the prover circuit for the exhaustive search is small enough, then L is exhaustive search. Otherwise, L is (possibly padded) Y.

   The idea is that if exhaustive search on length n inputs doesn't have small prover circuits, then for the input length of the prover circuits, we have a hard problem (specifically, Y). Unfortunately, provers have input length about n^{ck} for some constant c. For that prover to be hard enough for our circuit lower bound, length n^{ck} inputs must require size n^{ck^2} circuits. So to make sure the provers are hard enough, length n inputs for exhaustive search may have to use prover circuits as large as n^{ck^2}!

   Our **PCP** can improve the constant c in the Murray and Williams result, but improving the approximately n^{k^2} verifier time to near n^k requires new ideas.

4. Prove exponential lower bounds for **MAEXP**.

   A similar problem is to prove exponential circuit lower bounds for the exponential version of **MA**, known as **MAEXP**. The best circuit lower bounds known for **MAEXP** are "half-exponential" by

Miltersen, Vinodchandran, and Watanabe [MVW99]. Loosely, a function is half exponential if that function composed with itself is exponential.

The bottleneck to proving such a lower bound is not a better **PCP**. This is a similar problem to proving tight bounds for all k, but pushed beyond polynomials. The issue is related to the win-win argument used in these results. Either **PSPACE** has less then half exponential sized circuits, then we can use those circuits in **MA** protocols as provers for **PCP**s of half exponential space algorithms. Or **PSPACE** does not have half exponential sized circuits, and exponential time protocols can solve these problems thus also don't have half exponential sized circuits. It isn't clear how to use a similar argument to get better circuit lower bounds.

One potential approach is to use an iterated win-win argument. Such an approach was used by Chen, Li, and Liang [CLL24] to show better circuit lower bounds for a related, more powerful complexity class: $\mathbf{AMEXP}/2^{n^{\epsilon}}$. Still, improving the circuit lower bounds for **MAEXP** remains open.

One could also look to improve our **PCP**. In particular, one could try to replicate other existing results while maintaining the $\tilde{O}(n + \log(T))$ runtime. Standard techniques can reduce the number of queries, or improve the soundness. We suspect these techniques can be used to give $\mathsf{poly}(T)$ proof length. Can we construct **PCP**s with length $\tilde{O}(T)$ proofs while having a $\tilde{O}(n+\log(T))$ verifier runtime? Known **PCP**s with proof length $\tilde{O}(T)$ have decision complexity $\Omega(n + \log(T)^2)$.

In particular, we suspect ideas from theorem 2.6 in [Ben+05][10] (which extends the results of [Ben+04] from **NP** to **NEXP**) may give a **PCP** with decision complexity near $\tilde{O}(n + \log(T))$ while giving a proof of length $T^{1+o(1)}$. But it would not have proof length $\tilde{O}(T) = T\,\mathsf{polylog}(T)$. We stress that such an analysis has not been performed, and the **PCP** of [Ben+05] is much more complex than ours.

References

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

[ACR97] Alexander E. Andreev, Andrea E. F. Clementi, and José D. P. Rolim. "Optimal bounds for the approximation of boolean functions and some applications". In: *Theor. Comput. Sci.* 180.1–2 (June 1997), 243–268. ISSN: 0304-3975. DOI: 10.1016/S0304-3975(96)00217-4. URL: https://doi.org/10.1016/S0304-3975(96)00217-4.

[AS97] Sanjeev Arora and Madhu Sudan. "Improved Low-Degree Testing and Its Applications". In: *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing.* STOC '97. El Paso, Texas, USA: Association for Computing Machinery, 1997, 485–495. ISBN: 0897918886. DOI: 10.1145/258533.258642. URL: https://doi.org/10.1145/258533.258642.

[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 (May 1998), 501–555. ISSN: 0004-5411. DOI: 10.1145/278298.278306. URL: https://doi-org.ezproxy.lib.utexas.edu/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.

[10] We emphasize this is the theorem labeled "Efficient PCPPs with small query complexity", which does not have quasi-linear proof length.

[BV14] Eli Ben-Sasson and Emanuele Viola. “Short PCPs with Projection Queries”. In: *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I.* Ed. by Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias. Vol. 8572. Lecture Notes in Computer Science. Springer, 2014, pp. 163–173. DOI: `10.1007/978-3-662-43948-7\_14`. URL: `https://doi.org/10.1007/978-3-662-43948-7\_14`.

[Ben+04] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. “Robust Pcps of Proximity, Shorter Pcps and Applications to Coding”. In: *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing.* STOC ’04. Chicago, IL, USA: Association for Computing Machinery, 2004, 1–10. ISBN: 1581138520. DOI: `10.1145/1007352.1007361`. URL: `https://doi.org/10.1145/1007352.1007361`.

[Ben+05] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. “Short PCPs verifiable in polylogarithmic time”. In: *20th Annual IEEE Conference on Computational Complexity (CCC’05).* 2005, pp. 120–134. DOI: `10.1109/CCC.2005.27`.

[Ben+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. “On the Concrete Efficiency of Probabilistically-Checkable Proofs”. In: *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing.* STOC ’13. Palo Alto, California, USA: Association for Computing Machinery, 2013, 585–594. ISBN: 9781450320290. DOI: `10.1145/2488608.2488681`. URL: `https://doi.org/10.1145/2488608.2488681`.

[CLL24] Lijie Chen, Jiatu Li, and Jingxun Liang. “Maximum Circuit Lower Bounds for Exponential-time Arthur Merlin”. In: *Proceedings of the 57th Annual ACM Symposium on Theory of Computing.* STOC 2025. Prague, Czech Republic, 2024.

[Coo72] Stephen A. Cook. “A Hierarchy for Nondeterministic Time Complexity”. In: *Proceedings of the Fourth Annual ACM Symposium on Theory of Computing.* STOC ’72. Denver, Colorado, USA: Association for Computing Machinery, 1972, 187–192. ISBN: 9781450374576. DOI: `10.1145/800152.804913`. URL: `https://doi.org/10.1145/800152.804913`.

[DH09] Irit Dinur and Prahladh Harsha. “Composition of Low-Error 2-Query PCPs Using Decodable PCPs”. In: *2009 50th Annual IEEE Symposium on Foundations of Computer Science.* 2009, pp. 472–481. DOI: `10.1109/FOCS.2009.8`.

[DR04] Irit Dinur and Omer Reingold. “Assignment testers: towards a combinatorial proof of the PCP-theorem”. In: *45th Annual IEEE Symposium on Foundations of Computer Science.* 2004, pp. 155–164. DOI: `10.1109/FOCS.2004.16`.

[Dor+20] Dean Doron, Dana Moshkovitz, Justin Oh, and David Zuckerman. “Nearly Optimal Pseudorandomness From Hardness”. In: *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020.* IEEE, 2020, pp. 1057–1068.

[FM05] Gudmund Skovbjerg Frandsen and Peter Bro Miltersen. “Reviewing Bounds on the Circuit Size of the Hardest Functions”. In: *Inf. Process. Lett.* 95.2 (2005), 354–357. ISSN: 0020-0190.

[FS11] Lance Fortnow and Rahul Santhanam. “Robust Simulations and Significant Separations”. In: *Proceedings of the 38th International Colloquim Conference on Automata, Languages and Programming - Volume Part I.* ICALP’11. Zurich, Switzerland: Springer-Verlag, 2011, 569–580. ISBN: 9783642220050.

[FS95] Katalin Friedl and Madhu Sudan. “Some Improvements to Total Degree Tests”. In: *In Proceedings of the 3rd Annual Israel Symposium on Theory of Computing and Systems.* 1995, pp. 190–198.

[FST05] Lance Fortnow, Rahul Santhanam, and Luca Trevisan. “Hierarchies for Semantic Classes”. In: *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing.* STOC ’05. Baltimore, MD, USA: Association for Computing Machinery, 2005, 348–355. ISBN: 1581139608. DOI: `10.1145/1060590.1060642`. URL: `https://doi.org/10.1145/1060590.1060642`.

[FSW09] Lance Fortnow, Rahul Santhanam, and Ryan Williams. “Fixed-Polynomial Size Circuit Bounds”. In: *2009 24th Annual IEEE Conference on Computational Complexity.* 2009, pp. 19–26. DOI: `10.1109/CCC.2009.21`.

[Fis74] Michael Fischer. "Lecture Notes on Network Complexity". In: *Yale Technical Report 1104* (1974).

[HR18] Justin Holmgren and Ron Rothblum. "Delegating Computations with (Almost) Minimal Time and Space Overhead". In: *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*. 2018, pp. 124–135. DOI: `10.1109/FOCS.2018.00021`.

[HS65] J. Hartmanis and R. E. Stearns. "On the Computational Complexity of Algorithms". In: *Transactions of the American Mathematical Society* 117 (1965), pp. 285–306. ISSN: 00029947. URL: `http://www.jstor.org/stable/1994208`.

[HS66] F. C. Hennie and R. E. Stearns. "Two-Tape Simulation of Multitape Turing Machines". In: *J. ACM* 13.4 (1966), 533–546. ISSN: 0004-5411. DOI: `10.1145/321356.321362`. URL: `https://doi.org/10.1145/321356.321362`.

[Har+24] Prahladh Harsha, Mrinal Kumar, Ramprasad Saptharishi, and Madhu Sudan. "An Improved Line-Point Low-Degree Test*". In: *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*. 2024, pp. 1883–1892. DOI: `10.1109/FOCS61266.2024.00113`.

[Hol07] Thomas Holenstein. "Parallel Repetition: Simplifications and the No-Signaling Case". In: *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*. STOC '07. San Diego, California, USA: Association for Computing Machinery, 2007, 411–419. ISBN: 9781595936318. DOI: `10.1145/1250790.1250852`. URL: `https://doi.org/10.1145/1250790.1250852`.

[KRR21] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. "How to Delegate Computations: The Power of No-Signaling Proofs". In: *J. ACM* 69.1 (Nov. 2021). ISSN: 0004-5411. DOI: `10.1145/3456867`. URL: `https://doi.org/10.1145/3456867`.

[LW12] Richard J. Lipton and Ryan Williams. "Amplifying Circuit Lower Bounds against Polynomial Time with Applications". In: *2012 IEEE 27th Conference on Computational Complexity*. 2012, pp. 1–9. DOI: `10.1109/CCC.2012.44`.

[Lun+92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. "Algebraic Methods for Interactive Proof Systems". In: *J. ACM* 39.4 (Oct. 1992), 859–868. ISSN: 0004-5411. DOI: `10.1145/146585.146605`. URL: `https://doi.org/10.1145/146585.146605`.

[Lup58] O. B. Lupanov. "The synthesis of contact circuits". Russian. In: *Dokl. Akad. Nauk SSSR* 119 (1958), pp. 23–26. ISSN: 0002-3264.

[MP06] D. van Melkebeek and K. Pervyshev. "A generic time hierarchy for semantic models with one bit of advice". In: *21st Annual IEEE Conference on Computational Complexity (CCC'06)*. 2006, 14 pp.–144. DOI: `10.1109/CCC.2006.7`.

[MR08] Dana Moshkovitz and Ran Raz. "Two Query PCP with Sub-Constant Error". In: *2008 49th Annual IEEE Symposium on Foundations of Computer Science*. 2008, pp. 314–323. DOI: `10.1109/FOCS.2008.60`.

[MVW99] Peter Bro Miltersen, N. V. Vinodchandran, and Osamu Watanabe. "Super-Polynomial versus Half-Exponential Circuit Size in the Exponential Hierarchy". In: *Proceedings of the 5th Annual International Conference on Computing and Combinatorics*. COCOON'99. Tokyo, Japan: Springer-Verlag, 1999, 210–220. ISBN: 3540662006.

[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`.

[Mei09] Or Meir. "Combinatorial PCPs with Efficient Verifiers". In: *2009 50th Annual IEEE Symposium on Foundations of Computer Science*. 2009, pp. 463–471. DOI: `10.1109/FOCS.2009.10`.

[PF79] Nicholas Pippenger and Michael J. Fischer. "Relations Among Complexity Measures". In: *J. ACM* 26.2 (1979), 361–381. ISSN: 0004-5411. DOI: `10.1145/322123.322138`. URL: `https://doi.org/10.1145/322123.322138`.

[PS94] Alexander Polishchuk and Daniel A. Spielman. “Nearly-linear size holographic proofs”. In: *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*. STOC ’94. Montreal, Quebec, Canada: Association for Computing Machinery, 1994, 194–203. ISBN: 0897916638. DOI: 10.1145/195058.195132. URL: https://doi.org/10.1145/195058.195132.

[Pip77] Nicholas Pippenger. “Fast simulation ofA combinational logic networks by machines without random-access storage”. In: *Proc. 15th Allerton Conference on Communication, Control, and Computing*. 1977, pp. 25–33.

[Rao08] Anup Rao. “Parallel Repetition in Projection Games and a Concentration Bound”. In: *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*. STOC ’08. Victoria, British Columbia, Canada: Association for Computing Machinery, 2008, 1–10. ISBN: 9781605580470. DOI: 10.1145/1374376.1374378. URL: https://doi.org/10.1145/1374376.1374378.

[Raz98] Ran Raz. “A Parallel Repetition Theorem”. In: *SIAM Journal on Computing* 27.3 (1998), pp. 763–803. DOI: 10.1137/S0097539795280895. eprint: https://doi.org/10.1137/S0097539795280895. URL: https://doi.org/10.1137/S0097539795280895.

[SFM78] Joel I. Seiferas, Michael J. Fischer, and Albert R. Meyer. “Separating Nondeterministic Time Complexity Classes”. In: *J. ACM* 25.1 (1978), 146–167. ISSN: 0004-5411. DOI: 10.1145/322047.322061. URL: https://doi.org/10.1145/322047.322061.

[SK12] Jayalal Sarma and Dinesh K. *CS6840: Advanced Complexity Theory, Lecture 35 : Size and Depth complexity of Boolean Circuits*. 2012. URL: https://deeplearning.cs.cmu.edu/S22/document/readings/booleancircuits_shannonproof.pdf.

[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.

[Sha49] Claude. E. Shannon. “The synthesis of two-terminal switching circuits”. In: *The Bell System Technical Journal* 28.1 (1949), pp. 59–98. DOI: 10.1002/j.1538-7305.1949.tb03624.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.

[Weg87] Ingo Wegener. *The complexity of Boolean functions*. 1987. URL: https://eccc.weizmann.ac.il/static/books/The_Complexity_of_Boolean_Functions/.

[Wil11] Ryan Williams. “Non-uniform ACC Circuit Lower Bounds”. In: *2011 IEEE 26th Annual Conference on Computational Complexity*. 2011, pp. 115–125. DOI: 10.1109/CCC.2011.36.

[Žá83] Stanislav Žák. “A Turing machine time hierarchy”. In: *Theoretical Computer Science* 26.3 (1983), pp. 327–333. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(83)90015-4. URL: https://www.sciencedirect.com/science/article/pii/0304397583900154.

A Automata Proofs

In this section, we show how to construct the single tape TM for an algorithm, how to arithmetize a formula for its rules, and how to simulate it efficiently. To do this we solve the problem for a more general RAM algorithm. The result for a two tape TM, Lemma 5.2.1 is a straightforward corollary.

Here, we will use the word RAM model for a RAM algorithm. This model, for a time T algorithm, has a constant number of $O(\log(T))$ bit registers and a uniform instruction set on those registers, and a constant sized program that it runs. At any time step, the program can carry out one instruction, including reading or writing the value of one register to the location in memory of another register, standard register to register operations (like addition, setting a register to one or zero, bit shifting, etc), and perform basic checks on the data in registers to control the flow of the program (if statements). However, our RAM algorithm is *not* allowed to write to the read only portion of memory, or read from the write only portion of memory. We note that the exact instructions of the RAM algorithm are unimportant as long as they can be performed by a uniform Turing machine.

It is well known that a RAM algorithm can both time and space efficiently simulate a Turing machine (in the obvious way). Similarly, a multi-tape Turing machine can *space* efficiently simulate a RAM algorithm (with a polynomial overhead in the time). Further, there is a *single* tape Turing machine simulating RAM algorithms, however, this single tape Turing machine is *not* space efficient. We show that even though the single tape Turing machine is inefficient, it is predictable so a RAM algorithm running in a similar space to the initial RAM algorithm can print the computation history of the single tape simulation.

Lemma A.0.1 (RAM algorithms have simple Turing machines)**.** *Let A be a nondeterministic RAM algorithm recognizing L, running in time T and space S where $S = \Omega(\log(n))$ and $T = \Omega(S)$. Further, A uses input coming from a read only space of n bits.*

Then there is a one tape, one dimensional Turing machine, B, simulating A, such that

1. *B runs in time $T' = \mathsf{poly}(T, n)$, and space $S' = O(n + S)$.*
2. *B has a constant size alphabet, Σ, where for some k, we have $|\Sigma| = 2^{2^k}$. That is, Σ is represented by a power of 2 number of bits. Further, the alphabet Σ includes whether the head is currently on that cell in the work tape.*
3. *For any input x for A, there is a corresponding input for B, y_x, of length S'. And we also have that $y_x = (y^1, y^2_x, y^3)$ where*
   (a) *y^1 has length $O(\log(S'))$ and is independent of the specific x, only the length of x, and y^1 is computable in time $O(|y^1|)$.*
   (b) *y^2_x is exactly n symbols where for some $f : \{0, 1\} \to \Sigma$, for each $i \in [n]$, $(y^2_x)_i = f(x_i)$, where f is computable in constant time.*
   (c) *y^3 is exactly S copies of a specific symbol in Σ.*
4. *Not all transitions for B will be defined, and A accepts on x if and only if after time T' starting on y_x, B reaches a steady state. Similarly, A rejects on x if and only if there is no sequence of T' valid transitions in B starting from y_x.*
5. *If B has a starting state that is (y^1, z) for any z that is not (y^2_x, y^3) for some $x \in L$, then B will not have T' valid transitions.*

If A is a deterministic, then B is deterministic and the following holds. Let $x \in L$ be an input for A, with transformed input for B, y_x. Given a time $t \in [T']$ and a memory location $s \in [S']$, there is a RAM algorithm C that can compute the symbol in cell s at time t in B's computation history on y_x in time $O(T)$ and space $O(S)$ given read only access to x.

Proof. The idea is as follows. First convert the RAM machine into an input oblivious, single tape Turing machine, B', where there is $O(\log(S + n))$ space for the registers, n space for the read only input, followed by S working space reserved on the tape. Notably, this input oblivious Turing machine may temporarily modify the contents of this read only space. In fact, it needs to. But these will always be temporary since we are simulating a RAM machine where these are read only. On accepting, the state will remain constant. On rejecting, there will just be an undefined transition. This makes accepting equivalent to the existence of a valid computation history.

For more details, first, we take our input RAM algorithm A, and make a new RAM algorithm A' that does the same thing, but starts by making sure its working space is all 0.

Turing machine B' can be made from A' by first adding $O(\log(S'))$ bits before the first bit to hold the current memory configuration of the registers. Then the Turing machine starts at the beginning, goes through the motions it would need to do on any register to register operation and any state change. Then it goes from the beginning forward, looking for the index it wants to operate on for any register memory operation.

Each time it moves, it copies the bit in front of the head behind it, and shifts all its registers 1 forward. At each potential bit, it moves the tape head as if it was going to do every operation, but doesn't actually do it unless the indexes match for a register memory operation. Once it gets to the far side, it returns the registers to the start.

This Turing machine runs in time $T' = O(T(S+n)\log(S')^2)$ and only needs size $S' = O(S+n)$ to hold bits of the computation, and the registers. In particular, at time 0, it has $O(\log(S+n))$ space reserved for the registers, exactly n cells reserved for the read only input, and S cells reserved for the working space. Then y^1 will contain the register states, y^2_x will be the read-only input, and y^3 will be the working space.

We need to do one more thing to B' to get B. Before we start, we want to verify the format. B will do this by sweeping from the beginning to the end and back, making sure each symbol is of the appropriate format. This will tell us that the provided y^2_x and y^3 encode binary inputs. Such a procedure will work if y^1 is correct, specifically that it contains the correct starting head for B'. Then after that, the simulation of A' will further make sure y^3 actually encodes all 0.

Turing machine B does this by having a special sweeping state that sweeps from left to right that makes sure it only encounters binary inputs, and then sweeps back to the beginning where it turns into the head for the Turing machine B' simulating A'.

For a time t and a space s, the state of cell s in the history of B at time t can be computed by an algorithm C which does the following:

1. If the time t is in the preprocessing part of B before B' starts, we can just return the bit from y_x directly, if it is after the head, the head if it is on the head, or the bit from y_x activated if it is before.

   If t is not in the time for preprocessing, just subtract the amount of time to do the preprocessing from t and move on.

2. A time t in B is part of the simulation of some step at some time t' in A'. Since the Turing machine is input oblivious, we know exactly how many operations a step in A' is in B and can calculate t'. Run A up to time t'.

3. Calculate the current cell the Turing machine should be visiting at time t'. We can straightforwardly calculate how long it takes to simulate all the register operations, and then each cell takes the same amount of time.

   (a) If we are in the middle of a register register operation: If s is in a register, simulate B on the registers till time t, and then directly output it. If s is not a register, output that cell's value from the previous time, as nothing happened.

   (b) If we are looking at another cell: If s is a cell before the register's location during this operation, then just return the value of this cell at the time step after this. If s is a cell inside the register, then simulate B on this register and this one bit right up till time t, then output s at time t. Otherwise, output the value of this cell now. It hasn't been modified yet by this step in the RAM algorithm.

This can easily be done since the algorithm is input oblivious, we know exactly how many steps in B one step in A will take. There is a direct, simple way to translate from a state in A to a state in B. And each step from one bit in memory to the next as it seeks the appropriate index takes the same amount of time, so we can skip right to the correct one. Thus we can easily compute if the sought index is before, or after the one being checked and change the state appropriately. □

Since RAM algorithms can space efficiently simulate two tape Turing machines, and two tape Turing machines can space efficiently simulate RAM algorithms, we get Lemma 5.2.1.

This gives us a version of the original RAM algorithm with a locally checkable computation history, since the single tape Turing machine is a local model of computation. Remember that we are assuming the states are in some convenient binary encoding.

Lemma A.0.2 (Turing machines have Constant Size Consistency Checks)**.** *For any single tape Turing machine with S bounded cells in memory, for any $i \in [S]$ there is a constant size Boolean function on the states of the $i-1, i$, and $i+1$ cell, and a new proposed value for cell i that outputs whether that would be the new state of cell i after a time step.*

We want to use the multilinear extension of the computation history of the B in Lemma A.0.1 to get an arithmetization of Lemma A.0.2. As part of this, we need another arithmetization.

Lemma A.0.3 (Successor Arithmetization)**.** *For field $\mathbb{F}$, $l \geq 1$, there is a $O(l\,\mathsf{polylog}(|\mathbb{F}|))$ time algorithm computing the multilinear extension of $u+1=v$ for l bit numbers, u and v.*

Proof. For this proof, we will assume that u and v have their high order bits first, so v_1 is the bit with the largest magnitude, and v_l is the bit with the smallest magnitude. For all $l \geq 1$, define $f_l : \mathbb{F}^l \times \mathbb{F}^l \to \mathbb{F}$ inductively by

1. If $l = 1$, $f_l(u, v) = (1 - u_1) \cdot v_1$,
2. If $l > 1$, $f_l(u, v) = (1 - u_l) \cdot v_l \cdot \mathrm{equ}(u_{[l-1]}, v_{[l-1]}) + u_l \cdot (1 - v_l) f_{l-1}(u_{[l-1]}, v_{[l-1]})$.

Then by induction, each f_l is multilinear and consistent with the check that $v = u + 1$.

We can calculate every equ term together with $O(l)$ field operations, by starting with $\mathrm{equ}(u_1, v_1)$, then multiplying it by $\mathrm{equ}(u_2, v_2)$ to get $\mathrm{equ}(u_{[2]}, v_{[2]})$, and so on. Then using each of these, f can be calculated inductively in a straightforward way using only $O(l)$ field operations. □

Now we can construct the inconsistency function actually used in our **PCP**. The idea is to take 2 times, 3 spaces, and a claimed computation history, and output if the cells at these times and spaces violate Lemma A.0.2. For technical reasons, we will further ask for the states of those spaces at that time in the input, and only do the check if these states agree with the computation history. Of course, we will actually get an arithmetization of such a boolean function.

So now we can prove Lemma 5.2.3.

Lemma 5.2.3 (Inconsistency Function)**.** *Let k be a constant, s, t be integers, and $\mathbb{F}$ be a field. Let B be a Turing machine with 2^{2^k} different states per cell running in $S = 2^s$ cells, and time $T = 2^t$. Then there is a function Γ_B taking any function $X : \mathbb{F}^s \times \mathbb{F}^k \times \mathbb{F}^t \to \mathbb{F}$ and returning a function $Y : \mathbb{F}^{3s+2t+4(2^k)} \to \mathbb{F}$ such that:*

1. *If X is Boolean on Boolean inputs, Y is Boolean on Boolean inputs.*
2. *If X is Boolean on Boolean inputs, then Y is 0 on all Boolean inputs if and only if X on Boolean inputs encodes a valid computation history for B. That is, X describes a sequence of T different states of B and they are each derived by a valid transition rule.*
3. *If X is degree d, then Y is degree $O(s+t+d)$. If X is degree d in every variable individually, Y is degree $O(d)$ in every variable individually.*
4. *Given oracle access to X, Y can be computed in time $O((t+s)\,\mathsf{polylog}(|\mathbb{F}|))$ with a constant number of calls to X.*
5. *If $Y = \Gamma_B(X)$ is 0 on all Boolean inputs, then $\Gamma_B(MLB(X))$ is also 0 on all Boolean inputs.*

Proof. From Lemma A.0.2, there is a function, $\phi : \{0,1\}^{4(2^k)} \to \{0,1\}$, which takes $a_0, a_1, a_2, a_1' \in \{0,1\}^{2^k}$ and outputs $\phi(a_0, a_1, a_2, a_1') = 0$ if in the Turing machine with a_0, a_1, a_2 adjacent, in that order, in the Turing machine at one time, replaces a_1 with a_1' in the next time, and outputs 1 otherwise.

Let $\hat{\phi}$ be the multilinear extension of ϕ. Since ϕ has constant size, $\hat{\phi}$ is a constant size arithmetic expression that can be computed in time $O(\mathsf{polylog}(|\mathbb{F}|))$.

Let $\theta : \{0,1\}^{3s} \times \{0,1\}^{2t} \to \{0,1\}$ be the function that takes $s_0, s_1, s_2 \in \{0,1\}^s$ and $t_0, t_1 \in \{0,1\}^t$ and outputs 1 if $s_0 + 1 = s_1$, $s_1 + 1 = s_2$ and $t_0 + 1 = t_1$, and 0 otherwise. By Lemma A.0.3, in time $O((t+s)\,\mathsf{polylog}(|\mathbb{F}|))$ we can compute a function $\tilde{\theta} : \mathbb{F}^{3s} \times \mathbb{F}^{2t} \to \mathbb{F}$ that has constant degree in each variable and is consistent with θ on boolean values.

For $s_0, s_1, s_2 \in \mathbb{F}^s, t_0, t_1 \in \mathbb{F}^t, a_0, a_1, a_2, a_1' \in \mathbb{F}^{2^k}$, define Y by:

$$\begin{aligned} Y(s_0, s_1, s_2, t_0, t_1, a_0, a_1, a_2, a_1') = &\tilde{\theta}(s_0, s_1, s_2, t_0, t_1) \cdot \\ &\hat{\phi}(a_0, a_1, a_2, a_1') \cdot \\ &\prod_{j \in \{0,1,2\}} \prod_{i \in \{0,1\}^k} \mathrm{equ}(X(s_j, i, t_0), (a_j)_i) \cdot \\ &\prod_{i \in \{0,1\}^k} \mathrm{equ}(X(s_1, i, t_1), (a_1')_i). \end{aligned}$$

1. If X is binary on binary inputs, Y is binary on binary inputs since it is just a product of functions that are binary on binary inputs.

2. If X is binary on binary inputs, then for binary inputs, Y is 1 if and only if s_0, s_1, s_2 are adjacent states, a_0, a_1, a_2 are the states of s_0, s_1, s_2 at time t_0, a'_1 is the state of s_1 at time t_1, and the transition from a_1 to a'_1, given neighbors a_0 and a_2 is invalid.

   Thus, if X on binary inputs is a valid computation history, no constraints are ever violated, and Y is 0 on binary inputs. If X is not a valid computation history, it has an improper transition at some point, and at that point, Y would be 1.

3. Y is the product of only constantly many terms (since k is constant), all of which, but potentially X, have degree at most $O(s+t)$, and the X has degree d. Products only add degrees, and we only take constantly many products. So we have degree $O(s+t+d)$.

   For individual variable degree, the input to each of the constantly many X all have degree 1 in distinct variables. So if X has degree d in every variable individually, each call to X is degree at most d in each variable individually. Y is only a product of constantly many calls to X times functions with constant degree. So Y has degree $O(d)$ in each variable individually.

4. Function $\tilde{\theta}$ runs in time $O((s+t)\,\mathsf{polylog}(|\mathbb{F}|))$, and function $\hat{\phi}$ runs in time $O(\mathsf{polylog}(|\mathbb{F}|))$, and each of the constantly many equ only take $O(\mathsf{polylog}(|\mathbb{F}|))$ time with constantly many calls to X. So we only take $O((s+t)\,\mathsf{polylog}(|\mathbb{F}|))$ time overall with constantly many oracle calls to X.

5. Suppose Y is 0 on all boolean inputs. Let $Y' = \Gamma_A(\text{MLB}(X))$.

   In particular, suppose for some boolean values for $s_0, s_1, s_2, t_0, t_1, a_0, a_1, a_2$, and a'_1, function Y is 0. This happens if and only if one of the products making up Y is 0.

   $\tilde{\theta} = 0$ or $\tilde{\phi} = 0$: Neither of these terms involve X, so they are still 0 no matter what we switch X to.

   equ$(X(s_j, i, t_0), (a_j)_i)$ for some i, j: We know $(a_j)_i$ is binary, so by the definition of equ, this expression simplifies to either $X(s_j, i, t_0) = 0$, or $X(s_j, i, t_0) = 1$. In either case, this implies $X(s_j, i, t_0)$ is binary. Thus on this input, $\text{MLB}(X) = X$, and this term is still 0 in $\Gamma_A(\text{MLB}(X))$.

   equ$(X(s_1, i, t_1), (a'_1)_i)$ for some i: Same as above.

□

B Sum Check Proofs

Here we prove that sum check works: Lemma 5.3.1. For reference, here is how we defined our sum check protocol:

Definition 5.3.2 (Sum Check Protocol Definition)**.** *Let $n, d \in \mathbb{N}$, and $\mathbb{F}$ be a field with $|\mathbb{F}| > \max\{d, n\} + 1$. Suppose $f : \mathbb{F}^n \times \mathbb{F} \to \mathbb{F}$. Then the degree d Sum Check Protocol on f is the following randomized algorithm.*

1. *Get $2n$ random field elements, $R = (r_1, \ldots, r_n$ and $r'_1, \ldots, r'_n)$.*

2. *Reject if $f((r_1, \ldots, r_n), 1) \neq 0$.*

3. *For i from 1 to n:*

   (a) *For $j \in [d+1]$, query*
   $$a_i^j = f((r'_1, \ldots, r'_{i-1}, j, r_{i+1}, \ldots r_n), i+1).$$
   Using these, let $g_i : \mathbb{F} \to \mathbb{F}$ be the degree d polynomial so that for all $j \in [d+1]$, $g_i(j) = a_i^j$.

   (b) *If*
   $$f((r'_1, \ldots, r'_{i-1}, r_i, \ldots r_n), i) \neq (1 - r_i)g_i(0) + r_i g_i(1)$$
   reject.

(c) *If*

$$f((r'_1, \dots, r'_i, r_{i+1}, \dots r_n), i+1) \neq g_i(r'_i)$$

reject.

4. *If all checks pass, accept.*

We often refer to the ability of some function $g : \mathbb{F}^n \to \mathbb{F}$ to pass a sum check. The sum check on function f checks whether $g(x) = f(x, n+1)$ on binary inputs is the constant 0 function. It can be useful to refer to the probability of g passing the sum check, assuming the rest of f is defined optimally.

Definition B.0.1 (Passing A Sum Check)**.** *Let $n, d \in \mathbb{N}$, $\mathbb{F}$ be a field with $|\mathbb{F}| > \max\{n, d+1\}$, $\delta \in [0,1]$, and $g : \mathbb{F}^n \to \mathbb{F}$. Then we say g passes the degree d sum check with probability δ if there exists some function $f : \mathbb{F}^n \times \mathbb{F} \to \mathbb{F}$ so that for all $x \in \mathbb{F}$, $g(x) = f(x, n+1)$, and f passes the degree d sum check protocol with probability δ.*

If g is low degree, then the sum check does a good job checking if g is 0 on all binary inputs.

Lemma B.0.2 (Sum Check Of Low Degree Polynomial)**.** *Let $n, d \in \mathbb{N}$, $\mathbb{F}$ be a field with $|\mathbb{F}| > n(d+1)$, $\delta \in [0,1]$, and $g : \mathbb{F}^n \to \mathbb{F}$. Then:*

1. *If for all $x \in \{0,1\}^n$ we have $g(x) = 0$ and the max degree of g in any variable is at most d, then g passes the degree d sum check with probability 1.*
2. *If there exists $x \in \{0,1\}^n$ such that $g(x) \neq 0$ and the max degree of g in any variable is at most d', then g passes the degree d sum check with probability at most $\frac{(d'+1)n}{|\mathbb{F}|}$.*

Proof. First we define f in the format a sum check expects (whether or not the multilinear extension of g actually is 0).

$$\begin{aligned} f_{n+1}((x_1, \dots, x_n), n+1) =& g(x_1, \dots, x_n) \\ f((x_1, \dots, x_n), i) =& (1 - x_i) f((x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n), i+1) \\ & + x_i f((x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n), i+1) \\ f_i(x) =& f(x, i). \end{aligned}$$

For $i \notin [n+1]$, how we define $f(x,i)$ is arbitrary since it is never queried. By induction, see that for $n \geq j \geq i \geq 1$, then f_i is linear in variable j. In particular, f_1 is multilinear. Further, each f_i agree on boolean inputs.

1. Suppose for all $x \in \{0,1\}^n$ we have $g(x) = 0$ and the max degree of g in any variable is at most d. Then by induction, for $i \in [n+1]$ and $j \in [n]$, function f_i has degree at most d in variable j.

   Then choose randomness, $R = (r_1, \dots, r_n$ and $r'_1, \dots, r'_n)$. See that f_1 is multilinear, and 0 on all binary inputs, so it must be the 0 function. Thus $f((r_1, \dots, r_n), 1) = 0$.

   For every $i \in [n]$, by definition

   $$\begin{aligned} & f((r'_1, \dots, r'_{i-1}, r_i, r_{i+1}, \dots, r_n), i) \\ =& (1 - r_i) f((r'_1, \dots, r'_{i-1}, 0, r_{i+1}, \dots, r_n), i+1) \\ & + r_i f((r'_1, \dots, r'_{i-1}, 1, r_{i+1}, \dots, r_n), i+1). \end{aligned}$$

   Since f_{i+1} is degree at most d in variable i, function g_i in the sum check is a degree at most d polynomial, and g_i agrees with f_{i+1} on $d+1$ points, then $f_{i+1} = g_i$ as a function of variable i. So

   $$f((r'_1, \dots, r'_{i-1}, r'_i, r_{i+1}, \dots, r_n), i+1) = g_i(r'_i)$$

   and

   $$f((r'_1, \dots, r'_{i-1}, r_i, r_{i+1}, \dots, r_n), i) = (1 - r_i) g_i(0) + r_i g_i(1).$$

   So all tests pass.

2. Suppose there exists $x \in \{0,1\}^n$ such that $g(x) \neq 0$ and the max degree of g in any variable is at most d'. Then by induction, for $i \in [n+1]$ and $j \in [n]$, function f_i has degree at most d' in variable j. Now take any candidate function, $f' : \mathbb{F}^n \times \mathbb{F} \to \mathbb{F}$, so that $f'(x, n+1) = g(x)$. Define $f'_i(x) = f(x, i)$.

   Our goal is to show that if f'_1 is not equal to f_1, then with low probability will f' be able to change to f on the values we are evaluating without the sum check catching it. Since f' must be f at the last step, due to how we defined them, function f' must fail the sum check with high probability.

   Since $f_1(x) \neq 0$, function f_1 is not the constant 0 function. Since f_1 is multilinear, f_1 has degree at most n. Thus the probability that $f_1(r_1, \ldots, r_n) = 0$ is at most $\frac{n}{|\mathbb{F}|}$ by Schwartz-Zippel.

   Suppose $f_1(r_1, \ldots, r_n) \neq 0$. Then either $f'_1(r_1, \ldots, r_n) = f_1(r_1, \ldots, r_n)$ or not. If they are equal, sum check will fail. Now we will perform induction.

   Suppose for $i \leq n$, with probability at most $\frac{n+d'(i-1)}{|\mathbb{F}|}$ has f' not failed the sum check by step i and

$$f_i(r'_1, \ldots, r'_{i-1}, r_i, r_{i+1}, \ldots, r_n) = f'_i(r'_1, \ldots, r'_{i-1}, r_i, r_{i+1}, \ldots, r_n).$$

   So suppose

$$f_i(r'_1, \ldots, r'_{i-1}, r_i, r_{i+1}, \ldots, r_n) \neq f'_i(r'_1, \ldots, r'_{i-1}, r_i, r_{i+1}, \ldots, r_n).$$

   Define degree d' function g^*_i and degree d function g'_i so that for $j^* \in [d'+1]$ and $j' \in [d+1]$ we have

$$\begin{aligned} g^*_i(j^*) =& f_{i+1}(r'_1, \ldots, r'_{i-1}, j^*, r_{i+1}, \ldots, r_n) \\ g'_i(j') =& f'_{i+1}(r'_1, \ldots, r'_{i-1}, j', r_{i+1}, \ldots, r_n). \end{aligned}$$

   Since f_{i+1} is degree d' in variable i and agrees with g^*_i on $d'+1$ places, for any r'_i we have

$$\begin{aligned} f_{i+1}(r'_1, \ldots, r'_{i-1}, r'_i, r_{i+1}, \ldots, r_n) =& g^*_i(r'_i) \\ f_i(r'_1, \ldots, r'_{i-1}, r_i, r_{i+1}, \ldots, r_n) =& (1-r_1)g^*_i(0) + r_i g^*_i(1). \end{aligned}$$

   If $g^*_i = g'_i$, then the sum check fails because

$$f_i(r'_1, \ldots, r'_{i-1}, r_i, r_{i+1}, \ldots, r_n) \neq f'_i(r'_1, \ldots, r'_{i-1}, r_i, r_{i+1}, \ldots, r_n).$$

   So suppose $g^*_i \neq g'_i$. By Schwartz-Zippel, the probability $g^*_i(r'_i) = g'_i(r'_i)$ is at most $\frac{d'}{|\mathbb{F}|}$.

   So suppose $g^*_i(r'_i) \neq g'_i(r'_i)$. Then if

$$f'_{i+1}(r'_1, \ldots, r'_{i-1}, r'_i, r_{i+1}, \ldots, r_n) \neq g'_i(r'_i)$$

   the sum check fails. So suppose they are equal. Then we have

$$\begin{aligned} f'_{i+1}(r'_1, \ldots, r'_{i-1}, r'_i, r_{i+1}, \ldots, r_n) =& g'_i(r'_i) \\ \neq& g^*_i(r'_i) \\ =& f_{i+1}(r'_1, \ldots, r'_{i-1}, r'_i, r_{i+1}, \ldots, r_n). \end{aligned}$$

   So by a union bound, the probability that we haven't failed by step $i+1$ and

$$f_{i+1}(r'_1, \ldots, r'_{i-1}, r'_i, r_{i+1}, \ldots, r_n) = f'_{i+1}(r'_1, \ldots, r'_{i-1}, r'_i, r_{i+1}, \ldots, r_n)$$

   is at most $\frac{n+d'i}{|\mathbb{F}|}$.

   Finally, for $i = n+1$, we know $f'_{n+1} = f_{n+1}$, since they are equal to function g. So with probability at most $\frac{n(d'+1)}{|\mathbb{F}|}$ has the sum check not failed.

□

Now we need a prover to actually carry out this protocol in small space. But this can be done following the expected definition for f in the obvious way.

Lemma B.0.3 (Sum Check Proofs Require Low Space)**.** *Suppose $g : \mathbb{F}^n \to \mathbb{F}$ has degree d in each variable and can be computed in space S and is 0 on Boolean inputs. Then for some $f : \mathbb{F}^n \times \mathbb{F} \to \mathbb{F}$ so that for all $x \in \mathbb{F}$, $g(x) = f(x, n+1)$ and f passes the degree d sum check protocol with probability 1, f can be calculated in space $O(n\log(|\mathbb{F}|) + S)$.*

Further, if g has degree d in variable i, so does f.

Proof. First, we define f inductively in the usual way:

$$\begin{aligned} f((x_1,\dots,x_n), n+1) =& g(x_1,\dots,x_n) \\ f((x_1,\dots,x_n), i) =& (1-x_i) f((x_1,\dots,x_{i-1},0,x_{i+1},\dots,x_n), i+1) \\ &+ x_i f((x_1,\dots,x_{i-1},1,x_{i+1},\dots,x_n), i+1) \\ f_i(x) = f(x, i).& \end{aligned}$$

Now one may observe that we didn't define f for $i \notin [n+1]$. We can just assume they are all 0 for now, this will be addressed in our **ePCP**. Note this is the same f used in Lemma B.0.2, so passes the sum check.

We can then rewrite each f_i in a more convenient format of

$$f_i(x_1,\dots,x_n) = \sum_{y\in\{0,1\}^{n+1-i}} g(x_1,\dots,x_{i-1},y_1,\dots,y_{n+1-i}) \cdot \prod_{j\in[n+1-i]} \mathrm{equ}(y_j, x_{i-1+j}).$$

This can be shown to be correct by induction. Then this can be calculated for any f_i in a straightforward way keeping track of a constant number of field elements, and a pointer for y and j, requiring only $O(n\log(|\mathbb{F}|))$ bits. ☐

Given all of these, Lemma 5.3.1 follows.

Lemma 5.3.1 (Sum Check Protocol)**.** *Let $n, d \in \mathbb{N}$, and $\mathbb{F}$ be a field with $|\mathbb{F}| > (d+1)n$. Then there is some protocol, A, so that for any $f : \mathbb{F}^n \times \mathbb{F} \to \mathbb{F}$:*

1. *For some $m = O(nd)$ and $R = 2n$, there is a verifier $V : \mathbb{F}^m \to \{0,1\}$ and query function $Q : \mathbb{F}^R \times [m] \to \mathbb{F}^n \times \mathbb{F}$ so that*
$$A(f, r) = V(f(Q(r,1)), f(Q(r,2)), \dots, f(Q(r,m))).$$
2. *V runs in time $O(nd\,\mathsf{polylog}(|\mathbb{F}|))$ and space $O(nd\log(|\mathbb{F}|))$.*
3. *For any $r \in \mathbb{F}^R$, for $Q_r(i) = Q(r,i)$, Q_r is time $O(nd\,\mathsf{polylog}(|\mathbb{F}|))$ extrapolatable.*
4. *For any $r \in \mathbb{F}^R$, the last coordinate of Q is always an element of*[11] *$[n+1]$. That is, for all $i \in [m]$, $Q(r,i)_{n+1} \in [n+1]$*

   Further, the last coordinate of Q is only equal to $n+1$ at most $O(d)$ times.

Completeness *For any $g : \mathbb{F}^n \to \mathbb{F}$ where g has max degree d in any individual variable, if for all $x \in \{0,1\}^n, g(x) = 0$, then there is some $f : \mathbb{F}^n \times \mathbb{F} \to \mathbb{F}$ so that:*

- *For all $x \in \mathbb{F}^n$ we have $g(x) = f(x, n+1)$.*
- *Sum check succeeds on f:*
$$\Pr_r[A(f,r) = 1] = 1.$$
- *Function f has degree at most d in each of its first n variables.*
- *If function g is computable in space S, then function f is computable in space $O(n\log(|\mathbb{F}|) + S)$.*

[11] Any reasonable embedding of $[n+1]$ in $\mathbb{F}$ works, just as in the definition of extrapolatability from Section 4.1.

Soundness *for any* $g : \mathbb{F}^n \to \mathbb{F}$ *where* g *has max individual degree* d'*, if there exists* $x \in \{0,1\}^n$ *such that* $g(x) \neq 0$*, then for any* $f : \mathbb{F}^n \times \mathbb{F} \to \mathbb{F}$ *so that for all* $x \in \mathbb{F}^n, g(x) = f(x, n+1)$*, sum check fails with high probability:*

$$\Pr_r[A(f,r) = 1] \leq \frac{(d'+1)n}{|\mathbb{F}|}.$$

Proof. The verifier and query function are implicit in Definition 5.3.2. As are the verifier runtime, and where the queries are made. Extrapolatability of the queries is shown in Lemma 5.3.3. Completeness and soundness is shown in Lemma B.0.2. The low space in the completeness case is shown in Lemma B.0.3. □