Last time we saw how to solve *Avoid* in single-valued $\mathcal{FS}_2 \subseteq \mathcal{FZPP}^{\mathcal{NP}}$, and deduce circuit lower bounds in $\mathcal{S}_2\mathcal{E} \subseteq \mathcal{ZPE}^{\mathcal{NP}}$. Today we'll prove an incomparable result of a similar flavor:

- The algorithm will run in a smaller complexity class, namely $\mathcal{AM}$ instead of $\mathcal{S}_2 \subseteq \mathcal{ZPP}^{\mathcal{NP}}$. It's instructive to think of this as "half a level down in the polynomial-time hierarchy". However, the runtime will increase to be quasipolynomial, and the algorithm will use some non-uniform advice.
- The algorithm will solve a problem that is more general (when considering explicit construction problems): hitting any dense set in $co\mathcal{AM} \supseteq co\mathcal{NP}$.[1]

There are three reasons for learning this. The first is that it's one of the best currently known lower bounds for general circuits. The second is to re-hash the proof from [CLO+23]. The third is as an excuse to present some wonderful underlying technical tools from hardness-vs-randomness, which were developed in the 2000's.

1 Setup and result statement

The algorithm will solve a total search problem. When solving total search problems by proof systems, there is no apparent distinction between solving a problem and solving the "co-problem". To match the intuition from decision problems, we define the notion of solving such a total search problem using $\mathcal{AM} \cap co\mathcal{AM}$ terminology.

Definition 1. *An $\mathcal{AM} \cap co\mathcal{AM}$ verifier V for a function $f\colon \{0,1\}^* \to \{0,1\}^*$ specifies a constant-round proof system such that for every $x \in \{0,1\}^*$:*

1. *There is an honest prover P for which $\Pr[\langle V, P\rangle(x) = f(x)] = 1$.*
2. *For every prover $\tilde{P}$ we have $\Pr[\langle V, \tilde{P}\rangle(x) \notin \{f(x), \perp\}] \leq 1/2$.*

The definition extends to relations in the natural way.

We can reduce the error by repetition. We can also modify any $\mathcal{AM}$ protocol into a one-round protocol (i.e., the verifier sends coins, the prover responds with a message) while paying only a polynomial runtime overhead [BM88].

It is not obvious how to define $\mathcal{AM}$ verifiers that take non-uniform advice. Clearly, completeness should hold when V is given the correct advice, but should soundness hold only for the correct advice, or for all advice? Both models seem reasonable, and the result in the work we present today holds wrt the stricter notion. For simplicity, we will only present the more relaxed notion.

[1] The two problems are formally incomparable, since *Avoid* is a search problem (i.e,. with arbitrary instances) whereas we now solve an explicit construction problem (i.e., with instances 1^n). If we restrict our attention to uniform sequences of *Avoid* instances, then hitting dense $co\mathcal{NP}$-sets is more general.

Result statement. Chen, Li, and Liang showed that the explicit construction problem of any dense $S \in co\mathcal{AM}$ can be solved in $\mathcal{AM} \cap co\mathcal{AM}$ in quasipolynomial time, and with a small amount of non-uniform advice.

Theorem 2 ([CLL25]). *For every $\epsilon > 0$ and dense $S \in co\mathcal{AM}$ there is an $\mathcal{AM} \cap co\mathcal{AM}$ verifier running in quasipolynomial time and receiving $2^{(\log n)^\epsilon}$ bits of advice that, infinitely often, is single-valued and solves the search problem of S. Moreover, there is such an algorithm that succeeds on infinitely many powers of two.*

Corollary 3. *For every $\epsilon > 0$ there is a problem in $(\mathcal{AMEXP} \cap co\mathcal{AMEXP})/2^{n^\epsilon}$ that is hard for circuits of size $\Theta(2^n/n)$.*

Proof sketch. Define $x \in L \iff A(1^{2^{|x|}})_x = 1$. If n is such that $A(2^n)$ hits S, we give the appropriate advice for A, which is of length 2^{n^ϵ}; otherwise we give advice "$\perp$" (i.e., compute a trivial function). Note that $A(2^n)$ runs in time $2^{\text{poly}(n)}$.[2] ∎

2 The high-level construction

Fix a dense $S \in co\mathcal{AM}$. We will use the same high-level strategy of [CLO+23], which is based on derandomization and an iterative win-win. Recall that, loosely speaking, we partition the integers into intervals, and in each interval $I \supseteq [n_0, 2^{O(n_0)}]$:

- On input length n_0, consider $f_0(1^{n_0}) =$ "the first n_0-bit string in S", and base a pseudorandom generator on the computational history of $f_0(1^{n_0})$. The generator uses $\approx 2^{n_0}$ bits of hardness and outputs $\approx 2^{n_0}$ strings of length $n_1 = n^C$.[3]
- If the generator fails to hit S, then the reconstruction computes $f_0(1^{n_0})$ (i.e., outputs the first n_0-bit string in S) in time $\text{poly}(n_1) = \text{poly}(n_0)$. Otherwise, if the generator hits S on input length $n_1 = n_0^C$, then we can solve the search problem of S on input length n_1 in time $2^{n_0} = 2^{n_1^{1/C}}$. We define

  $$f_1(1^{n_1}) = \text{"the first } n_1\text{-bit string in the output of } G^{f_0(1^{n_0})} \text{ that is in } S\text{"}$$

  and recurse.
- At each iteration i, either we win immediately on input length n_i (if the generator fails to hit S and the reconstruction works), or we get a function computable in time $2^{n_0} = 2^{n_{i+1}^{1/C^{i+1}}}$. When $i = O(\log(n)/\text{loglog}(n))$ we have $n_{i+1} = 2^{n_0} = \text{poly}(n_{i+1})$, so either case gives us a polytime algorithm.

[2] In the paper there is a more complicated proof, which shows that the stricter notion of "$\mathcal{AMEXP} \cap co\mathcal{AMEXP}/2^{n^\epsilon}$" is satisfied: even on incorrect advice, the soundness condition still holds.

[3] Indeed, the actual number of hard bits is more than 2^{n_0}, and so is the number of outputs, but for simplicity let's pretend that both are exactly 2^{n_0}.

Moreover, when we want to find objects whose lengths are powers of two (e.g. hard truth-tables), we can set all input lengths in the proof to be powers of two.

This worked well when we had $S \in \mathcal{P}$, but now $S \in co\mathcal{AM}$, which will be the main challenge. Let us examine the two places above that fall apart:

1. When defining the hard function in each iteration, we want to find the first string in the pseudorandom set that is in S. But to decide which strings in the pseudorandom set are in S, we need to run a $co\mathcal{AM}$ verifier.[4] This is a problem because we don't know how to construct pseudorandom generators (with a polynomial-sized output-set) based on hard functions in $co\mathcal{AM}$.

2. The reconstruction we used before isn't really polytime – it's polytime when given oracle access to the distinguisher, which in this case is a $co\mathcal{AM}$ verifier for S. Using the same reconstruction would yield an algorithm in $\mathcal{BPP}^{\mathcal{AM}}$, which is worse than what we proved last time (i.e., than $\mathcal{S}_2 \subseteq \mathcal{ZPP}^{\mathcal{NP}}$).

At a high-level, we'll handle the first problem using advice – a non-uniform entity will give us the index of the first S-string within the pseudorandom set. Most of our effort will be dedicated to plugging-in correct tools from the hardness-vs-randomness literature to resolve the second problem (i.e., to get a reconstruction in $\mathcal{AM} \cap co\mathcal{AM}$).

2.1 Resolving the first problem

The function f_0 is defined using the brute-force algorithm, as before. The function f_1 now doesn't just get input 1^{n_1}, but instead it gets as input $x \in \{0,1\}^{n_1}$, which it parses as an index $x_1 \in \{0,1\}^{n_0}$ of an element in $G^{f_0(1^{n_0})}(1^{n_1})$. Then $f_1(x)$ is defined as the x_1^{th} element in $G^{f_0(1^{n_0})}(1^{n_1})$.

More generally, f_i will parse its input $x \in \{0,1\}^{n_i}$ as a sequence of indices $x = (x_0, x_1, ..., x_{i-1})$; we recursively define

$$G^{f_0(1^{n_0})}(1^{n_1}), \quad G^{f_1(x_1)}(1^{n_2}), \quad ... \quad G^{f_{i-1}(x_{i-1})}(1^{n_i})$$

where

$$f_j(x_j) = G^{f_{j-1}(x_{j-1})}(1^{n_j})_{x_j} .$$

Note there is an algorithm running in time $\text{poly}(n_i, 2^{n_0})$ that computes $f_i(x)$ recursively, feeding the computational history of $f_j(x_j)$ into G each time to compute the next set. As before, we pretend that the runtime is 2^{n_0} for simplicity.

We will use the same pseudorandom generator as in [CLO+23], relying on the fact that the generator is "targeted", in the sense that the same statement (i.e., that if the generator fails then the reconstruction works) still holds when both parties are given an arbitrary input x. The win-win analysis is based on fixing some "canonical" sequence of x_i's, indicating (say) the first element in the pseudorandom set each time.

[4] This is not a problem in the first iteration, since we have enough time to enumerate over all communication patterns; but it is a problem in all subsequent iterations.

- If for every i the generator hits $S \cap \{0,1\}^{n_i}$ when it is given the i^{th} subsequence, then at the last iteration we have a deterministic polytime algorithm that hits S when it is given this sequence of x_i's as input.
- Otherwise, in some iteration i, when the reconstruction gets the i^{th} subsequence as input it pseudodeterministically outputs some element in $S \cap \{0,1\}^{n_i}$ in time $\text{poly}(n_i)$ (and with oracle access to the distinguisher).

In both situations, we transform the algorithm that receives an input into an algorithm that only receives 1^n, by giving the input as advice. Note that the number of advice/input bits that we actually use is $O(n^{1/C}) \ll n$ (i.e., since the advice/input at length n_i is only used to specify at most $\log(n_0)$ indices in a set of size 2^{n_0}). Hence, this yields a single-valued algorithm in $\mathcal{FBPP}^{\mathcal{AM}}$ with n^{ϵ_C} bits of advice, where the $\mathcal{AM}$ oracle comes from the distinguisher in the "reconstruction" case.

In this setting is the algorithm runs in polynomial time (rather than quasipolynomial), but uses n^ϵ bits of advice rather than $2^{(\log n)^\epsilon}$.

2.2 The actual construction

The more interesting part is getting rid of the $\mathcal{AM}$ oracle. Previously we used the generator-reconstruction pair of [CT21; CLO+23], which has reconstruction in $\mathcal{BPP}^D$ because it relies on the [NW94] generator as a sub-component. We will use a generator that follows [MS23], using ideas from [MV05; SU07] as sub-components, and getting reconstruction in $\mathcal{AM} \cap co\mathcal{AM}$ when D represents a $co\mathcal{AM}$ protocol.

Theorem 4. *Let $f\colon \{0,1\}^* \to \{0,1\}^*$ be computable in time T and let $m \leq T^{.01}$. Then there is an algorithm G_f and an $\mathcal{AM}$ verifier R_f such that for every $x \in \{0,1\}^n$:*

- **Generator.** *$G_f(x)$ runs in time $\text{poly}(T)$ and outputs a list of m-bit strings.*
- **Reconstruction.** *When given input x and a $co\mathcal{AM}$ procedure[5] $D\colon \{0,1\}^m \to \{0,1\}$ of size $\text{poly}(m)$ that accepts at least $1/2$ of its inputs, the verifier R_f runs in time $m^{\text{polyloglog}(T)}$ and satisfies:*
  - **(Completeness.)** *If D rejects all strings in $G_f(x)$, then there is P such that $\Pr[\langle R_f, P\rangle(x) = f(x)] = 1$.*
  - **(Soundness.)** *For any $\tilde{P}$ we have $\Pr[\langle R_f, P\rangle(x) \notin \{f(x), \perp\}] \leq 1/2$.*

As a distinguisher D, we will give the reconstruction the $co\mathcal{AM}$ algorithm that decides S, which is indeed guaranteed to have high acceptance probability. Note that the reconstruction runs in time $m^{\text{polyloglog}(T)}$ instead of $\text{poly}(m)$, which is why the final algorithm runs in quasipolynomial time instead of polynomial time. I know of no reason to suspect we can't improve Theorem 4 to get a $\text{poly}(m)$-time reconstruction.

[5] That is, a circuit describing a verifier V that is guaranteed to have $co\mathcal{AM}$-type behavior on all inputs.

Proof sketch for the main theorem. In the first iteration, the reconstruction runs in time $m^{\text{polyloglog}(T)} = 2^{\text{polylog}(n_0)}$ rather than $\text{poly}(n_0)$. Given that we already pay this, we parameterize the proof differently, with

$$n_1 = 2^{(\log n_0)^C}, \quad ..., \quad n_i = 2^{(\log n_0)^{C^i}}.$$

The advantage in this parameterization is that now the advice (specifying a sequence of indices in a set of size 2^{n_0}) will look smaller to each n_i; that is, the advice on input length n_i will be of length $\tilde{O}(n_0) < 2^{(\log n_i)^{eC}}$.

The generator runs in the same time as before, and the reconstruction is a single-valued $\mathcal{AM} \cap co\mathcal{AM}$ algorithm running in time that is quasipolynomial (in the first iteration; it's less in subsequent iterations). Using the same win-win analysis as before with respect to some "canonical" sequence of x_i's, the theorem follows.

3 The pseudorandom generator: Ideas and proof sketch

In the rest of the lecture we will share interesting technical ideas from the proof of Theorem 4. Hence, our goal now is to design a targeted PRG whose reconstruction algorithm, when working with a $co\mathcal{AM}$ distinguisher D, runs in $\mathcal{AM} \cap co\mathcal{AM}$ (rather than in polytime with oracle access to D as in [CT21; CLO+23]).

3.1 The basic framework

The starting point is a generator that is considerably simpler than the one we used before, but has a disadvantage – the reconstruction is non-deterministic. Since today we will be paying in non-determinism anyway, this disadvantage doesn't bother us.

Verification-based reconstruction using PCPs. Consider a hard function f computable in time T. Given input $x \in \{0,1\}^n$:

- The generator encodes the computational history of $f(x)$ as a PCP witness $\pi \in \{0,1\}^{\tilde{O}(T)}$ using the PCP of [BGH+05],[6] and applies the NW generator to π, considered as a truth-table. It runs in time $\text{poly}(T)$ and outputs m-bit strings.
- The prover will send a purported value v for $f(x)$, and our goal is to run the PCP verifier to quickly check that this is indeed the value. Of course, the verifier needs a PCP proof for that, and such proofs are of length $\tilde{O}(T) \gg m$. But given that the generator failed, we will be able to build a concise proof, as follows.

  Recall that the NW reconstruction guarantees the existence of a $\text{poly}(m)$-sized circuit C such that the truth-table of C^D is π. The prover sends a candidate C to us, and we run the PCP verifier, using C^D to answer proof queries.

[6] We usually think of PCPs as verification procedures for $L \in \mathcal{NTIME}[T]$. In most cases, what is actually going on is that the PCP witnesses encodes the computation of the verifier on a witness, i.e. it is a PCP for the deterministic computation $V(x,w)$ with some accepting witness w.

The procedure above only works when it is allowed oracle queries D, which we want to avoid. To see why, say that we try to run D as a (co-)non-deterministic procedure. Then, for some inputs $q' \in D^{-1}(0)$, a malicious prover can send a proof convincing the verifier that $D(q') = 0$, but can also send another proof that doesn't convince the verifier, in which case the verifier might mistakenly think that $D(q') = 1$.

This can be exploited. A malicious prover will send an adversarial C, the verifier will choose randomness r for the PCP verifier, yielding queries $q = q(r)$ to the PCP proof, and in turn yielding queries q' to D. But now the prover can answer each query q' in two ways (i.e,. either with 0 or to 1) – in other words, the prover can tailor its answer to PCP queries *after the locations of PCP queries have been determined.*

To sum up, when D is evaluated with proofs (rather than accessed as an oracle), instead of committing to a PCP in advance, the prover can first see the queries and then choose its answers.[7] In a PCP system, we want the prover to *commit* to a PCP, and only afterwards the verifier chooses PCP queries.

3.2 Commit-and-evaluate

We will use an additional interactive protocol $(Commit, Eval)$ that has two steps:

- The prover first interactively commits to some C.
- After the commitment, with high probability there is a single F and prover strategy P such that for every z we have $\Pr[\langle Eval, P\rangle (C, z) = F(z)] = 1$, and for every $\tilde{P}, z$ we have $\Pr[\langle Eval, \tilde{P}\rangle (C, z) \notin \{F(z), \perp\}] \leq 1/2$.

Crucially, we will show protocols in which the prover only needs to convince the verifier that some inputs are in $D^{-1}(0)$ (which can be done with a proof, as D is co-non-deterministic). In fact, in all settings below, we will start with an "error-reduced" version of D, which rejects at most 2^{m^ϵ} inputs, and the prover will only need to convince the verifier that certain inputs lie in this small set $D^{-1}(0)$.[8]

Disclaimer. I want to cover a lot of ground now, and there is no time to see full proofs. I'll show key technical ideas, and refer you to the papers for details.

3.2.1 Commit-and-evaluate #1: Hash-based commitments [Sip88]

Observe that there is a concise representation of any $z \in D^{-1}(0)$, namely its m^ϵ-bit index in the set $D^{-1}(0)$. In particular,

Observation 5. *If we randomly choose a hash function $h\colon \{0,1\}^m \to \{0,1\}^{m^{O(\epsilon)}}$ (say, h is pairwise-independent) then whp there are no collisions in $D^{-1}(0)$.*

[7] This is not fully accurate, since the prover only sees the q' queries to D rather than the PCP queries. But the prover still chooses its answers after the PCP queries have been chosen.

[8] A standard way to obtain such D is by randomness-efficient error reduction, using extractors/samplers. This causes a blow-up in the input size and circuit size, which I will ignore for simplicity. It also corresponds to a problem called "quantified derandomization", see [GW13; Tel22].

The generator partitions π into consecutive substrings of size m, which we'll call "chunks", and adds each chunk into the hitting-set. Our focus is the reconstruction, and we may assume that D rejects all chunks, i.e. every chunk is in $D^{-1}(0)$.

To commit, the verifiers sends a hash key, and the prover sends a sequence of hash-values, which are supposed to be the hash values of every m-bit chunk of some string $\tilde{\pi} \in \{0,1\}^{\tilde{O}(T)}$ (ideally $\tilde{\pi} = \pi$, but for now all we care about is that these hash-values will form a commitment to a single global string).[9] To compute $\tilde{\pi}$ at location z with access to D, the verifier non-deterministically de-hashes the relevant chunk; that is, the verifier asks the prover to send the true value $\tilde{\pi}_t$ of the chunk that contains z, and the verifier checks that $\tilde{\pi}_t \in D^{-1}(0)$ and that $h(\tilde{\pi}_t)$ matches the stored hash value.

Recall that no two string in $D^{-1}(0)$ hash to the same value. In particular, after the prover sends initial hash values, for every hash-value there is at most one string in $D^{-1}(0)$ that maps to it. Hence, the commitment binds the prover to a single global $\tilde{\pi}$.

The verifier runs in time $\approx T/m$, so this method is useful when $m = T^{\Omega(1)}$. But even in this case the saving is quite small, and for $m = T^{o(1)}$ the commitment does not even save a polynomial factor, i.e. it's of size $T^{1-o(1)}$.

3.2.2 Commit-and-evaluate #2: Tensor codes [MV05; TZS06]

The next idea will get a commitment of size $T^{O(\epsilon)}$ rather than T/m, and it works even when $m = T^{o(1)}$.

We arithmetize π as a polynomial $\hat{\pi}\colon \mathbb{F}^v \to \mathbb{F}$ of degree $d = \sqrt{m}$, for $v = 3\log(T)/\log(m)$ and field size $q = |\mathbb{F}|$ such that $q \cdot \log(q) = m$.[10] The choice of parameters ensures that $d \ll q$ and $d^v \gg |\pi|$ and the bit-representation of the truth-table of any univariate $\mathbb{F} \to \mathbb{F}$ is of the target output length m.

The generator outputs all q^{v-1} restrictions of $\hat{\pi}$ to axis-parallel lines. Each restriction is the truth-table of a univariate, hence its bit-representation is of length m. The number of output strings is $v \cdot q^{v-1} < m^{O(\log T)/\log m} = \mathrm{poly}(T)$.

In the reconstruction case we may assume that $D^{-1}(0)$ contains all restrictions of $\hat{\pi}$ to axis-parallel lines, which are degree-d univariates. Consider the set $\mathcal{L}$ of truth-tables of degree-d univariates that are contained in $D^{-1}(0)$.

Fact 6. *If we choose a random $S \subseteq \mathbb{F}$ of size $m^{O(\epsilon)}$, and project every truth-table in $\mathcal{L}$ into S (i.e., only look at S-elements), then with high probability there will be no collisions.*

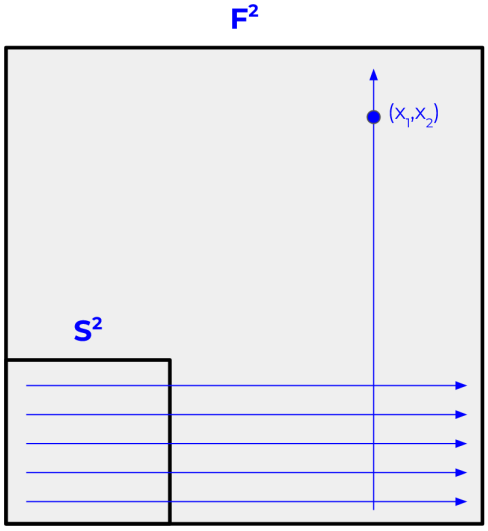
Proof. This follows by a Chernoff bound and a union-bound. We rely on the facts that there are at most 2^{m^ϵ} vectors in $\mathcal{L}$, and that each pair of vectors disagree on most elements (because these are truth-tables of low-degree polynomails). ■

[9] A dishonest prover may also send a hash-value that does not have a pre-image in $D^{-1}(0)$, in which case the relevant part in $\tilde{\pi}$ that this prover committed to can be thought of as "$\perp$".

[10] This can be done via the standard low-degree extension as long as $(d+1)^v > |\pi|$, which is satisfied by our choice of parameters (recall that $|\pi| = \tilde{O}(T)$).

The actual verifier. The verifier chooses at random $S \subseteq \mathbb{F}$ of size $m^{O(\epsilon)}$, and asks the prover to send values for every point in S^v (ideally these will be the evaluations of $\hat{\pi}$, but again, all we care about is that the prover commits to some "global" function $\mathbb{F}^v \to \mathbb{F}$). Note that the prover only sent $|S|^v = m^{O(\epsilon)\cdot v} = T^{O(\epsilon)}$ elements.

Given point $(x_1, ..., x_v) \in \mathbb{F}^v$, let's see how to interactively decommit. For illlustrative purposes, let's first assume that $v = 2$. We are given $(x_1, x_2) \in \mathbb{F}^2$.



- For each $s_2 \in S$, we ask the prover for the restriction of $\hat{\pi}$ to the horizontal-axis-parallel line at "height" s_2, i.e. $(\hat{\pi}(u, s_2))_{u\in\mathbb{F}}$. For each $s_2 \in S$ the prover sends m bits, and we check that the vector is in $\mathcal{L}$ (i.e., by checking that it's the truth-table of a degree-d polynomial and by requesting a proof convincing us that it is in $D^{-1}(0)$), and that it agrees with the stored values at points $\{(s_1, s_2)\}_{s_1\in S}$.

  Overall, the prover sent $|S| \cdot m = m^{1+O(\epsilon)}$ bits, and this allowed us to deduce values at points $\{(x_1, s)\}_{s\in S}$, ideally the evaluations of $\hat{\pi}$.

- Now we ask the prover to send the restriction of $\hat{\pi}$ to the vertical-axis-parallel line at "horizontal shift" x_1, i.e. $(\hat{\pi}(x_1, u))_{u\in\mathbb{F}}$. We check that the vector sent is in $\mathcal{L}$ and agrees with the values we have for $\{(x_1, s_2)\}_{s_2\in S}$ from the previous step. If so, we take the value at (x_1, x_2) from the sent vector.

Let us see that the values at S^2 form a commitment. By Fact 6, we know (whp) that for every fixed s_2, there is only one vector in $\mathcal{L}$ that agrees with the stored values on the line $\{(s_1, s_2)\}_{s_1\in S}$, and the same argument holds for any x_1 and $\{(x_1, s_2)\}_{s_2\in S}$. Since this survives a union-bound over $|\mathbb{F}|^2$ points, by sending values for S^2, for every (x_1, x_2) the prover committed to at most a single decompression for each of the lines "on the journey to (x_1, x_2)", and hence committed to a value at (x_1, x_2).

Higher dimensions. Generalizing to $v > 2$, the prover sends values on S^v. The first step when decommitting is to learn values[11] at the points $\{(x_1, s_2, ..., s_v)\}_{s_2,...,s_v \in S}$.[12] To do so, for each $s_2, ..., s_v \in S^{v-1}$ we non-deterministically decompress the stored $|S|$ points $\{(s_1, s_2, ..., s_v)\}_{s_1 \in S}$ to a degree-d univariate in $\mathcal{L}$, and we store the evaluation of this univariate at x_1. This gives us values at $\{(x_1, s_2, ..., s_v)\}_{s_2,...,s_v \in S^{v-1}}$. Next, we want to learn values at the points $\{(x_1, x_2, s_3, ..., s_v)\}_{s_3,...,s_v \in S}$, and so on.

The same argument as before shows that values at S^v form a commitment to values at $\mathbb{F}^v$. The first step runs in time $|S|^{v-1} \cdot \text{poly}(m) = \text{poly}(T^\epsilon, m)$ and it essentially dominates the runtime complexity, which is at most $v \cdot \text{poly}(T^\epsilon, m) = \text{poly}(T^\epsilon, m)$.

3.3 Commit-and-evaluate #3: A recursive version [SU07]

The last known version allows getting runtime $m^{\text{polyloglog}(T)}$, which comes close to the ideal $\text{poly}(m)$ (recall that even specifying one query to D requires writing m bits).

I will only sketch very high-level ideas. In the protocol above, the runtime is basically $|S|^v$. Can we make this small?

- The size of S comes from the size of $D^{-1}(0)$ (i.e., $|S| \approx \log(|D^{-1}(0)|)$). It's not clear how to reduce $D^{-1}(0)$ beneath $2^{-T^{\Omega(1)}}$ without a super-polynomial blow-up (in the runtime of D and in its input length), so we won't try to do it.
- Instead, it would be great then to have $v = O(1)$. Recall that v is the number of variables in the arithmetization, and that naively arithmetizing with $v = O(1)$ causes a blow-up in degree; that is, the degree becomes $d = T^\epsilon$. Unfortunately, the reconstruction will then pay a runtime factor of $d = T^\epsilon$ at every step.

The key idea in [SU07] is to identify $\mathbb{F}^v \equiv (\mathbb{F}^{v_1})^r$ where $r = O(1)$ and v_1 is smaller than $v = v_1$ by a constant multiplicative factor, and then use recursion (!).

That is, they first apply the generator-reconstruction scheme with the field $\mathbb{F}' = \mathbb{F}^{v_1}$; of course, the field $\mathbb{F}'$ is now so big (i.e., of size $T^{\Omega(1)}$) that the reconstruction can't afford to store the truth-table of a univariate. So instead of working with $\mathbb{F}' = \mathbb{F}^{v_1}$ directly, they apply the generator-reconstruction scheme also to $\mathbb{F}^{v_1} = (\mathbb{F}^{v_2})^r$, and the reconstruction gives a way to commit-and-evaluate a univariate over $\mathbb{F}^{v_1}$. Applying this recursively until the field is small enough, the final runtime is $m^{\text{polyloglog}T}$. It's pretty amazing that this works, and the construction requires additional clever ideas.[13]

[11] Recall that these values are ideally of $\hat{\pi}$, but if not, then at least of some polynomial that the prover committed to.

[12] In three dimensions, imagine shifting the y-z plane to be at the correct x-axis alignment.

[13] For example, instead of the generator outputting the evaluations of $\hat{\pi}$ on an entire line and the reconstruction reading the entire line, the generator outputs evaluations on various subsets of the line (chosen using a sampler/extractor) and the reconstruction probabilistically recovers $\hat{\pi}$ on the line.

References

[BGH+05] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. "Short PCPs Verifiable in Polylogarithmic Time". In: *Proc. 20th Annual IEEE Conference on Computational Complexity (CCC)*. 2005, pp. 120–134.

[BM88] László Babai and Shlomo Moran. "Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes". In: *Journal of Computer and System Sciences* 36.2 (1988), pp. 254–276.

[CLL25] Lijie Chen, Jiatu Li, and Jingxun Liang. "Maximum circuit lower bounds for exponential-time Arthur Merlin". In: *Proc. 57th Annual ACM Symposium on Theory of Computing (STOC)*. [2025] ©2025, pp. 1348–1358.

[CLO+23] Lijie Chen, Zhenjian Lu, Igor Carboni Oliveira, Hanlin Ren, and Rahul Santhanam. "Polynomial-Time Pseudodeterministic Construction of Primes". In: *arXiv preprint arXiv:2305.15140* (2023).

[CT21] Lijie Chen and Roei Tell. "Hardness vs Randomness, Revised: Uniform, Non-Black-Box, and Instance-Wise". In: *Proc. 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 2021, pp. 125–136.

[GW13] Oded Goldreich and Avi Widgerson. "On derandomizing algorithms that err extremely rarely". In: *Electronic Colloquium on Computational Complexity: ECCC* 20 (2013), p. 152.

[MS23] Dieter van Melkebeek and Nicollas Sdroievski. "Instance-Wise Hardness versus Randomness Tradeoffs for Arthur-Merlin Protocols". In: *Proc. 38 Annual IEEE Conference on Computational Complexity (CCC)*. 2023.

[MV05] Peter Bro Miltersen and N. V. Vinodchandran. "Derandomizing Arthur-Merlin games using hitting sets". In: *Computational Complexity* 14.3 (2005), pp. 256–279.

[NW94] Noam Nisan and Avi Wigderson. "Hardness vs. randomness". In: *Journal of Computer and System Sciences* 49.2 (1994), pp. 149–167.

[Sip88] Michael Sipser. "Expanders, randomness, or time versus space". In: *Journal of Computer and System Sciences* 36.3 (1988), pp. 379–383.

[SU07] Ronen Shaltiel and Christopher Umans. "Low-end uniform hardness vs. randomness tradeoffs for AM". In: *Proc. 39th Annual ACM Symposium on Theory of Computing (STOC)*. 2007, pp. 430–439.

[Tel22] Roei Tell. "Quantified derandomization: how to find water in the ocean". In: *Foundations and Trends® in Theoretical Computer Science* 15.1 (2022), Paper No 1, 125.

[TZS06] Amnon Ta-Shma, David Zuckerman, and Shmuel Safra. "Extractors from Reed-Muller codes". In: *Journal of Computer and System Sciences* 72.5 (2006), pp. 786–812.