Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 91 (2025)

Tree PCPs

Tamer Mour* Alon Rosen† Ron D. Rothblum‡

November 10, 2025

Abstract

Probabilistically checkable proofs (PCPs) allow encoding a computation so that it can be quickly verified by only reading a few symbols. Inspired by *tree codes* (Schulman, STOC'93), we propose *tree PCPs*; these are PCPs that evolve as the computation progresses so that a proof for time t is obtained by appending a short string to the end of an accepting proof for time $t-1$. At any given time, the tree PCP can be locally queried to verify the *entire* computation so far.

We construct tree PCPs for non-deterministic space-s computation, where at time step t, the proof only grows by an additional $\mathrm{poly}(s) \cdot t^{\varepsilon}$ bits, and the number of queries made by the verifier to the overall proof is $\mathrm{poly}(s) \cdot t^{\varepsilon}$, for an arbitrary constant $\varepsilon > 0$.

Tree PCPs are well-suited to proving correctness of ongoing computation that unfolds over time, in particular in a distributed setting where the computation is carried by mutually untrusting generations. They may be thought of as an information-theoretic analog of the cryptographic notion of *incrementally verifiable computation* (Valiant, TCC'08).

To obtain tree PCPs, we present the first results establishing strong local testability and local correctability for tree codes, and construct a tree code that achieves both properties simultaneously.

"Forty-two!" yelled Loonquawl. "Is that all you've got to show for seven and a half million years' work?" "I checked it very thoroughly," said the computer, "and that quite definitely is the answer."

Douglas Adams, The Hitchhiker's Guide to the Galaxy

*Bocconi University, BIDSA. E-mail: tamer.mour@unibocconi.it. Work supported by European Research Council (ERC) under the EU's Horizon 2020 research and innovation programme (Grant agreement No. 101019547), and Stellar Foundation Grant.

†Bocconi University, BIDSA. E-mail: alon.rosen@unibocconi.it. Work supported by European Research Council (ERC) under the EU's Horizon 2020 research and innovation programme (Grant agreement No. 101019547), Cariplo CRYPTONOMEX grant, and Stellar Foundation Grant.

‡Succinct. E-mail: rothblum@gmail.com.

ISSN 1433-8092

Contents

1 Introduction

Consider an enormous computational task carried across multiple generations. For instance, the mathematical corpus produced by humanity or, say, the mere ledger of some blockchain. Is it possible to quickly verify that the current generation's computation state is correct?

Using a *probabilistically checkable proof* (PCP), it is possible to encode computations so that at any later point one can quickly verify their correctness [BFLS91, FGL+96, AS98, ALM+92]. However, in the context of our trans-generational computation, we additionally wish to refrain from regenerating the entire encoding each time a new computation step is performed.

Traditional PCPs lack such an incremental encoding property. They are in fact non-incremental by design: their representation of computation guarantees large Hamming distance between the encoding of any two distinct paths of computation. Thus, any additional computational step requires regenerating a large portion of the encoding.

We introduce *tree PCPs*: proofs that can be *incrementally* generated and *quickly verified*. Whatever part of the proof has already been generated becomes canon and any newly generated proof is appended to the existing one. The proof can be efficiently verified at any point in time by reading a small number of symbols and running a prescribed verification algorithm on them.

Importantly, any accepting proof can be efficiently extended to a proof for the next step by reading few symbols, even if it is not the correct proof defined by the tree PCP construction. This guarantees robustness in the presence of malicious generations of provers, who may attempt to sabotage the work of future generations by deviating from the prescribed generation algorithm.

Tree PCPs are inspired by the notion of *tree codes* [Sch93]. Originally motivated by interactive communication, tree codes allow for *online encoding* and guarantee *tree distance*: the i-th symbol of the codeword can depend only on the first i symbols of the message, and any two codewords differ in many coordinates but only starting from the point at which they diverge.

A PCP is to an error-correcting code what a tree PCP is to a tree code. The former inherits Hamming distance from the underlying code, whereas the latter tree distance. Both require their respective error-correcting code to be *locally testable*. Local testability means that proximity of a given word to the tree code can be tested by reading very few locations in the alleged codeword, so that highly corrupt codewords are rejected with high probability.

In [MRR25] it was shown how to construct locally testable tree codes. This is a crucial first step towards attaining tree PCPs, but far from being sufficient. While the blueprint for tree PCPs and some of the machinery that comes into realizing it are inherited from traditional PCPs, the setting in which tree PCPs are constructed requires taking incrementality into consideration.

Closest to tree PCPs is Valiant's notion of *incrementally verifiable computation* (IVC) [Val08]. An IVC is a computationally-sound proof system in which one can incrementally prove that a long computation was done correctly. Soundness of tree PCPs is unconditional, but it is assumed that the PCP string is stored in its entirety and can be (locally) read by the verifier. We further contrast tree PCPs with IVC and other notions of incremental proofs in Section 1.4.

1.1 Tree PCPs

Similarly to [Val08], our focus is on verifiability of *non-deterministic, space-bounded computation*, relative to a circuit $C : \{0,1\}^{3s} \to \{0,1\}$ that verifies a non-deterministic *step function*.

The computation starts with an initial (canonical) configuration $a_0 \in \{0,1\}^s$. At time-step t, the computation can progress from configuration $a_{t-1} \in \{0,1\}^s$ to configuration $a_t \in \{0,1\}^s$ if and only if there exists a witness $w_t \in \{0,1\}^s$ that satisfies $C(a_{t-1}, a_t, w_t) = 1$.

Definition 1.1. *A sequence $p = (a_0, a_1, \ldots, a_n)$, is said to be a* path *under C if and only if there exists a sequence of witnesses $(w_1, \ldots, w_n)$ so that $C(a_{t-1}, a_t, w_t) = 1$ for all $t \in [n]$.*

We denote the set of all paths under C by $\textsc{Paths}(C)$. The size s of configurations a_t and of witnesses w_t, as well as the size $|C|$ of the circuit C, are all constant in the length n of the computation.

The goal is to verify that the computation induced by C reaches a configuration a in n steps. Define the language *circuit reachability* as:

$$\textsc{CktReach} = \{(C, a, n) \mid \exists (a_0, \ldots, a_n) \in \textsc{Paths}(C) : \ a_0 = 0, a_n = a\}\,. \tag{1}$$

A *witness* for membership of (C, a, n) in CktReach consists of a path $p = (a_0, \ldots, a_n)$ and a witness $w = (w_1, \ldots, w_n)$ for membership of p in $\textsc{Paths}(C)$.

Remark 1.2. *Unlike IVCs that require succinctness, the restriction to space-bounded computations is not inherent to tree PCPs but is rather a limitation of our construction. While we choose to restrict the definition of tree PCPs to this special case for simplicity, we note that one can think of natural generalizations to broader classes of incremental computations, as we discuss in Section 1.5.*

Given a witness (p, w), the language CktReach can be verified in time $O(|C| \cdot n)$ by simply verifying that $C(a_{t-1}, a_t, w_t) = 1$ for all $t \in [n]$. In a model where a verifier is allowed to query a proof π, PCPs enable probabilistic verification of CktReach in $o(|C| \cdot n)$ time.

Definition 1.3 (PCP Verifier)**.** *A* PCP verifier *V for a language $L \subset \{0,1\}^*$ with soundness error $\epsilon \in (0, 1)$ is a probabilistic algorithm that, on input a statement $x \in \{0,1\}^n$, makes oracle queries to a* proof *π and outputs 0 (rejects) or 1 (accepts), such that the following properties hold:*

- Completeness*: For any $x \in L$, there exists a proof oracle π such that $\Pr[V^\pi(x) = 1] = 1$.*
- Soundness*: For any $x \notin L$ of length n, and any proof oracle π, $\Pr[V^\pi(x) = 1] < \epsilon$.*

Tree PCPs are PCPs for the language CktReach in the setting of incremental computation. Besides standard completeness and soundness, we require that an accepting PCP can be extended incrementally with the computation: Tree PCPs admit an *incremental prover* which can extend any accepting proof π corresponding to computation of length n to an accepting proof for computation of length $n + 1$, by reading only few locations in π and appending a short string to it.

Definition 1.4 (Tree PCP)**.** *An (ℓ, q, ϵ)-*tree PCP *is a pair of algorithms (P, V), where:*

- *V is a* PCP verifier *for* CktReach *which on input (C, a, n) makes at most $q(n, |C|)$ queries and has soundness error $\epsilon(n, |C|)$.*
- *P is an* incremental prover *which on input (C, a, a', w, n) makes at most $q(n, |C|)$ queries to an oracle π and outputs $\pi' \in \{0,1\}^{\ell(n,|C|)}$ that satisfies: If $\Pr[V^\pi(C, a, n) = 1] := \epsilon' > \epsilon(n, |C|)$ and $C(a, a', w) = 1$, then $\Pr[V^{(\pi,\pi')}(C, a', n + 1) = 1] \geq \epsilon'$.*

The above definition implies perfect completeness of P by induction, as it can be used to incrementally construct a PCP that convinces the verifier with probability 1. Our definition is however stronger than merely requiring an accepting proof to be constructed incrementally; this alone is not satisfactory in a setting where different generations of provers are involved in constructing the PCP and where a corrupt prover may deviate from the prescribed proof, creating a PCP that accepts but that cannot be extended by future generations.

Tree PCPs are non-trivial when both the (verifier's) query complexity q and the length of a new step in the proof ℓ are sublinear in the size of the computation $n \cdot |C|$. If q is linear, a verifier could simply read the entire path of computation. If ℓ is (quasi-)linear, a tree PCP can be created by concatenating (standard) PCPs, each proving the computation from scratch.

Ideally, we would like q to be much smaller than $n \cdot |C|$ and ℓ to be independent of n, leading to a total proof length almost linear and approaching the length of state-of-the-art standard PCPs. We are additionally interested in minimizing the prover's runtime complexity, preferably making it proportional to q. Just like in standard PCPs, the soundness error ϵ can be always reduced by repetition as long as it is bounded away from 1 (say 2/3).

1.2 An Amortized Tree PCP

It was pointed to us by Rafail Ostrovsky and Daniel Wichs [OW25] that tree PCPs with an *amortized* incremental prover can be generically built from standard PCPs.

The tree PCP for a computation of length n is composed by a hierarchy of $\lceil \log n \rceil$ levels of standard PCPs, proving statements of geometrically growing size: Level $i = 0, \ldots, \lceil \log n \rceil - 1$ contains $\lfloor n/2^i \rfloor$ PCPs for segments of the computations consisting of consecutive 2^i steps. A new PCP is added to the proof when the corresponding segment is completed. That is, a PCP is added to level i every 2^i timesteps. To verify, the verifier queries at most one PCP at every level and locally checks that the $O(\log n)$ statements it verifies via the PCPs are consistent at their endpoints.

Plugging in state-of-the-art PCPs [BS08, Din07], the above construction achieves query complexity polylogarithmic in n for the verifier, where the next piece in the proof at time n can be computed by the prover in amortized time and query complexity of $\mathrm{polylog}(n \cdot s)$.

To the best of our knowledge, any attempt to de-amortize this construction falls short of our definition of tree PCPs and, in particular, fails to guarantee completeness in the presence of corrupt generations of provers. For instance, straight-forward de-amortization where the construction of any PCP at level i is performed over $O(2^i)$ generations breaks if one of the provers corrupts the de-amortized computation at one of its steps. To guarantee completeness in such an outline, one must devise machinery to continuously verify the incremental construction of the PCP, taking us back to the original problem of verifying incremental computation, namely building a tree PCP.

Not surprisingly, an analogous hierarchical structure to the construction above underlies the generic construction of tree codes from error-correcting block codes by Schulman [Sch94] (see Fig. 2). Similar design ideas further appear in the literature of proof systems in the incremental setting, e.g. [CKO14, BBBF18, DGMV20, EFKP20].

1.3 Our Results

We construct a tree PCP where both the verifier's and prover's complexity (query and runtime) at time step n, as well as the length of the new string added to the proof, are all proportional to n^γ, for an arbitrarily small constant $\gamma > 0$.

Theorem 1.5 (Tree PCP)**.** *For any $\gamma > 0$, there exists an (ℓ, q, ϵ)-tree PCP with $\ell = q = n^\gamma \cdot \mathrm{poly}(|C|)$ and $\epsilon = n^{-\omega(1)}$.*

Along the way, we develop new instruments for tree codes that are central in our tree PCP construction, and which we believe are of independent interest similarly to their standard analogs for block codes.

First,we demonstrate that tree codes can be locally testable in a strong sense. Namely, that there exists a local test that rejects any non-codeword with probability proportional to its distance

from the code, no matter who small it is. We show this for the generic tree code construction from [MRR25], where strong local testability was posed as an open problem.

Theorem 1.6 (Strong Locally Testable Tree Codes)**.** *There exists a strong locally testable tree code* TC*, where the local test reads $q = n^{\gamma}$ symbols from a word w of length n, for arbitrarily small $\gamma > 0$, and rejects with probability $\frac{1}{\text{polylog}(n)} \cdot \Delta_{\mathsf{T}}(w, \text{TC})$.*

In the above, Δ_{T} denotes tree distance. The theorem is derived by Propositions 4.9 and 4.10.

Second, we show that tree codes can be locally correctable. Local correctability means that it is possible to recover a symbol in a codeword, even when given a corrupted version of it, be reading a small number of symbols. The following is a corollary of Lemmas 3.8 and 4.20.

Theorem 1.7 (Locally Correctable Tree Codes)**.** *There exists a locally correctable tree code* TC*, where the local corrector reads $q = n^{\gamma}$ symbols from a word w of length n satisfying $\Delta_{\mathsf{T}}(w, \text{TC}) < 1/\text{polylog}(n)$, and on input $t \in [n]$ outputs the t^{th} symbol in the codeword closest to w.*

We further show the existence of a tree code that simultaneously achieves both strong local testability and local correctability (a combination of Corollary 4.21, Lemma 4.11, and Proposition 4.10). This is particularly useful, as it allows testing that a given word is close enough to the code and, if so, to correct it to the closest codeword.

We additionally show how to obtain a relaxed notion of local correctability [GRR20] for tree codes (that are also strong locally testable), where the corrector is allowed to fail on a non-codeword, via a simpler and more generic construction.

1.4 Incremental Proofs

Tree PCPs are distinct from *Incremental PCPs* by Naor, Paneth and Rothblum [NPR19]. In an incremental PCP, the proof's symbols change as the computation evolves but they do so separately. That is, each symbol "updates itself" independently of other symbols. In contrast, in a tree PCP symbols never change and the proof is updated by appending new symbols to it.

Closer to tree PCPs is Valiant's notion of *incrementally verifiable computation* (IVC) [Val08]. An IVC is a cryptographic proof system in which one can succinctly prove that a long computation was done correctly. The requirement is that the proof can be efficiently updated as the computation proceeds, in time independent in the length of the computation thus far.

Such stringent efficiency requirements put a hard limit on the proof length and hence also on its soundness guarantee, requiring it to be merely "computational": no polynomial-size attacker can find an accepting proof of a false statement. In contrast, tree PCPs are unconditionally sound.

Valiant demonstrated how IVC could be realized via a technique called *proof merging*. Later, Bitansky *et al.* [BCCT13] relied on recursive composition of proofs. Soundness was based on strong assumptions on the proofs, and postulated access to an idealized random oracle. Since the random oracle ultimately has to be instantiated by a function with short description, these approaches run into circular reasoning. Later works suggest that the standard random oracle model is not sufficient for incremental proofs. They rule out not just explicitly recursive designs, but any IVC that either satisfies some natural constraints (e.g. zero-knowledge) [CL20, HN23] or can prove computations that themselves have access to the random oracle (so-called relativized IVC) [BCG24]. More recent works [DGKV22, PP22] construct IVC from falsifiable assumptions for deterministic computations, but requires a heavy use of expensive "public-key" operations.

Tree PCPs can be compiled à la Kilian/Micali [Kil92, Mic95] into succinct, computationally sound proofs in the random oracle model. The resulting proof-system is a form of IVC in which

the prover needs to maintain a large state consisting of the evolving tree PCP and its Merkle tree,[1] but otherwise adheres to the standard IVC model [BBBF18, DGMV20, EFKP20].

Incremental verifiability also makes an appearance in [CHK+19], where it is shown how to construct a procedure that, given a SAT instance over n variables, counts the number of satisfying assignments. This is accomplished via an exponential sequence of small steps, each computable in time $\text{poly}(n)$. Incremental verifiability in this context means that each intermediate state includes a sumcheck-based proof of its correctness, and the proof can be updated and verified in time $\text{poly}(n)$.

1.5 Further Research

The tree PCPs we build for CktReach capture non-deterministic space-bounded computations. Their proof length is almost optimal and their sublinear verification complexity grows with n^{γ}. This is inferior to state-of-the-art standard PCPs [BS08, Din07], where verification complexity is polylogarithmic in n, for a comparable soundness error.

A first question is whether the incrementality of tree PCPs can be attained with a smaller cost in complexity. Efficient constructions of standard PCPs typically require PCP composition techniques, which are not directly applicable to the incremental setting, or locally testable tree codes with a more efficient local test. Such codes often take the form of m-fold tensor products, for a super-constant m. In the incremental setting of tree codes, $m = \omega(1)$ would require to somehow increase the dimensionality of the tensor tree code over time, which seems to necessitate new techniques beyond those used in [MRR25].

Another path for improved efficiency is via "purely algebraic", more "PCP-friendly", tree codes (see Section 2.3) that could facilitate more straight-forward tree PCPs. While constructing such codes seems to be challenging (see, e.g., [Pud13]), we hope that their application to tree PCPs will motivate further research in this direction.

Perhaps the most intriguing question is whether tree PCPs can realize efficient verification of broader classes of computations beyond space-bounded. One can think of a model where the circuit C is replaced by a function that has local random-access to the path of configurations $(a_0, \ldots, a_n)$ and its witness $(w_1, \ldots, w_n)$. Less generally, one can consider special cases where only specific yet meaningful local access patterns are allowed.

In the notion of tree PCPs we consider, the computation at time t is allowed access only to "memory locations" t and $t-1$ (hence space-bounded). Our construction realizes this access pattern by embedding it on a more expressive access graph. Thus, it already allows access patterns beyond what is captured by the construction (these are defined by the shifts Γ_i – see Fig. 3 and Section 5.2). Extending the functionality further requires new techniques for enforcing a more general structure of consistency constraints across witnesses in the different time-steps.

2 Technical Overview

Our goal is to build a tree PCP for $(C, a, n) \in$ CktReach (see Definitions 1.1 and 1.4 and Eq. (1)). Since a PCP verifier reads only few locations in the proof, a PCP at the very least necessitates a redundant encoding of the witness, where any local change in the witness results in global change in the proof. Whereas traditional PCPs rely on standard error-correcting codes, tree PCPs rely on tree codes, which are well-suited to our incremental setting.

[1] Merkle trees can be made "incremental" using standard techniques.

2.1 The BFLS Blueprint

We follow the "BFLS blueprint" for constructing PCPs [BFL90, BFLS91] (see [Sud04]):

(i) The statement is reduced to the satisfiability of many *local constraints*, akin to the way in which an NP statement reduces to a conjunction of 3-CNF clauses. A witness to the statement is an assignment A that satisfies all local constraints.

(ii) The proof consists of a *redundant encoding* $\widetilde{A}$ of A, using an error-correcting code. This amplifies any divergence from a satisfying assignment, facilitating efficient verification: if an assignment breaks even one of the local constraints, its encoding breaks many of them.

(iii) The code used to encode the witness possesses structure (typically algebraic) that allows evaluating the local constraints "underneath" codewords. By making only few queries to $\widetilde{A}$, it is possible to compute any symbol in the codeword $\widetilde{E}$ that encodes E – the evaluations of the local constraints over A: $E(i) = 0$ if and only if the i^{th} constraint is satisfied by A.

(iv) The *PCP verifier* performs the following two checks:

1. Test that $\widetilde{A}$ is close enough to a codeword. For this, the code is required to be *locally testable*, where proximity of a word to the code can be tested by reading few locations.
2. Test that $\widetilde{E}$ encodes zero evaluations. This is performed by a *zero test* PCP, which is usually based on the *sumcheck protocol* [LFKN92]. The soundness of the zero test is guaranteed whenever $\widetilde{E}$ is a codeword, and relies on the minimum distance of the code.[2]

Notice the gap between the soundness guarantee of the local test that $\widetilde{A}$ is close to a codeword, and the requirement for soundness of the zero test that presumes $\widetilde{E}$, which emerges from $\widetilde{A}$ via (iii), is an *exact* codeword. To bridge this gap, the code is often required to satisfy some notion of *local correctability*, namely that the "correct" value at any location in a slightly-corrupted codeword can be recovered by reading additional few random locations.

2.2 The Tree PCP Outline

To adapt the above outline to tree PCPs, the following ingredients are required:

- An incremental representation of a CktReach statement by a conjunction of *local constraints*. Incrementality here means that extending the statement by a new computation step entails adding few new constraints to the current set of constraints.
- A tree code that is *locally testable* and *locally correctable*, and of structure that allows evaluating the local constraints underneath codewords (à la (iii)).
- A *sumcheck protocol* for the locally testable tree code, which can be converted into a "zero test tree PCP" where a proof for a codeword can be extended to obtain a proof for any extension of the codeword, akin to the incrementality of tree PCPs.

Our starting point is a description of $(C, a, n) \in$ CktReach as a conjunction of local constraints over n input triplets $A_t = (a_{t-1}, a_t, w_t) \in \{0,1\}^{3s}$ to C such that: (1) A_1 and A_n contain valid initial configuration $a_0 = 0^s$ and, respectively, final configuration $a_n = a$, (2) A_t is a valid transition under

The Tree PCP Outline

The Proof:

1. Let $A_t = (a_{t-1}, a_t, w_t) \in \{0,1\}^{3s}$ and denote by $A^i : [n] \to \mathbb{F}$ the column $A^i(t) = A_t(i)$.
2. The proof contains
$$\widetilde{A}^1 \;=\; \mathrm{TC}(A^1),\; \ldots,\; \widetilde{A}^{3s} = \mathrm{TC}(A^{3s}).$$

The Verifier:

1. Using *local testability*, verify that for all $i = 1, \ldots, 3s$, $\widetilde{A}^i$ is close enough to the code TC.
2. Using a *zero test*, verify that the unique closest codewords encode $A^1, \ldots, A^{3s}$ that satisfy:

   2.1 (*Endpoints*) $(A^1(1), \ldots, A^s(1)) = 0^s$ and $(A^{s+1}(n), \ldots, A^{2s}(n)) = a$.

   2.2 (*Transition*) For all $t = 1, \ldots, n$, $C(A^1(t), \ldots, A^{3s}(t)) = 1$.

   2.3 (*Consistency*) For all $i = 1, \ldots, s$ and $t = 2, \ldots, n$, $A^i(t) = A^{s+i}(t-1)$.

Figure 1: Tree PCP for CktReach instance (C, a, n), with respect to path $p = (w_1, a_1, \ldots, w_n, a_n)$.

C for any $t = 1, \ldots, n$, and (3) A_{t-1} and A_t assign consistent values to a_{t-1} for any $t = 2, \ldots, n$. This suggests a tree PCP outline, which we sketch in Fig. 1.

Given locally testable tree codes have been constructed in prior work [MRR25], it seems that to realize the outline from Fig. 1 we only need to make them locally correctable and to devise a corresponding sumcheck protocol that can be turned into a zero test.

To some extent, both ingredients can be derived somewhat generically in the standard setting, for any locally testable *block* code that builds on *tensoring*: Gur *et al.* [GRR20] prove that any tensor code already satisfies a relaxed notion of local correctability that is sufficient for PCP soundness [BGH+06], and Meir [Mei13] shows that the classic sumcheck protocol [LFKN92] can be applied to any tensor code. Since the code from [MRR25] also relies on tensoring, albeit of tree codes, we are able to successfully adapt the respective results from the literature to locally testable tree codes.

In Section 4, we recall the locally testable tree code construction from [MRR25] and prove that it satisfies relaxed local correctability (Lemma A.1).[3] Interestingly, using its relaxed local correctability, we are able to show that the code is *strongly* locally testable (Proposition 4.9), namely that there exists a local test that rejects any non-codeword with non-zero probability. This is an improvement compared to [MRR25], where we prove local testability in a weak sense, guaranteeing that a non-codeword is rejected with non-zero probability only if its distance from the code exceeds a certain threshold. Notably, strong local testability is crucial for the soundness analysis of the tree PCP. See further discussion in Section 4.1.1.

In Section 6, we build an interactive sumcheck protocol for tree code tensors (Section 6.1) then

[2] In actuality, E is not expected to be the all-zero string but rather to contain zeros only in a set of relevant locations where constraint evaluations reside. Otherwise, making few random queries to $\widetilde{E}$ and checking they are all zeros would have sufficed for small-enough soundness error, due to the distance of the code.

[3] In fact, we consider an intermediate construction from [MRR25] that does not satisfy tree distance but only a weaker notion of distance that suffices for our analysis. Our proofs of relaxed local correctability and strong local testability extend, however, to the locally testable tree codes of [MRR25] that satisfy (probabilistic) tree distance.

show how to convert it to a zero test proof oracle that is "incremental" (Lemma 6.1), which is important for obtaining a tree PCP.

While locally testable tree codes that are also relaxed locally correctable and equipped with a zero test take us close to tree PCPs, there are still two considerable gaps.

First, recall that we crucially require that the verifier can "evaluate" the local constraints from Fig. 1 over the assignments $A^1, \ldots, A^{3s}$ given their encodings $\widetilde{A}^1, \ldots, \widetilde{A}^{3s}$ ((iii)). To that end, the set of local constraints and the encodings must be compatible in structure.

Second, while relaxed local correctability is sufficient for soundness, we ultimately need a tree code that satisfies full-fledged local correctability for completeness. Recall tree PCPs (Definition 1.4) require that any accepting proof can be incrementally extended. Such an accepting proof may consist of a small amount of corruptions that goes undetected by the verifier yet hinders extending the proof via straight-forward tree code encoding. Given local correctability even in the presence of corruptions, we may apply the incremental encoding function of the tree code in a manner that is robust to bounded corruptions: instead of reading directly from the encoding generated thus far, use local correctability to read from the values defined by the closest codeword.

The tensor structure, using which we get local testability, is alone not sufficient to guarantee local correctability (unlike relaxed local correctability). Nevertheless, it does preserve it: the tensor product of locally correctable tree codes is locally correctable.

We close these two gaps by designing a tree code that, on the one hand, is locally correctable and, on the other, is well-structured to allow for constraint evaluation. In the rest of the overview, we elaborate on our design, and point the reader to Sections 4.2 and 5 for details.

2.3 PCP-Friendly Tree Codes?

BFLS PCPs build on low-degree extensions, facilitating algebraic techniques to evaluate constraints over the encoded messages, as the constraints are typically *arithmetized* and made algebraic.

For simplicity, think of the univariate special case of low-degree extensions, aka Reed-Solomon, as an example. These are *multiplication codes* [Mei13], where the encoding of the point-wise product of two words A^1 and A^2, is simply the point-wise product of the codewords $\widetilde{A}^1$ and $\widetilde{A}^2$. This, besides linearity, allows to evaluate any low-degree polynomial underneath codewords, in a point-wise manner: by reading few locations from $A^1, \ldots, A^{3s}$, the verifier can compute any location in the codeword $\widetilde{E}$ that encodes $E(t) = P(A^1(t), \ldots, A^{3s}(t))$, where P is any degree-3 polynomial. Consequently, using standard arithmetization techniques and assuming C is a 3-CNF circuit w.l.o.g., the verifier can evaluate the transition constraints (2.2) as we require.

The same algebraic structure additionally allows to evaluate consistency constraints (2.3): for instance, due to the *affine invariance* of Reed-Solomon codes, say over prime fields, a codeword that encodes a *shift* of an assignment A, i.e. $\hat{A}$ where $\hat{A}(t) = A(t-1)$, can be obtained by (circularly) shifting the codeword $\widetilde{A}$. This is sufficient since the consistency constraint between A^i and A^{s+i} (see 2.3) can be written as $E = A^i - \hat{A}^{s+i} \equiv 0$.

Things do not work that easily with tree codes: unlike with block codes, explicit "purely algebraic" tree code constructions with reasonable alphabet size are not known to exist, let alone ones that are also locally testable . The only explicit algebraic tree codes we are aware of either have alphabet size that grows exponentially in the message length[4] [Pud13, CHS18] or are heuristic [MS14, BCN21], namely where minimum distance is only conjectured. Even if we are willing to compromise on the latter, we do not know how to exploit the algebraic structure therein for our goals.

[4]While the code from [CHS18] is based on an algebraic design, the final construction with small alphabet involves combinatorial machinery that breaks a potentially useful structure and even makes the code non-linear.

2.4 The Base Tree Code

In light of the above, we turn to rely on a simple, generic, tree code construction by Schulman [Sch94] that can be based on *any* family of block codes that allows encoding messages of any length. The hope is that if we instantiate the construction with algebraic block codes, e.g. low-degree extensions, it inherits some of their desired properties and that these further propagate to the locally testable tree codes obtained by plugging-in the construction in the framework of [MRR25].

A codeword in Schulman's tree code is a concatenation of codewords from the underlying family of block codes, where each "block-codeword" encodes a certain chunk of the message.

The codeword corresponding to a length n message consists of $\lceil \log(n) \rceil$ "threads", where the k^{th} thread, for $k = 1, \ldots, \lceil \log(n) \rceil$, contains encodings of consecutive chunks of length 2^{k-1} in the message (see Fig. 2). Note that a codeword in the tree code might contain *only part* of some block-codewords, which will be eventually entirely included as the codeword grows.

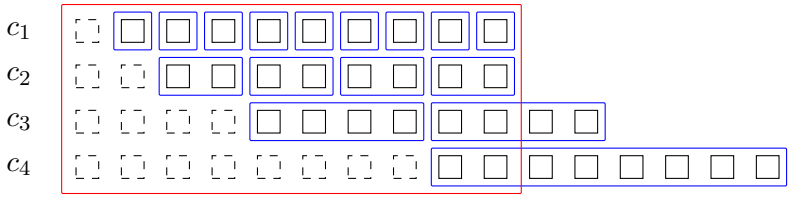

Figure 2: Encoding a message of length $n = 10$ with Schulman's original tree code [Sch94]. The codeword (red) consists of $\lceil \log n \rceil = 4$ threads, where thread c_k, for $k = 1, \ldots, 4$, contains block-codewords (blue) that encode chunks of the message of length 2^{k-1}. Any such block-codeword is split into 2^{k-1} equal-length symbols (black) that are added one column at a time to the tree-codeword.

Since a codeword in Schulman's construction is simply a concatenation of block-codewords, if the underlying block code is a linear multiplication code, then so is the obtained tree code. Consequently, the transition constraints can be evaluated under the tree code as described above, namely by evaluating low-degree polynomials locally, in a point-wise manner, within each of the block-codewords. We show this formally in Section 5.1.

2.5 Evaluating Consistency

Verifying the consistency constraints turns out to be much more challenging. In fact, a substantial part of the technical work put to achieve the main result of this paper is dedicated to this goal. In what follows we provide a simplified account, omitting many of the moving parts, and refer the reader to Section 5.2 for more intuition and formal details.

The difficulty in evaluating the consistency constraints stems from the fact that, in contrary to the transition constraints which are "point-wise", the consistency constraints involve values from different locations in different assignments. Recall that evaluation can be done by shifting an assignment A underneath its encoding by $t \mapsto t - 1$ to obtain an encoding of $\hat{A}$ (recall $\hat{A}(t) = A(t-1)$). These shifts are not directly compatible with the structure of the locally testable tree codes we consider. Roughly speaking, there are two sources of incompatibility corresponding to two "combinatorial layers" in the code: First, the block structure of Schulman's tree code (Fig. 2) and, second, the "flattened tensor" structure of the locally testable codes from [MRR25]. (Recall we do not directly use the tree code by Schulman to encode the assignments in the tree PCP since the code is not locally testable. The tree code is instead used as the base code to the locally testable tree code construction of [MRR25].)

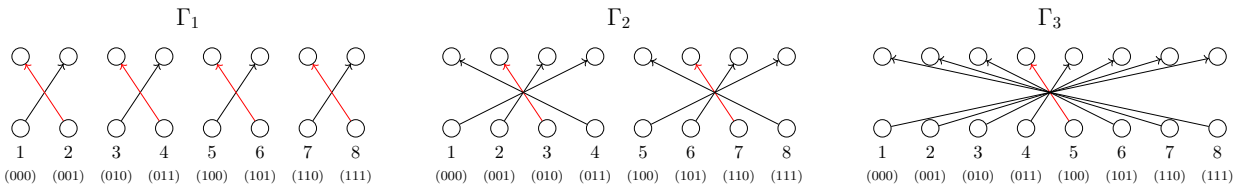

Figure 3: The shifts Γ_1, Γ_2 and Γ_3 over $\{1, \ldots, 8\}$. In red is the embedding of $t \mapsto t-1$, namely the edges going out from t satisfying $\Gamma_i(t) = t-1$ (the set Λ_i defined in Section 5.2). In parentheses are the coordinate labels, denoted $\mathrm{b}(t)$ in Section 5.2.

Naively shifting the assignment underlying a codeword $\widetilde{A}$ from Schulman's tree code entails "moving around" information across different block-codewords. The presumed minimum distance of the underlying block codes makes this impossible to do with a small number of queries. Instead, we implement the shifts $t \mapsto t-1$ by embedding them in a collection of different shifts (i.e. permutations) $\Gamma_i : \mathbb{N} \to \mathbb{N}$ over the coordinate space of A, described in Fig. 3. The shifts Γ_i are compatible with the combinatorial structure of the tree code in that they can be performed by first permuting blocks then permuting coordinates *within* each of the blocks (but never across different blocks). The latter is possible with existing algebraic block codes, specifically (univariate or multivariate) low-degree extension codes over extensions of GF(2) (Definition 4.16), which we use to instantiate the tree code construction.

We note that the shifts Γ_i are similar to permutations that appear in prior work on PCPs in a similar context, where routing techniques are used to design well-structured consistency constraints [Spi95, PS94, BGH+06].

One technical issue that arises in shifting an encoded assignment A is that it might involve block-codewords that appear only partially in the codeword $\widetilde{A}$ (see Fig. 2). We handle this by letting the prover write down the missing parts "in advance" to the PCP. While these parts are not included in the original codeword yet, we may consider an extension of Schulman's construction where codewords contain all block-codewords in their entirety. Such a code is a tree code that inherits the tree distance of the original construction, as well as its rate in an "amortized sense". Further, the tensor product of the extended code is locally testable and therefore the verifier is still able to test the validity of the given encodings.

The second main challenge in evaluating the consistency constraints has to do with the combinatorial structure underlying the locally testable tree codes of [MRR25]. The transformation in [MRR25] takes a base (linear) tree code and makes it locally testable in two steps:

(i) *Tensoring*: The m-fold tensor product of the tree code, for any $m > 1$, defines an "m-dimensional tree code", where messages are viewed as m-dimensional rectangles and are encoded to m-dimensional codewords by an encoding function that is "online in m dimensions".

(ii) *Flattening*: To obtain a tree code with an online encoding function in the standard sense, the tensor tree code is flattened by a monotone embedding of the high-dimensional coordinate space $\mathbb{N}^m$ to the one-dimensional coordinate space $\mathbb{N}$ (defined in Fig. 4, illustrated in Fig. 5).

The ability to shift messages encoded under the base code translates to the ability to shift m-dimensional messages encoded under the tensor code *along any of the dimensions* (Lemma 5.13). While this is sufficient to shift the flattened encoding of an assignment at most coordinates (Lemma 5.14), for many values of t, the shift $t \mapsto t-1$ in a (one-dimensional) assignment corresponds to a "jump" in its lifting to m-dimensions, for the m-dimensional coordinates where t and $t-1$ are embedded are not at all adjacent (Lemma 5.15).

We are nevertheless able to express the consistency constraints as consistency between adjacent coordinates in the m-dimensional tensors (Lemma 5.16). This is done by introducing auxiliary variables that act as "bridges" in-between, and are also encoded using the locally testable tree code and added to the tree PCP next to the assignments.

2.6 Making Schulman's Construction Locally Correctable

The last missing component for tree PCPs is (full-fledged) local correctability of the tree code which, recall, is essential for completeness in the presence of few corruptions. Recall also that tensoring preserves local correctability, therefore it is sufficient to attain it for the base tree code which, in our case, is based on Schulman's generic construction.

A natural attempt is to instantiate Schulman's construction with block codes that are locally correctable and hope that the property is passed on to the tree code. Conveniently, our choice of low-degree extension codes that facilitated constraint evaluation can give us local correctability under certain choices of parameters [GS92, Sud95] (while the univariate Reed-Solomon codes are sufficient for constraint evaluation, for local correctability we opt for multivariate low-degree extensions that generalize Reed-Muller codes).

Assume we want to recover the correct value of a symbol from a slightly corrupted Schulman codeword w. A symbol in w consists of symbols coming from supposed block-codewords (Fig. 2). While we can try to use the local correctability of the underlying block codes to recover the correct value from each of the blocks, nothing guarantees that these blocks are sufficiently close to valid codewords, even when w is close to a tree-codeword: w may be entirely corrupted over any of the sufficiently small blocks, in which case the recovery is not guaranteed to be correct.

Our strategy for locally correcting symbols in w is to extract the correct value by probing not only the blocks that contain them, but rather all larger blocks at subsequent levels that encode related parts of the message. The reasoning is that these blocks span an entire suffix of coordinates in w and, hence, if w is close to the tree code, then so must be the majority of these blocks to their respective block codewords.

This idea seems to require the block codes to possess a certain structure, allowing to read from a codeword that encodes a message x_1 by reading from a codeword corresponding to a larger message (x_1, x_2) of double the length. We observe that such a property, to some extent, can be attained by a yet more specific choice of low-degree extensions. It was shown by Reingold *et al.* [RRR16] that reading from a k-variate low-degree extension that systematically encodes a message x_i can be done given access to the $(k+1)$-variate low-degree corresponding to $(x_1, \ldots, x_d)$, where d is the individual degree of the additional variable (Proposition 4.19). For local correctability with reasonable parameters, d must be chosen to be a super-constant, deeming the above unsuitable to the original construction of Schulman, that considers blocks of doubling length.

For that reason, we in fact consider a generalization of Schulman's construction (Fig. 6), where the length of blocks encoded at level $k+1$ are $\mathrm{poly}(k)$-times larger than the blocks at level k and instantiate it with suitable family of low degree extensions that satisfy all of the necessary properties: multiplication and affine invariance for constraint evaluation, as well as local correctability and the above "inter-codeword" correlations. As a result, we obtain that a message of length n is encoded using $k = O(\mathrm{polylog}(n))$ levels and that the tree code has rate n^{γ}, for arbitrarily small $\gamma > 0$, and incurs a $1/\mathrm{polylog}(n)$ loss in distance compared to the distance of the underlying block codes.

3 Preliminaries

For $n \in \mathbb{N}$, we denote $[n] = \{1, \ldots, n\}$ and the n^{th} harmonic number by $H_n = \sum_{i=1}^{n} 1/i$. For a subset $J \subseteq [m]$, we denote by $1_J \in \{0,1\}^m$ the binary vector that is 1 at any $j \in J$ and 0 at any $j \notin J$. For m-dimensional coordinates $t = (t_1, \ldots, t_m)$ and $t' = (t'_1, \ldots, t'_m)$, we write $t \geq t'$ if $t_j \geq t'_j$ for all j. A (possibly infinite) set $I \subseteq \mathbb{N}^m$ is a *rectangle* if it can be written as $I = I_1 \times \cdots \times I_m$.

We often view a word $w \in \Sigma^n$ as a function $w : [n] \to \Sigma$, where $w(t)$ is the t^{th} symbol in w and, more generally, m-dimensional words $w \in \Sigma^{n_1 \times \cdots \times n_m}$ as $w : [n_1] \times \cdots \times [n_m] \to \Sigma$. For a function $f : I \to \Sigma$, we denote by $\mathrm{dom}(f) = I$ the domain of f and write $|f| = |I| \cdot \log |\Sigma|$. For a set S, we denote by $f_S : S \to \Sigma$ the restriction of f to $S \cap \mathrm{dom}(f)$.

For a distance function $\Delta = \{\Delta_n : \Sigma^n \times \Sigma^n \to \mathbb{R}\}$ and any string $w \in \Sigma^n$ and set $S \subseteq \Sigma^*$, we denote the distance between w and S by $\Delta(w, S) = \min_{w' \in S \cap \Sigma^n} \Delta_n(w, w')$. We sometimes consider distances between high-dimensional words where this notation generalizes naturally.

We say that an algorithm is efficient if it runs in time polynomial in the length of its input. We say that a function is efficiently computable if there exists an efficient algorithm that computes it. Algorithms in this paper are often oracle-aided, namely they are given access to an oracle. We assume algorithms always take the size of their oracles as input and omit this from the notation.

We say that a function $\epsilon : \mathbb{N} \to \mathbb{R}^+$ is negligible if $\epsilon(n) = o(1/n^c)$ for any constant c. We say that a multivariate function is negligible if it is negligible in its largest input.

3.1 Incremental Ensembles

For functions f, g, where $\mathrm{dom}(f) \subseteq \mathrm{dom}(g)$, we denote $f \preceq g$ and say f and g are *consistent*, if $g(t) = f(t)$ for all $t \in \mathrm{dom}(f)$. We use this notation particularly for words, namely functions over coordinates $[n]$ or, more generally $[n_1] \times \cdots \times [n_m]$, where in such a case we say that f is a *prefix* of g. For $f \preceq g$, we denote by $g|_f$ the restriction of g to $\mathrm{dom}(g) \setminus \mathrm{dom}(f)$ and, more generally for $f_1, \ldots, f_r \preceq g$, we let $g|_{f_1,\ldots,f_r}$ denote the restriction of g to $\mathrm{dom}(g) \setminus (\bigcup_j \mathrm{dom}(f_j))$.

We consider ensembles of functions that associate any $x \in \Gamma^{n_1 \times \cdots \times n_m}$ with a function f_x and are *monotone* in the sense that a prefix of x is always associated with a prefix of f_x.

Definition 3.1 (Monotone ensemble)**.** *An ensemble of functions*

$$\{f_x \mid x \in \Gamma^{n_1 \times \cdots \times n_m},\ n_j \in \mathbb{N}\ \forall j\}$$

is monotone *if for any $x \preceq x'$ it holds that $f_x \preceq f_{x'}$.*

We are interested in cases where f_x can be incrementally computed and, in particular, where the incremental computation is robust against bounded corruptions.

Definition 3.2 (Incremental Ensemble)**.** *We say that a monotone $\{f_x\}$ is* (T, L)-incremental *if for any $x : [n_1] \times \cdots \times [n_m] \to \Gamma$, letting x_j denote the restriction of x to $[n_1] \times \cdots \times [n_j - 1] \times \cdots \times [n_m]$ and $f_j := f_{x_j}$, it holds that $f_x|_{f_1,\ldots,f_m}$ is of size at most $L(n_1, \ldots, n_m)$ over Γ and there exists a deterministic algorithm that computes it in time $T(n_1, \ldots, n_m)$ given $x(n_1, \ldots, n_m)$ and oracle access to $f_1, \ldots, f_m$.*

We say that $\{f_x\}$ is (Δ, δ, T, L)-robustly incremental*, for a distance function Δ, if the algorithm outputs $f_x|_{f_1,\ldots,f_m}$, with probability all but negligible in n, even when given access to $f'_1, \ldots, f'_m$ such that $\Delta(f_j, f'j) \leq \delta(n_1, \ldots, n_m)$.*

3.2 Tree Codes

A *code* over an alphabet Σ is a subset of strings, namely *codewords*, $\mathrm{C} \subseteq \Sigma^*$. Typically, a code is associated with a 1-1 *encoding function* from a *message space* of size $|\mathrm{C}|$ to C. The standard notion of codes is that of *block codes*, where all codewords are of the same length.

Central to our work is the notion of *tree codes* [Sch93]. These are infinite codes that exhibit an online encoding function.

Definition 3.3 (Tree Code)**.** *A* tree code over alphabet $\Sigma = \{\Sigma_n\}$ *is an infinite collection of subsets* $\mathrm{TC} = \{\mathrm{TC}_n \subset (\Sigma_n)^n\}_{n\in\mathbb{N}}$*, where, for any* $n \in \mathbb{N}$*, it holds that* $\Sigma_{n-1} \subseteq \Sigma_n$ *and, for any codeword* $(c_1, \ldots, c_n) \in \mathrm{TC}_n$ *it holds that* $(c_1, \ldots, c_{n-1}) \in \mathrm{TC}_{n-1}$*.*

We often define a tree code via an injective *encoding function* which, overriding notation, we denote by $\mathrm{TC} = \{\mathrm{TC}_n : \Gamma^n \to \Sigma_n\}$. The function encodes a message $(x_1, \ldots, x_n) \in \Gamma^n$ by

$$\mathrm{TC}(x_1, \ldots, x_n) = (\mathrm{TC}_1(x_1), \mathrm{TC}_2(x_1, x_2), \ldots, \mathrm{TC}_n(x_1, \ldots, x_n)).$$

The corresponding code is $\{\mathrm{TC}(x) \mid x \in \Gamma^*\}$ and its rate is defined as $\rho(n) = \log|\Gamma|/\log|\Sigma_n|$. Note that the codewords of a tree code with a well-defined encoding function make a monotone ensemble of functions (Definition 3.1).

We say that tree code with an encoding function $\mathrm{TC} = \{\mathrm{TC}_n\}$ is *systematic* if $\Sigma_n \subseteq \Gamma \times \Sigma'_n$ for some Σ'_n and the n^{th} codeword symbol $c_n = \mathrm{TC}_n(x_1, \ldots, x_n)$ is always of the form (x_n, c'_n). We refer to the first part in any $c_n \in \Sigma_n$, which is over Γ, as the *systematic part*.

We sometimes work with tree codes where there is no well-defined encoding function yet it is always the case that there exists an *input alphabet* Γ where for any message $x \in \Gamma^n$, there exists a well-defined *set* of codewords $\mathrm{TC}(x)$ that encode x (Remark 4.14) and, additionally, for any codeword $c \in \mathrm{TC}$ there exists a well-defined message x that satisfies $c \in \mathrm{TC}(x)$. In this case, we say that the code is systematic if the n^{th} symbol of any codeword $c \in \mathrm{TC}(x)$ is of the form (x_n, c'_n).

We will always assume in this work that, given a word $w \in \Sigma^n$, it is possible to efficiently tell (in time polynomial in n) if $w \in \mathrm{TC}$ and, if so, to efficiently find the message x satisfying $w \in \mathrm{TC}(x)$. All of the tree codes that we consider satisfy this property.

Tree codes inherently fail to achieve Hamming distance, which is the standard in coding theory. The appropriate notion for tree codes is *tree distance*, which measures (relative) Hamming distance between two words starting from their first disagreement.

Definition 3.4 (Tree Distance)**.** *Let* Σ *be an alphabet and* $n \in \mathbb{N}$*. Let* $w, w' \in \Sigma^n$ *and let* $i^* = \min\{i : w_i \neq w'_i\}$ *(and* $i^* = 0$ *when* $w = w'$*). We define the* tree distance *between* w *and* w' *as*

$$\Delta_{\mathsf{T}}(w, w') = \Delta_{\mathsf{H}}(w_{\geq i^*}, w'_{\geq i^*}),$$

where Δ_{H} *denotes relative Hamming distance and* $w_{\geq i^*}$ *the suffix of* w *starting at position* i^**.*

We define linear tree codes similarly to [Pud13]. This is a special case of linear tree codes as sometimes defined in the tree code literature (e.g. [CHS18], where codes over general rings, not necessarily vector spaces, are considered) and equivalent to the notion of vector linear tree codes from [MRR25].

Definition 3.5 (Linear Tree Code)**.** *Let* $\mathbb{F} = \{\mathbb{F}(n)\}$ *denote a sequence of finite fields where* $\mathbb{F}(n-1)$ *is a subfield of* $\mathbb{F}(n)$ *for all* $n \in \mathbb{N}$*. Let* $L : \mathbb{N} \to \mathbb{N}$*. A* linear tree code over $\mathbb{F}$ *is a tree code with input alphabet* $\Gamma = \mathbb{F}(0)$ *and output alphabet* $\Sigma_n = \mathbb{F}(n)^{L(n)}$*, such that for any* $n \in \mathbb{N}$*, any* $c, c' \in \mathrm{TC}_n$ *and any* $\alpha, \beta \in \mathbb{F}(n)$*, it holds that* $\alpha \cdot c + \beta \cdot c' \in \mathrm{TC}_n$*.*

We often use $\mathbb{F}$ to denote the finite field $\mathbb{F}(n)$, rather than the sequence of fields, when n is clear from context. The following remark gives a characterization of any linear tree code as an infinite collection of linear block codes, of increasing block-length, each with block lower-triangular generator matrix.

Remark 3.6. *Let* $\mathrm{TC} = \{\mathrm{TC}_n \subset (\mathbb{F}(n)^{L(n)})^n\}$ *be a linear tree code over* $\mathbb{F} = \{\mathbb{F}(n)\}$ *with a well-defined encoding function. Then, for any* $n \in \mathbb{N}$, TC_n *is isomorphic (up to padding with zeros) to a linear block code of dimension* n *and length* $L(n) \cdot n$ *that has a block lower-triangular generator matrix* $G_n \in \mathbb{F}(n)^{(L(n)\cdot n)\times n}$, *where* $G_{n-1} \preceq G_n$

While tree distance is a powerful notion and captures the desired distance guarantee in many tree code applications, it is often difficult to work with. In particular, tree distance is not formally a distance function (i.e. a metric) as it does not satisfy the triangle inequality. In our analysis, we mostly use a different distance notion called *suffix distance* [MRR25], which is actually a distance function, can be usefully described as probability of disagreement at a random location (similarly to relative Hamming distance), and is strongly related to tree distance (Lemma 3.8). We present a definition of suffix distance which generalizes over high-dimensional words.

Definition 3.7 (m-dimensional Suffix Distance)**.** *Let* Σ *be an alphabet. Define* $H_n = \sum_{j=1}^{n} 1/j$ *for any* $n \in \mathbb{N}$ *and* $H_{n_1,\ldots,n_m} = \prod_j H_{n_j}$ *for* $n_1, \ldots, n_m \in \mathbb{N}$. *For any* $m \in \mathbb{N}$, $n_1, \ldots, n_m \in \mathbb{N}$ *and coordinate* $t = (t_1, \ldots, t_m) \in [N]$, *we define*

$$\sigma_{n_1,\ldots,n_m}(t) = \frac{1}{H_{n_1,\ldots,n_m}} \cdot \prod_{j=1}^{m} \frac{1}{n_j - t_j + 1}.$$

Notice that $\sigma_{n_1,\ldots,n_m} = \bigotimes_{j=1}^{m} \sigma_{n_j}$ *and, since* σ_n *is a probability density function, then so is* $\sigma_{n_1,\ldots,n_m}$. *We sometimes override notation and use* $\sigma_{n_1,\ldots,n_m}$ *to denote the corresponding distribution over* $[n_1] \times \cdots \times [n_m]$.

Let $w, w' \in \Sigma^{n_1 \times \cdots \times n_m}$. *We define the* suffix distance *between* w *and* w' *as*

$$\Delta_{\mathsf{S}}(w, w') = \Pr_{t \leftarrow \sigma_{n_1,\ldots,n_m}} [w_t \neq w'_t].$$

We further define $\omega_{\mathsf{S}}(w) = \Delta_{\mathsf{S}}(w, 0)$ *to be the* suffix weight *of* w.

The following lemma (special case of [MRR25, Lemma 5.2])[5] gives a lower bound on the (one-dimensional) suffix distance between any two words by the Hamming distance in any of its suffixes. This immediately implies a connection between suffix distance and tree distance.

Lemma 3.8 (Suffix Distance from Tree Distance)**.** *Let* $n \in \mathbb{N}$ *and* $w, w' \in \Sigma^n$. *Assume there exists* $i^* \in [n]$ *such that* $\Delta_{\mathsf{H}}(w_{\geq i^*}, w'_{\geq i^*}) \geq \delta$. *Then, it holds that*[6]

$$\Delta_{\mathsf{S}}(w, w') \geq \frac{1}{H_n} \cdot \big(\delta - o(1)\big),$$

where, recall, H_n *is the* n^{th} *Harmonic number. In particular, for any* $w, w' \in \Sigma^n$,

$$\Delta_{\mathsf{S}}(w, w') \geq \frac{1}{H_n} \cdot \big(\Delta_{\mathsf{T}}(w, w') - o(1)\big).$$

[5] The statement in the lemma from [MRR25] refers to a weaker lower bound of $\delta/H_n - o(1)$, but its proof actually implies the above stronger version.

[6] Asymptotic notation is with respect to n.

3.3 Local Testability and Local Correctability

We recall the notions of local testability [BLR93, RS96, GS06] and local correctability [BFLS91, KT00, Sud01] for codes. While the standard notions are defined for block codes and w.r.t. Hamming distance, we consider their natural generalization to codes that are possibly infinite and/or that are over high-dimensional coordinate space (namely subsets of high-dimensional words), and w.r.t. to any given distance metric, akin to [MRR25]. By default, we consider the strong notion of local testability where rejection of a non-codeword increases proportionally to its distance from the code.

Definition 3.9 (Strong Local Testability)**.** *Let $m \in \mathbb{N}$ and let Δ be a distance function over m-dimensional words. For $q : \mathbb{N}^m \to \mathbb{N}$ and $\epsilon : \mathbb{N}^m \to \mathbb{R}$, we say that a code $\mathrm{C} = \{\mathrm{C}_{n_1,\ldots,n_m} \subset \Sigma^{n_1 \times \cdots \times n_m}\}$ is (Δ, q, ϵ)-*locally testable*, or (Δ, q, ϵ)-*LTC *for short, if there exists a randomized algorithm T that has oracle access to a word $w : [n_1] \times \cdots \times [n_m] \to \Sigma$ and satisfies:*

- *(Completeness:) If $w \in \mathrm{C}$, then $\Pr[T^w = 1] = 1$.*
- *(Soundness:) There exists a negligible function ν such that, for any w,*

$$\Pr[T^w = 0] \geq \min\Big(\epsilon(n_1, \ldots, n_m) \cdot \Delta(w, \mathrm{C}),\ 1 - \nu(n_1, \ldots, n_m)\Big).$$

- *(Query complexity:) T makes at most $q(n_1, \ldots, n_m)$ queries to its oracle.*

Definition 3.10 (Local Correctability)**.** *Let $m \in \mathbb{N}$ and let Δ be a distance function over m-dimensional words. For $q : \mathbb{N}^m \to \mathbb{N}$ and $\epsilon : \mathbb{N}^m \to \mathbb{R}$, we say that a code $\mathrm{C} = \{\mathrm{C}_{n_1,\ldots,n_m} \subset \Sigma^{n_1 \times \cdots \times n_m}\}$ is $(\Delta, q, \delta, \epsilon)$-*locally correctable*, or $(\Delta, q, \delta, \epsilon)$-*LCC *for short, if there exists a randomized algorithm $\mathcal{C}$ that has oracle access to a word $w : [n_1] \times \cdots \times [n_m] \to \Sigma$, takes as input a coordinate $(t_1, \ldots, t_m) \in [n_1] \times \cdots \times [n_m]$, and satisfies:*

- *(Completeness:) If $w \in \mathrm{C}$, then $\Pr[\mathcal{C}^w(t_1, \ldots, t_m) = w(t_1, \ldots, t_m)] = 1$.*
- *(Soundness:) If $\Delta(w, c) \leq \delta$ for some $c \in \mathrm{C}$, then*

$$\Pr[\mathcal{C}^w(t_1, \ldots, t_m) = c(t_1, \ldots, t_m)] \geq 1 - \epsilon(n_1, \ldots, n_m).$$

- *(Query complexity:) $\mathcal{C}$ makes at most $q(n_1, \ldots, n_m)$ queries to its oracle.*

We say that C *is simply (Δ, q, δ)-LCC if ϵ is negligible.*

Note that the error in local correction can be always made negligible by repetition at the cost of polylogarithmic blow-up in query complexity.

4 The Tree Code

At the core of our tree PCP is an encoding of a witness to the CktReach statement using a tree code. For soundness, we want this code to be locally testable (LTC, Definition 3.9). Additionally, to allow for robust encoding as required by our notion of tree PCPs, we want the online encoding function corresponding to the tree code to be robust against bounded corruptions in the codeword produced so far. That is, we want the locally testable tree code to admit a robustly incremental ensemble (Definition 3.2). We achieve robust encoding by constructing a locally correctable tree code and using it as a building block in our tree LTC.

4.1 Local Testability via Tensoring

Locally testable tree codes were built in [MRR25] using *code tensoring*, which is a general strategy to obtain local testability in the standard setting of block codes [BS04, Mei09, Vid15, KMRS17]. *Tensor tree codes* are defined by the tensor product operation over linear tree codes.

Definition 4.1 (Tensor Product of Tree Codes)**.** *Let* $\mathrm{TC} = \{\mathrm{TC}_n \subset (\mathbb{F}(n)^{L(n)})^n\}$ *be a linear tree code over* $\mathbb{F} = \{\mathbb{F}(n)\}$ *and let* $m \in \mathbb{N}$*. The* m-fold tensor product *of* TC*, which we denote by* TC^m*, is defined as the ensemble* $\mathrm{TC}^m = \{\mathrm{TC}^m_{n_1,\ldots,n_m} \subset \mathbb{F}(n)^{L(n_1)n_1 \times \cdots \times L(n_m)n_m} \mid n = \max_j n_j\}$ *where, for any* $n_1, \ldots, n_m \in \mathbb{N}$*,*

$$\mathrm{TC}^m_{n_1,\ldots,n_m} = \mathbf{Span}\left(\{c_1 \otimes \cdots \otimes c_m \mid \forall j,\ c_j \in \mathrm{TC}_{n_j}\}\right).$$

We sometimes shorthand $L(n_j)$ by L_j and $L(n_1) \times \cdots \times L(n_m)$ by L^m, when $n_1, \ldots, n_m$ are clear from context. In particular, we sometimes denote the alphabet of TC^m by $\mathbb{F}^{L^m}$.

Similarly to a standard tensor code, an alternative definition for a tensor tree codes is the set of all tensors where the restriction to any column parallel to any of the dimensions is a codeword in the base code.

Remark 4.2 (Combinatorial Characterization of Tensor Tree Codes)**.** *Any* $c : [n_1] \times \cdots \times [n_m] \to \mathbb{F}^{L_1 \times \cdots \times L_m}$ *is in* TC^m *if and only if any restriction of* c *to a column over* $\mathbb{F}$ *is a codeword in* TC*. Formally, if* c *is viewed as a function from* $[L_1 n_1] \times \cdots \times [L_m n_m]$ *to* $\mathbb{F}$*, then, for any* $j \in [m]$ *and any* $i_1, \ldots, i_{j-1}, i_j, \ldots, i_m$*, where* $i_{j'} \in [L_{j'} n_{j'}]$*, the column* $d : [L_j n_j] \to \mathbb{F}$ *where* $d(i) = c(i_1, \ldots, i_{j-1}, i, i_{j+1}, \ldots, i_m)$ *is in* TC*, when viewed as* $d : [n_j] \to \mathbb{F}^{L_j}$*.*

The above characterization induces an incremental encoding algorithm for tensors of tree codes with explicit encoding function that, to compute a symbol in the codeword at a given coordinate $(n_1, \ldots, n_m)$ reads only $n_1 + \cdots + n_m$ codeword symbols from coordinates $(t_1, \ldots, t_m) \leq (n_1, \ldots, n_m)$. The algorithm is obtained by applying the encoding algorithm of TC along each of the m directions, one at a time, to compute the last symbol in the axis-parallel column passing through $(n_1, \ldots, n_m)$. The total runtime of such encoding is $\sum_j (\prod_{j'<j} L_{j'}) \cdot T(n_j)$.

Lemma 4.3 (Incremental Encoding of Tensors)**.** *Let* TC *be a linear tree code where the codewords form a* (T, L)*-incremental ensemble (Definition 3.2). Then, the codewords of* TC^m *form an* $(O(L(n)^m T(n)), L(n)^m)$*-incremental ensemble, where* $n = \max_j n_j$*.*

Additionally, if TC *is systematic then so is the encoding of* TC^m*.*

Tree distance in the base code TC translates into suffix distance in TC^m (Definition 3.7).

Proposition 4.4 (Lemma 5.3 in [MRR25])**.** *If* TC *has tree distance* $\delta(n)$*, then* TC^m *has suffix distance* $\left((\delta(n) - o(1))/H_n\right)^m$ *over words of length* $n_1 \times \cdots \times n_m$*, where* $n = \max_j n_j$ *and, recall,* H_n *is the* n^{th} *harmonic number.*

Additionally, [MRR25, Corollary 5.5] show the existence of a tester that can tell whether a given m-dimensional word is in TC^m, by reading a sublinear number of locations from it. The tester achieves the strong notion of local testability w.r.t. suffix distance, where the rejection probability of any non-codeword grows with its distance from the code, even if the latter is arbitrarily small (see Definition 3.9).

Theorem 4.5 (Local Testability of TC^m)**.** *Let* $m \in \mathbb{N}$ *and let* TC *be a linear tree code over* $\mathbb{F}$ *with constant tree distance. Then,* TC^m *is a* $(\Delta_{\mathsf{S}}, q, \epsilon)$*-LTC for* $\epsilon(n_1, \ldots, n_m) = \Omega(1/\log^m(n))$ *and* $q(n_1, \ldots, n_m) = O(n^2)$*, where* $n = \max_j n_j$*.*

Although the tensor product of a linear tree code TC^m is a well-defined code over $\mathbb{F}^{\mathbb{N}^m}$, it does not directly constitute a tree code (unlike the tensor product of block codes which is a block code). In particular, the above definition does not induce an online encoding function in the sense required by tree codes, but only a more general notion of "online encoding in m dimensions", where codeword symbols can be encoded given the previous message symbols *along all m dimensions*.

The tensor product TC^m can be "flattened" and turned into a tree code by embedding the coordinates of $\mathbb{N}^m$ onto the one-dimensional timeline, namely $\mathbb{N}$.

We say that a 1-1 mapping $\varphi : \mathbb{N} \to \mathbb{N}^m$ is *monotone* if $\varphi(t) \leq \varphi(t')$ implies $t \leq t'$ for all $t, t' \in \mathbb{N}$. (Recall $(t_1, \ldots, t_m) \leq (t'_1, \ldots, t'_m)$ iff $t_j \leq t'_j$ for all j.) Note that such a mapping defines a full order over the coordinates in $\mathbb{N}^m$ that is consistent with the standard partial order over $\mathbb{N}^m$.

While any monotone mapping φ can be used to make TC^m a tree code, [MRR25] chooses a specific mapping with useful structure that preserves the local testability of TC^m, yielding a flattened code that is locally testable. The mapping is natural and orders the coordinates in $\mathbb{N}^m$ by their L_∞-norm, where ties are resolved recursively over lower dimensions.

Concretely, let φ^m denote the mapping to m dimensions (we omit m when it is clear from context). Then, $\varphi^1(t) = t$ is the only monotone mapping from $\mathbb{N}$ to itself. For any $n \in \mathbb{N}$, φ^2 maps $\{1, \ldots, n^2\}$ to the square of coordinates $t = (t_1, t_2) \in \mathbb{N}^2$ satisfying $L_\infty(t) \leq n$, i.e. $[n]^2$, as follows: First, recursively map $\{1, \ldots, (n-1)^2\}$ to the square $[n-1]^2$. Then, map $\{(n-1)^2+1, \ldots, n(n-1)\}$ to the row of coordinates $\{n\} \times [n-1]$ using φ^1 and, next, $\{n(n-1)+1, \ldots, n^2-1\}$ to the column $[n-1] \times \{n\}$ using φ^1 again. Lastly, map n^2 to the corner (n, n). See the top row in Fig. 5.

Over 3 dimensions, φ^3 is defined similarly, where the coordinates in $[n]^3$ are covered by first mapping recursively into $[n-1]^3$, then using φ^2 to map into the three planes of coordinates adjacent to $[n-1]^3$, one at a time in a lexicographic order, then using φ^1 to map into the three lines adjacent to these planes. Lastly, n^3 is mapped to (n, n, n). See the bottom row in Fig. 5

In Fig. 4, we formally define φ^m by describing a recursive procedure that traverses over $\mathbb{N}^m$ and maps each coordinate to its order in the traversal (determined by the value of a global counter that increases by 1 every time it maps a coordinate). Actually, we describe a procedure that traverses over $[n]^m$, for any $n \in \mathbb{N}$, and defines a mapping $\varphi^m_n : [n^m] \to [n]^m$. φ^m is uniquely defined by the φ^m_n since the latter are consistent. We stress that both φ and φ^{-1} are computable in time polynomial in the length of their input (and polylogarithmic in the length of the codeword).

Definition 4.6 (Flattening of a Tensor Tree Code [MRR25])**.** *Let* TC *be a linear tree code over* $\mathbb{F}$ *and let* TC^m *be its m-fold tensor tree code. Let* $\varphi : \mathbb{N} \to \mathbb{N}^m$ *be the mapping from Fig. 4. The* flattening *of* TC^m*, denoted by* $\overline{\mathrm{TC}^m}$*, consists of all* $w \in (\mathbb{F}^{L^m})^*$ *for which there exists* $W \in \mathrm{TC}^m$ *such that* $W(\varphi(t)) = w(t)$ *for all* $1 \leq t \leq |w|$*.*

The code $\overline{\mathrm{TC}^m}$ exhibits an online encoding function and is therefore a tree code: To encode the next symbol, place it in the m-dimensional coordinate space using φ and use the incremental encoding of TC^m (Lemma 4.3) over the rectangular tensor that ends at the new coordinate (the rectangle consists of past symbols by the monotonicity of φ). In a codeword of length n, the dimensions of the tensor are all bounded by $O(n^{1/m})$ by the construction of φ. Consequently, online encoding of $\overline{\mathrm{TC}^m}$ is incremental with complexity proportional to the encoding of a $n^{1/m}$-long codeword of TC (due to Lemma 4.3).

Proposition 4.7 ([MRR25])**.** *For any $m \in \mathbb{N}$ and any linear tree code* TC*, the flattened tensor code* $\overline{\mathrm{TC}^m}$ *is a tree code. Further, if* TC *is (T, L)-incremental, then* $\overline{\mathrm{TC}^m}$ *is* $(O(L(n^{1/m})^m T(n^{1/m})), L(n^{1/m})^m)$*-incremental.*

We note that despite $\overline{\mathrm{TC}^m}$ being syntactically a tree code, it does not attain tree distance. In [MRR25], $\overline{\mathrm{TC}^m}$ is bootstrapped to a code that has (probabilistic) tree distance by relying on the

The Mapping $\varphi_n^m : [n^m] \to [n]^m$

0. Start a global counter $t = 1$.
1. For $d = 0, \ldots, m-1$,

   For all $J \in \binom{m}{d}$ in lexicographic order,

   1.1. Let $\mathbf{n}_J^{(0)} \in \mathbb{N}^m$ denote the coordinate that is n at $j \in J$ and 1 anywhere else.

   1.2. Let $\mathbf{n}_J^{(1)} \in \mathbb{N}^m$ denote the coordinate that is n at $j \in J$ and $n-1$ anywhere else.

   1.3. Map the next $(n-1)^{m-d}$ integers to $\{\mathbf{t} \in \mathbb{N}^m \mid \mathbf{n}_J^{(0)} \leq \mathbf{t} \leq \mathbf{n}_J^{(1)}\}$ recursively via φ_{n-1}^{m-d} over dimensions $[m] \setminus J$.
2. Map $t \mapsto (n, \ldots, n)$ then increase t by 1.

Figure 4: The mapping $\varphi^m = \{\varphi_n^m\}$ onto $\mathbb{N}^m$ used to flatten TC^m in Definition 4.6. The lexicographic order over $\binom{m}{d}$ is the standard lexicographic order over representations of the subsets as sorted strings in $[m]^d$.

suffix distance in TC^m and using further machinery. For our tree PCPs, tree distance by itself is not necessary and, therefore, we can use $\overline{\mathrm{TC}^m}$ directly. Consequently, our PCP analysis is largely based on the minimal suffix distance in the underlying tensor code TC^m.

A key property of the mapping φ, that in [MRR25] was crucial to convert Theorem 4.5 to a local test for the flattened code $\overline{\mathrm{TC}^m}$ and will play an important role in our PCP construction, is the fact that any flattened codeword can be represented as a merge of $O(1)$ codewords in TC^m. To formalize, let us denote the m-dimensional coordinate set of a flattened codeword of length n by $I^m(n) = \{\varphi^m(t) \mid t \leq n\}$. We omit m when it is clear from context, which is usually the case.

Proposition 4.8 ([MRR25])**.** *For any $n \in \mathbb{N}$ and $m \in \mathbb{N}$, there exists a collection of 2^m rectangles $I_r = [n_1^r] \times \cdots \times [n_m^r]$ such that $I^m(n) = \bigcup_{r=1}^{2^m} I_r$ and $n_j^r \leq \lceil n^{1/m} \rceil$ for all r and j. We call the set $\{I_r\}$ the* rectangle cover *of $I(n) := I^m(n)$ and denote it by $\mathbf{Rect}(n) := \mathbf{Rect}^m(n)$.*

Notation. We introduce the following notation related to tensor tree codes and their flattening, that will facilitate exposition in the sequel: For a word $w : [n] \to \Sigma$ (typically from the codeword or message space of $\overline{\mathrm{TC}^m}$) and a rectangle $I \subseteq I(n)$, we denote by $w_I : I \to \Sigma$ the word defined by $w_I(t) = w(\varphi^{-1}(t))$, where φ is the mapping from Fig. 4.

4.1.1 A Strong Local Test for The Flattened Code

In [MRR25] we show that a flattened code $\overline{\mathrm{TC}^m}$ is locally testable by proving that if a word w of length n is far from $\overline{\mathrm{TC}^m}$ in tree distance, then there exists a rectangle I in the rectangle cover $\mathbf{Rect}(n)$ (Proposition 4.8) over which w is far from TC^m in *suffix distance*. Since there is a constant number of rectangles in $\mathbf{Rect}(n)$, a test for $\overline{\mathrm{TC}^m}$ can be then obtained by performing the local test for TC^m (Theorem 4.5) over each of the rectangles in $\mathbf{Rect}(n)$ separately.

Translating tree distance in w to suffix distance in the furthest rectangle, however, incurs a significant loss in the distance. As a result, the test from [MRR25] is proven to catch a codeword with non-zero probability only if it is far enough from the code. In particular, the test is not

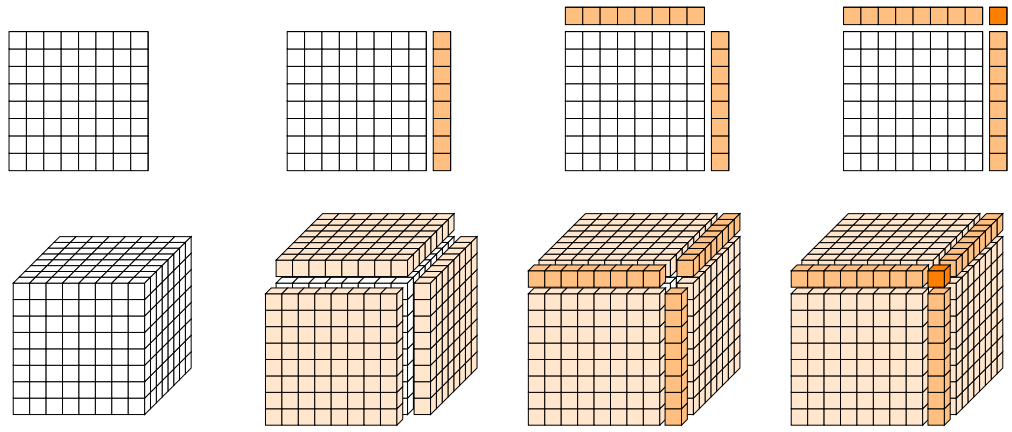

Figure 5: A visualization of the mapping φ^m, for $m = 2$ and $m = 3$, over coordinates with L_∞-norm $n = 9$. In orange are the recursive calls to the mapping over lower dimensions, ordered from left to right. A formal definition of φ^m is given in Fig. 4.

known to realize *strong local testability* where non-zero rejection probability is required for any non-codeword with arbitrarily little corruptions. In fact, it may be possible that a word w is never rejected by the test although its distance from the code exceeds its error-correction radius, in which case we cannot talk about a unique closest codeword to w. In contrary, a well-defined closest codeword for any w that passes the local test is crucial to our PCP analysis.

The reason for this limitation in the [MRR25] test is that all rectangles in the cover of w might be close to the code albeit each to a different closest codeword. Indeed, if one can argue that all closest codewords are consistent then the existence of a close codeword to w is easily implied.

We observe that we can bootstrap the test to a strong local test only if we could recover the values in the closest codeword over any rectangle. The idea is simple: after applying the local test over each of the rectangles, test whether they are close to consist codewords by recovering a few random locations where the rectangles overlap.

The ability to recover values from the closest codeword is precisely local correctability (Definition 3.10). We show how to make the tensor code locally correctable in Section 4.2, by building a concrete locally correctable tree code and plugging it in the tensor construction. This would already give us a tree code $\overline{\mathrm{TC}^m}$ that satisfies strong local testability.

We observe, however, that a relaxed notion of local correctability [GRR20] can be attained *generically* for tensor tree codes. In the relaxed notion, the recovery is allowed to fail (and output $\perp$) if it reads from a non-codeword. This is sufficient for the sake of local testability since, when the recovery fails, the test can immediately reject. (We stress that proper local correctability, that holds for a certain choice of TC, is still important to us for robust incremental encoding. We use relaxed local correctability here for the sake of generality.)

In Appendix A, we prove that any tensor tree code is relaxed locally correctable w.r.t. suffix distance, following a similar proof for tensor block codes w.r.t. Hamming distance from [GRR20].

We obtain a local tester that tests whether there exists a unique codeword $c \in \overline{\mathrm{TC}^m}$ that is close to w over all rectangles $I \in \mathbf{Rect}(n)$ *simultaneously*, and formulate the statement as such. We stress, however, that our tester also satisfies *strong local testability for tree distance*, due to the connections made in [MRR25] between tree distance in flattened codewords and the suffix distance in their furthest rectangle, which we recall in Proposition 4.10.

Proposition 4.9 (Local Testability of $\overline{\mathrm{TC}^m}$)**.** *Let $m \in \mathbb{N}$ and let* TC *be a linear tree code with*

tree distance δ. Then, $\overline{\mathrm{TC}^m}$ is $(\overline{\Delta_{\mathsf{S}}}, q, \epsilon)$-LTC with $q = O(\log(n)^{2m+2} \cdot n^{2/m}/\delta(n^{1/m})^m)$ and $\epsilon(n) = \Omega(1/\log^m(n))$, where the distance $\overline{\Delta_{\mathsf{S}}}$ over any $w, w' \in \Sigma^n$ is defined by

$$\overline{\Delta_{\mathsf{S}}}(w, w') := \max_{I \in \mathbf{Rect}(n)} \Delta_{\mathsf{S}}(w_I, w'_I).$$

Proof. Denote $\delta := \delta(n^{1/m})$. The tester applies the local test from Theorem 4.5 over each of the 2^m rectangles in the cover $\mathbf{Rect}(n)$, $\lambda = \log^{2m+2}(n)/\delta^m$ times, and rejects if any of the tests does. Then, for any two rectangles $I, I' \in \mathbf{Rect}(n)$, the tester repeats the following λ times:

1. Sample a coordinate $t \leftarrow \sigma_{I \cap I'}$ (Definition 3.7).
2. Apply the relaxed local corrector for TC^m (Lemma A.1) twice:
   - 2.1 Over w_I with input coordinate t to obtain a symbol y.
   - 2.2 Over $w_{I'}$ with input coordinate t to obtain a symbol y'.
3. If $\perp \in \{y, y'\}$ or $y \neq y'$, reject.

If all tests pass, the tester accepts.

Completeness of the test is by inspection. For $I_r \in \mathbf{Rect}(n)$, let $c^r : I_r \to \mathbb{F}^{L^m}$ be the codeword $c^r \in \mathrm{TC}^m$ minimizing $\Delta_{\mathsf{S}}(c^r, w_{I_r})$. We may assume that $\Delta_{\mathsf{S}}(c^r, w_{I_r}) < (\delta/2H_n)^m$ for all r since, otherwise, the local test over I_r rejects with probability at least $\epsilon(n) \cdot (\delta/2H_n)^m = \Omega(\delta^m/\log^{2m}(n))$ in any of the tests (Theorem 4.5), and with probability at least $1 - (1 - \Omega(\delta^m/\log^{2m}(n)))^\lambda = 1 - e^{-\Omega(\log^2(n))}$ in at least one of the tests.

By the relaxed local correctability of TC^m (Lemma A.1), we may assume, then, that the local corrector always outputs the correct symbol or $\perp$, since this occurs with probability all but negligible. If the corrector ever outputs $\perp$ the test rejects, thus we may ignore such cases.

If there exist I_r, I_q s.t. $c^r_{I_r \cap I_q} \not\equiv c^q_{I_r \cap I_q}$, then by the suffix distance of the code (Lemma 3.8) $\Delta_{\mathsf{S}}(c^r_{I_r \cap I_q}, c^q_{I_r \cap I_q}) \geq (\delta/H_n)^m$ and, by our assumptions, the local corrector will output two different symbols with probability $(\delta/H_n)^m$ for every choice of t in the iteration over $I = I_r$ and $I' = I_q$. The probability that it outputs different symbols for at least one coordinate out of the λ is then $1 - (1 - (\delta/H_n)^m)^\lambda = 1 - e^{-\Omega(\log^m n)}$.

If $c^r_{I_r \cap I_q} \equiv c^q_{I_r \cap I_q}$ for all I_r, I_q, then the well-defined codeword $c \in \overline{\mathrm{TC}^m}$ that is consistent with (the flattening of) all c^r is the closest codeword to w in $\overline{\Delta_{\mathsf{S}}}$ distance. In particular, there exists a rectangle $I \in \mathbf{Rect}(n)$ such that $\Delta_{\mathsf{S}}(w_I, c_I) = \overline{\Delta_{\mathsf{S}}}(w, c)$ and, hence, the local test over I will reject with probability at least $\epsilon(n) \cdot \overline{\Delta_{\mathsf{S}}}(w, c)$ (Theorem 4.5). □

Strong local testability of $\overline{\mathrm{TC}^m}$ w.r.t. $\overline{\Delta_{\mathsf{S}}}$ (Proposition 4.9) immediately implies strong local testability w.r.t. tree distance Δ_{T} (Theorem 1.6) by the following upper bound on tree distance, which follows by combining Lemma 5.2 and Claim 6.11 in [MRR25], or more directly, by combining the former with Lemma B.1.

Proposition 4.10. *For any $w, w' \in \Sigma^n$, $\overline{\Delta_{\mathsf{S}}}(w, w') \geq \Omega\left(\frac{1}{H_n^m}\right) \cdot \Delta_{\mathsf{T}}(w, w')$.*

4.1.2 Robust Incremental Encoding

Incremental encoding of $\overline{\mathrm{TC}^m}$ codewords can be performed using the incremental encoding algorithm for TC^m (Proposition 4.7). Recall, however, our goal is to devise a robust incremental encoding algorithm, that computes the next symbol in the encoding even when reading from a corrupted codeword. This corresponds to the notion of robust incrementality from Definition 3.2.

We achieve robust incrementality assuming the underlying code is locally correctable (Definition 3.10). Robust encoding can be performed by applying the incremental encoding algorithm, except instead of reading directly from the corrupted codeword, we invoke the local correction procedure to recover the correct codeword values.

In the following lemma, we show that the tensoring operation preserves local correctability w.r.t. suffix distance, implying that in order to attain a code TC^m (or $\overline{\mathrm{TC}^m}$) that is locally correctable (and therefore robustly incremental), it is sufficient to build on a locally correctable tree code TC. A similar result for tensors of block codes, where local correctability is considered w.r.t. Hamming distance, is implicit in the literature (e.g. in [GRR20]).

Lemma 4.11 (Tensoring Preserves Local Correctability)**.** *Let* $\mathrm{TC} = \{\mathrm{TC}_n \subset (\mathbb{F}(n)^{L(n)})^n\}$ *be a linear tree code that is* $(\Delta_{\mathsf{S}}, q, \delta', \epsilon)$*-LCC. Then, for any* $m \in \mathbb{N}$, TC^m *is* $(\Delta_{\mathsf{S}}, q', \delta', \epsilon')$*-LCC with* $q'(n_1, \ldots, n_m) = \prod_j L(n_j)^m q(n_j)$, $\delta'(n_1, \ldots, n_m) = \prod_j \delta(n_j)$ *and* $\epsilon'(n_1, \ldots, n_m) = m \cdot q' \cdot \epsilon$.

Proof. Recall that a symbol in a codeword $c \in \mathrm{TC}^m$ of dimensions $n_1 \times \cdots \times n_m$ is in $\Sigma = \mathbb{F}^{L_1 \times \cdots \times L_m}$, where $L_j = L(n_j)$ and $\mathbb{F} := \mathbb{F}(n)$ for $n = \max_j n_j$.

For a word $w : [n_1] \times \cdots \times [n_m] \to \Sigma$, we denote by $w^{\langle i_1, \ldots, i_m \rangle} : [n_1] \times \cdots \times [n_m] \to \mathbb{F}$, for $(i_1, \ldots, i_m) \in [L_1] \times \cdots \times [L_m]$, the straight-forward decomposition of w into words over $\mathbb{F}$: Any symbol $w(t_1, \ldots, t_m) \in \mathbb{F}^{L_1 \times \cdots \times L_m}$ contains the field element $w^{\langle i_1, \ldots, i_m \rangle}(t_1, \ldots, t_m)$ at location $(i_1, \ldots, i_m)$.

To simplify notation in what follows, we use $w^{\langle i_2, \ldots, i_m \rangle}$ to denote the word over $\mathbb{F}^{L_1}$ defined by $w^{\langle i_2, \ldots, i_m \rangle}(t_1, \ldots, t_m) = (w^{\langle 1, i_2, \ldots, i_m \rangle}(t_1, \ldots, t_m), \ldots, w^{\langle L_1, i_2, \ldots, i_m \rangle}(t_1, \ldots, t_m))$ and use $w^{\langle i_1 \rangle}$ to denote the word over $\mathbb{F}^{L_2 \times \cdots \times L_m}$ defined by $w^{\langle i_1 \rangle}(t_1, \ldots, t_m) = \left(w^{\langle i_1, i'_2, \ldots, i'_m \rangle}(t_1, \ldots, t_m)\right)_{(i'_2, \ldots, i'_m) \in [L_2] \times \cdots \times [L_m]}$.

Any symbol in w may be viewed as a collection of $L_2 \times \cdots \times L_m$ "column symbols" from $\{w^{\langle i_2, \ldots, i_m \rangle}\}$ that are in $\mathbb{F}^{L_1}$ and parallel to the first axis, or as a collection of L_1 "row symbols" from $\{w^{\langle i_1 \rangle}\}$ that are in $\mathbb{F}^{L_2 \times \cdots \times L_m}$ and orthogonal to the first axis.

Consider the following corrector algorithm for TC^m. On input a coordinate $(t_1, \ldots, t_m)$ and oracle $w : [n_1] \times \cdots \times [n_m] \to \Sigma$, the corrector invokes the local corrector for TC with input t_1 over each of the "columns" $w^{\langle i_2, \ldots, i_m \rangle}(\cdot, t_2, \ldots, t_m) : [n_1] \to \mathbb{F}^{L_1}$, for $(i_2, \ldots, i_m) \in [L_2] \times \cdots \times [L_m]$. However, instead of reading a symbol $w^{\langle i_2, \ldots, i_m \rangle}(u_1, t_2, \ldots, t_m)$ from its oracle directly, it recursively extracts each of the L_1 field elements therein using the local corrector for TC^{m-1} over the corresponding "row" $w^{\langle i_1 \rangle}(u_1, \cdot, \ldots, \cdot) : [n_2] \times \cdots \times [n_m] \to \mathbb{F}^{L_2 \times \cdots \times L_m}$.

Completeness is straight-forward and query complexity for m-dimensional tensors is given by the recursive formula $Q_m(n_1, \ldots, n_m) = \prod_j L_j \cdot q(n_1) \cdot Q_{m-1}(n_2, \ldots, n_m) = \left(\prod_j L_j\right)^m \prod_j q(n_j)$.

For soundness, let $E_m(n_1, \ldots, n_m)$ denote the error probability of the local corrector for TC^m over dimension $n_1 \times \cdots \times n_m$. Assume that $\Delta_{\mathsf{S}}(w, c) \leq \prod_{j=1}^m \delta(n_j)$ for some $c \in \mathrm{TC}^m$. For any $u \in [n_1]$, denote by $w_u = w(u, \cdot, \ldots, \cdot)$ and $c_u = c(u, \cdot, \ldots, \cdot)$ the restrictions of w, c to the u^{th} hyperplane "row" orthogonal to the first axis. We similarly use $w_u^{\langle i_1 \rangle}$ and $c_u^{\langle i_1 \rangle}$ to denote the restrictions of any $w^{\langle i_1 \rangle}$ and $c^{\langle i_1 \rangle}$.

Let $S = \{u \in [n_1] \mid \Delta_{\mathsf{S}}(w_u, c_u) \leq \prod_{j=2}^m \delta(n_j)\}$. By definition of suffix distance (Definition 3.7) and averaging argument (Markov), it holds that $\Pr_{u \leftarrow \sigma_{n_1}}[u \in S] \geq 1 - \delta(n_1)$.

By induction, for any $u \in S$ and $i_1 \in L_1$, the local corrector for TC^{m-1}, when invoked over $w_u^{\langle i_1 \rangle}$, returns the correct value from $c_u^{\langle i_1 \rangle}$ with probability at least $1 - E_{m-1}(n_2, \ldots, n_m)$. By union bound, except with probability $\left(q(n_1) \cdot \prod_j L_j\right) \cdot E_{m-1}(n_2, \ldots, n_m)$, any simulation of the local corrector for TC over a "column" $w^{\langle i_2, \ldots, i_m \rangle}(\cdot, t_2, \ldots, t_m)$ reads from an oracle that is equal to $c^{\langle i_2, \ldots, i_m \rangle}(\cdot, t_2, \ldots, t_m)$ at all coordinates in S and is, consequently, $\delta(n_1)$-close to it.

We may then invoke the soundness of the base corrector and deduce that the corrector for TC^m outputs $c(t_1, \ldots, t_m)$ with probability at least $1 - \left(q(n_1) \cdot \prod_j L_j\right) \cdot E_{m-1}(n_2, \ldots, n_m) - \epsilon$.

We conclude $E_m(n_1,\ldots,n_m) \le \left(q(n_1)\cdot\prod_j L_j\right)\cdot E_{m-1}(n_2,\ldots,n_m)+\epsilon \le m\cdot\left(\prod_j L_j^m q(n_j)\right)\cdot\epsilon$. □

Building on Lemma 4.11, we next show that the encoding algorithm for TC^m (Lemma 4.3) can be used for robust incremental encoding for the flattened code $\overline{\mathrm{TC}^m}$ w.r.t. $\overline{\Delta_{\mathsf{S}}}$ (defined in Proposition 4.9), given TC is locally correctable w.r.t. suffix distance.

Proposition 4.12 (Robust Incremental Encoding for $\overline{\mathrm{TC}^m}$)**.** *Let* TC *be a linear tree code that is* $(\Delta_{\mathsf{S}}, q, \delta)$*-LCC, where the codewords form a* (T, L)*-incremental ensemble. Then, the codewords of* $\overline{\mathrm{TC}^m}$ *form an* $(\overline{\Delta_{\mathsf{S}}}, \delta', T', L')$*-robustly incremental ensemble, where* $\delta'(n) = \delta(\lceil n^{1/m}\rceil)^m$, $T'(n) = L(\lceil n^{1/m}\rceil)^{3m}T(\lceil n^{1/m}\rceil)\cdot q(\lceil n^{1/m}\rceil)^m$ *and* $L' = L(\lceil n^{1/m}\rceil)^m$.

Proof. Let $w : [n-1] \to \Sigma$ be a corrupted codeword and let $c = \mathrm{TC}(x_1,\ldots,x_{n-1})$ be the closest codeword to w satisfying $\overline{\Delta_{\mathsf{S}}}(c, w) \le \delta^m$.

Recall, if $w = c$, computing the n^{th} symbol in $\mathrm{TC}(x_1,\ldots,x_n)$ given x_n could be done using the incremental encoding of TC^m over the coordinates of some rectangle (Proposition 4.7). For robustness against corruptions in w, we perform the same encoding yet, in order to read $w(t)$ for any $t \in [n-1]$, we use the local corrector of TC^m from Lemma 4.11 over w_I for an arbitrary rectangle $I \in \mathbf{Rect}(n-1)$, to retrieve the correct value $c(t)$.

Correctness, robustness and complexity of the encoding follows from the correctness and complexity of the incremental encoding of $\overline{\mathrm{TC}^m}$ (Proposition 4.7) and the correctness, soundness and complexity of the local corrector of TC^m (Lemma 4.11), which we apply over rectangles in $\mathbf{Rect}(n-1)$ which have dimension at most $\lceil n^{1/m}\rceil$ at any direction. □

4.2 Locally Correctable Tree Code

To make our locally testable code $\overline{\mathrm{TC}^m}$ have robust incremental encoding, we must instantiate it with a locally correctable base code (Proposition 4.12). In this section, we build such a code by combining (a variant of) a classic construction by Schulman [Sch94] with low-degree extensions.

We stress that, while local testability and robust encoding for $\overline{\mathrm{TC}^m}$ are implied generically whenever TC is a tree LCC, our choice of TC exhibits additional structural properties that are crucial to our tree PCP construction (in particular, to allow constraint evaluation under codewords (iii)), on which we elaborate in Section 5.

4.2.1 Schulman's Construction

We use a generalization of the tree code by Schulman [Sch94], which is among the first known tree code constructions and arguably the simplest. The construction is generic and uses linear block codes as black-box. We describe it formally in Fig. 6 and pictorially in Fig. 7. We additionally refer the reader to the exposition of the original construction in [Gel17, Section 3.1.1], where $d_k = 2$ for all k and $n_k = 2^k$.[7]

Proposition 4.13 ([Sch94, Gel17])**.** *Let* $\{\mathrm{C}_k : \mathbb{F}(0)^{n_k} \to \mathbb{F}(k)^{L_k n^k}\}$ *be a family of linear block codes where, for all* k, $\mathbb{F}(k-1) \subseteq \mathbb{F}(k)$, $n_k = d_k \cdot n_{k-1}$ *for* $d_k \in \mathbb{N}$*, and* C_k *has relative Hamming distance* δ_k*. Assume for simplicity that* δ_k *is non-increasing in* k *and that* d_k *and* L_k *are non-decreasing.*

Then, the encoding function TC *from Fig. 6 defines a linear tree code over* $\{\mathbb{F}(k)\}$ *with tree distance* $\delta_{k_{\max}}/2d_{k_{\max}+1}$ *and rate* $\Omega(1/(k_{\max}\cdot L_{k_{\max}}))$*, where* $k_{\max} = k_{\max}(n)$ *is the smallest* k *s.t.* $\sum_{i=1}^{k} n_k \ge n$.

[7] Besides allowing arbitrary d_k, we present a variant that is simpler than that from [Gel17] which still gives sufficiently good parameters.

A Tree Code TC from any Block Code $\mathrm{C} = \{\mathrm{C}_k : \mathbb{F}(k)^{n_k} \to \mathbb{F}(k)^{L_k n_k}\}$
$n_k = d_k \cdot n_{k-1}$ for $d_k \in \mathbb{N}$

$\mathrm{TC}(x_1, \ldots, x_n)$:

1. Let $k_{\max} := k_{\max}(n)$ be the smallest k such that $\sum_{i=1}^{k} n_i \geq n$.
   For $k = 1, \ldots, k_{\max}$ and $j = 1, \ldots, \lceil n/n_k \rceil - 1$, let
   $$c_{k,j} = \mathrm{C}_k(x_{(j-1)n_k+1}, \ldots, x_{jn_k}) \in \mathbb{F}(k)^{L_k n_k}.$$
2. Let
   $$c_k = (0^{L_k n_k}, c_{k,1}, \ldots, c_{k,\lceil n/n_k \rceil - 1}) \in \mathbb{F}(k)^{L_k n'(k)}, \tag{2}$$
   where $n'(k) = n_k \cdot \lceil n/n_k \rceil$ and view it as a function $c_k : [n'(k)] \to \mathbb{F}^{L_k}$ by dividing it into blocks of length L_k.
3. Output $c : [n] \to \mathbb{F}(k_{\max})^{1+k_{\max} L_{k_{\max}}}$, where
   $$c(t) = (x_t, c_1(t), \ldots, c_{k_{\max}}(t)). \tag{3}$$

Figure 6: The generic tree code construction by Schulman [Sch94].

Proof. The tree code is linear since it is simply a concatenation of codewords from linear block codes, and rate follows by construction.

We give a lower bound on the minimum tree weight of any codeword (its tree distance from the zeros codeword). Let $x \in \mathbb{F}^n$ be a non-zero message and let t be the location of the first non-zero symbol in x. Denote $c = \mathrm{TC}(x)$.

For any $k \in \{0, \ldots, k_{\max}\}$, let j_k denote the integer j satisfying $t \in \{(j-1)n_k + 1, \ldots, jn_k\}$. It holds that $c_{k,j_k} \in \mathrm{C}_k$ is a non-zero codeword since it encodes a message that contains $x(t)$ and, therefore, has relative Hamming weight at least $\delta_k \geq \delta_{k_{\max}}$. In the following, we show that these non-zero codewords cover sufficiently many locations in the suffix starting at t.

Let $S_k = \{j_k \cdot n_k + 1, \ldots, (j_k+1) \cdot n_k\}$ denote the locations in c where c_{k,j_k} resides (see illustration in Fig. 8). We construct a set $S \subseteq \{t, \ldots, n\}$ as follows:

1. Start with $S = \emptyset$ and $n^* = n$.
2. Let k^* be the largest k satisfying $(j_k + 1) \cdot n_k \leq n^*$.
3. Update $S \leftarrow S \cup S_{k^*}$ and $n^* = j_{k^*} n_{k^*}$, and repeat from Step 2.
4. If no such k^* exists, add t to S and finish.

We argue that $|S| \geq \left\lfloor \frac{1}{2d_{k_{\max}+1}} \cdot |\{t, \ldots, n\}| \right\rfloor$. This would complete the proof. By construction, all the sets S_{k^*} that are chosen to be added to S are disjoint, namely we never add the same location twice. Therefore, it suffices to shown that, at any iteration, $|S_{k^*}| \geq \left\lfloor \frac{1}{2d_{k^*+1}} \cdot |\{j_{k^*} n_{k^*} + 1, \ldots, n^*\}| \right\rfloor$

By the definition of j_k, $(j_{k+1} - 1)n_{k+1} < j_k n_k$ for any k since, otherwise, t belongs to two disjoint sets. By this and the maximality of k^*, we finish with

$$j_{k^*} n_{k^*} > j_{k^*+1} n_{k^*+1} - n_{k^*+1} > n^* - 2n_{k^*+1} = n^* - 2d_{k^*+1}|S_{k^*}|.$$

□

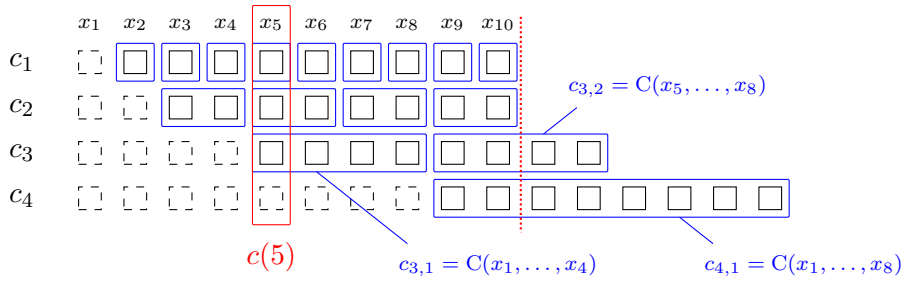

Figure 7: A codeword $c = \mathrm{TC}(x_1, \ldots, x_{10})$ of the tree code from Fig. 6 with $d_k = 2$ and length $n = 10$. The codeword consists of $k_{\max} = 4$ threads, where thread c_k, for $k = 1, \ldots, 4$, contains codewords in C that encode parts of the message of length $n_k = 2^{k-1}$. The codewords of C are marked in blue. A square at level k denotes a tuple of $L = L_k \leq L_{k_{\max}}$ field elements and dotted squares denote 0^L. The t^{th} symbol in c is a column consisting of at most $1 + L_{k_{\max}} k_{\max}$ field elements that includes x_t.

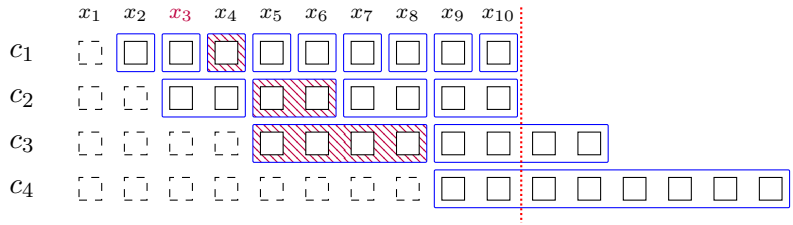

Figure 8: The effect of a non-zero at $t = 3$ in a codeword of length $n = 10$ with $d_k = 2$, defined by the coordinate sets $S_0, \ldots, S_{k^*}$ (striped).

When the block codes $\mathrm{C} = \{\mathrm{C}_k\}$ are linear and induce a well-defined encoding function, it is possible to efficiently verify that a word w is in the tree code TC and, if so, to decode it: The verification checks if each of the complete block-codewords composing w is in C and, importantly, that block-codewords from different threads that encode overlapping intervals in the message are indeed consistent with a unique message. Additionally, the verification checks if the partial block-codewords are consistent with this message.

We further generalize Schulman's construction to codes with no well-defined encoding function, as such codes appear in our tree PCP construction. Recall, $\mathrm{C}(x)$ and $\mathrm{TC}(x)$ denote, in this case, a *set of codewords* that can encode x under each of the respective codes.

Remark 4.14 (Schulman's Construction without Well-defined Encoding)**.** *We generalize the construction from Fig. 6 to the case where the underlying block codes* $\{\mathrm{C}_k\}$ *lack an encoding function, to still give a well-defined tree code* TC *(that in turn does not have a well-defined encoding function), as follows.*

The tree code TC *is defined as the set of all words of length* $n \in \mathbb{N}$ *where there exists* $(x_1, \ldots, x_n) \in \mathbb{F}(0)^n$ *such that* $c(t)$ *is as in Eq. (3) where, for* $k = 0, \ldots, k_{\max}$, c_k *is as in Eq. (2) where* $c_{k,j} \in \mathrm{C}_k(x_{(j-1)n_k+1}, \ldots, x_{jn_k})$ *for any* $j = 1, \ldots, \lfloor n/n_k \rfloor - 1$.

Note that in the generalization above, when n_k does not divide n, $c_{k,j}$ for $j = \lceil n/n_k \rceil - 1$ may be arbitrary. This is the last block-codeword in the thread c_k, that is not entirely contained in c. This is important to allow efficiently verifying that a given word w is in TC, where partial block-codewords can be ignored (as there is no generic way to efficiently verify partial codewords of C). At the same time, this does not harm the linearity of the tree code and neither its tree

distance: notice that the distance analysis in the proof of Proposition 4.13 never involves such partial block-codewords (see Fig. 8). Consequently, we obtain similar implication via an identical proof.

Proposition 4.15. *Proposition 4.13 holds with the same parameters also w.r.t. the generalized construction from Remark 4.14.*

4.2.2 Well-Structured Locally Correctable Block Codes

Schulman's tree code is not locally correctable in general. We instantiate the family of block codes $\{\mathrm{C}_k\}$ underlying the construction using low-degree extension codes (LDEs). It is a well-established fact that low-degree extension codes are locally correctable [GS92, Sud95]. Local correctability of $\{\mathrm{C}_k\}$, however, is not sufficient to imply local correctability of the tree code TC: The straightforward attempt to correct a symbol in some w using the local correction of the block codeword that contains it fails since that block codeword might be completely corrupted even when w is overall close to TC.

By carefully choosing parameters for the LDE codes and for the tree code construction, we are able to obtain a tree code where a symbol can be corrected by invoking the LDE local corrector not only over one block codeword but rather over all block codewords that span a suffix of w.

Definition 4.16 (The Block Code C)**.** *Let $0 < \mu < 1$ be any constant. Let $h_1 = 1$ and, for $k \geq 2$, let $h_k = \lceil \log(k)/\mu \rceil$. Denote $\mathbb{F}(k) = \mathrm{GF}(2^{h_k + \lceil \log k \rceil} + 3)$.*

Fix an infinite sequence of field elements $e_1, e_2, \ldots$ such that, for any $k \in \mathbb{N}$, $e_1, \ldots, e_{h_k} \in \mathbb{F}(k)$ are linearly independent over $\mathrm{GF}(2)$. For $k \in \mathbb{N}$, denote $\mathbb{H}(k) = \mathbf{Span}_{\mathrm{GF}(2)}(e_1, \ldots, e_{h_k}) \subset \mathbb{F}(k)$.[8]

We identify $[2^{h_k}]$ with $\mathbb{H}(k)$ as follows: any $t \in [2^{h_k}]$ maps to $\widetilde{t} = \sum_{i=1}^{h_k} b_i(t) \cdot e_i \in \mathbb{H}(k)$, where $b_i(t) \in \mathrm{GF}(2)$ is the i^{th} least significant bit in the binary representation of $t-1$. More generally, for message length $n_k = 2^{\sum_{i=1}^{k} h_i}$, we identify any $t \in [n_k]$ with $(\widetilde{t}_1, \ldots \widetilde{t}_k) \in \mathbb{H}(1) \times \cdots \times \mathbb{H}(k)$ by its mixed-radix representation over $[2^{h_1}] \times \cdots \times [2^{h_k}]$, in a little-endian order (i.e. least significant digit over $[2^{h_1}]$).[9]

Let $\{\mathrm{C}_k : \mathrm{GF}(2)^{n_k} \to \mathbb{F}(k)^{|\mathbb{F}(k)|^k}\}$, be the family of linear block codes where C_k is the low-degree extension defined as follows: For any $x : [n_k] \to \mathrm{GF}(2)$, the encoding $\mathrm{C}_k(x) = \widetilde{x}$ is the truth table of the unique k-variate polynomial that has degree at most $2^{h_i} - 1$ in its i^{th} input, and satisfies $\widetilde{x}(\widetilde{t}_1, \ldots, \widetilde{t}_{m'}) = x(t)$ for all $t \in [n_k]$.

To understand the parameters of the block codes from Definition 4.16, observe that $\log(n_k) = \sum_{i=1}^{k} h_i \geq \log(k!)/\mu = (1 - o(1)) \cdot k \log(k)/\mu$ and, therefore, $k < (1 + o(1)) \cdot \mu \log(n_k)/\log\log(n_k)$.

Hence, the code C_k encodes messages of length n_k over $\mathbb{F}(0) := \mathrm{GF}(2)$ using block length $|\mathbb{F}(k)|^k \leq (16k)^k \cdot 2^{k \cdot h_k} = n_k^{1+\mu+o(1)}$ over a field of size $|\mathbb{F}(k)| = \mathrm{polylog}(n_k)$, and has relative Hamming distance $\prod_{i=1}^{k}(1 - (\mathbb{H}(i))/|\mathbb{F}(k)|) \geq (1 - 1/8k)^k = \Omega(1)$ by the polynomial identity lemma (Schwartz-Zippel). Additionally, the encoding function C_k is efficiently computable.

The family $\{\mathrm{C}_k\}$ may be used to instantiate the tree code construction from Fig. 6 with $d_k = 2^{h_k} = O(\log(n_k)^{1/\mu})$, $L_k = n_k^{\mu+o(1)}$ and $\delta_k = \Omega(1)$.

Consequently, plugging in $\{\mathrm{C}_k\}$ in the generalized Schulman's construction (Fig. 6), we obtain the following as a corollary of Proposition 4.13. (Note the parameter $k_{\max}$ from the proposition satisfies $n \geq \sum_{i=1}^{k_{\max}-1} n_i > n_{k_{\max}-1}$ and, therefore, $k_{\max} - 1 < 2\mu \log(n)/\log\log(n)$ by the above.)

[8] Note that $\mathbb{H}(k)$ is an h_k-dimensional subspace of $\mathrm{GF}(2)^{h_k + \lceil \log k \rceil}$ but not a subfield of $\mathbb{F}(k)$.

[9] Specifically, $(\widetilde{t}_1, \ldots, \widetilde{t}_{m'})$ corresponds to $(t_1, \ldots, t_k) \in [2^{h_1}] \times \cdots \times [2^{h_k}]$, where $t = 1 + \sum_{i=1}^{k} 2^{h_0 + \cdots + h_{k-1}}(t_i - 1)$ (here, $h_0 = 0$).

Corollary 4.17 (The Tree Code TC)**.** *The linear tree code* TC*, obtained by instantiating the construction from Fig. 6 using* $\{\mathrm{C}_k\}$ *from Definition 4.16 with parameter* μ*, is over field of size* $\mathrm{polylog}(n)$*, has tree distance* $\Omega(1/\log(n)^{1/\mu})$ *and rate* $\Omega(1/n^{\mu+o(1)})$*. Additionally, encoding a message of length* n *takes time polynomial in* n*.*

Our low-degree extension codes are subcodes of corresponding Reed-Muller codes, which are known to be locally correctable w.r.t. relative Hamming distance Δ_{H} [GS92, Sud95].

Lemma 4.18. *For any* $\delta \leq 1/\log(n_k)^{1+1/\mu}$*, there exists* $\epsilon = 2^{-\Omega(\lambda)}$ *such that the family* $\{\mathrm{C}_k\}$ *from Definition 4.16 is* $(\Delta_{\mathsf{H}}, q, \delta, \epsilon)$*-LCC with* $q = \lambda \cdot \mathrm{polylog}(n_k)$*.*

We recall the following proposition from [RRR16, Proposition 3.8], which demonstrates a useful structural property of low-degree extensions, essentially allowing us to recover a symbol from a codeword of TC, that comes from a codeword of some C_k by reading symbols from different block codewords at different levels.

Proposition 4.19 (LDE Composition and Decomposition [RRR16])**.** *There exists:*

- *(Composition:) An efficient algorithm that takes as input* $k \in \mathbb{N}$ *and* $t \in \mathbb{F}(k)^k$ *and, for any* $(x_1, \ldots, x_{n_{k+1}}) \in \mathrm{GF}(2)^{n_{k+1}}$*, computes the* t^{th} *symbol in* $\mathrm{C}(x_1, \ldots, x_{n_{k+1}})$ *by reading one symbol from each of* $\mathrm{C}_k(x_{(j-1)n_k+1}, \ldots, x_{jn_k})$*, for* $j = 1, \ldots, d_{k+1}$*.*
- *(Decomposition:) An efficient algorithm that takes as input* $k \in \mathbb{N}$*,* $j \in [d_{k+1}]$ *and* $t \in \mathbb{F}(k)^k$ *and, for any* $(x_1, \ldots, x_{n_{k+1}}) \in \mathrm{GF}(2)^{n_{k+1}}$*, computes the* t^{th} *symbol in* $\mathrm{C}_k(x_{(j-1)n_k+1}, \ldots, x_{jn_k})$ *by reading one symbol from* $\mathrm{C}_{k+1}(x_1, \ldots, x_{n_{k+1}})$*.*

Note that the decomposition algorithm may be applied recursively to recover a symbol in a codeword $\mathrm{C}_k(x)$ be reading a symbol from a codeword of $\mathrm{C}_{k'}$, for any $k' > k$, that encodes an extension of x.

4.2.3 A Local Corrector for TC

Our local corrector for the tree code TC from Corollary 4.17 relies on the local correctability of the underlying codes $\{\mathrm{C}_k\}$ and the composition and decomposition algorithms from Proposition 4.19.

Lemma 4.20 (Local Correctability of TC)**.** *The tree code* TC *from Corollary 4.17 with parameter* μ *is* $(\Delta_{\mathsf{S}}, q, \delta)$*-LCC with* $q = O(n^{\mu+o(1)})$ *and* $\delta = \Omega(1/\log^2(n))$*.*

Proof. Let $w : [n] \to \mathbb{F}(n)^L$ denote the word given to the corrector as oracle (here, $L = O(n^{\mu+o(1)})$ is the size of the largest codeword symbol over $\mathbb{F}(n)$) and assume it is δ-close to $c \in \mathrm{TC}$ in suffix distance. For $1 \leq k \leq k_{\max}$ and $1 \leq j \leq \lceil n/n_k \rceil - 1$, we denote by $w_{k,j}$ and w_k the parts in w that are analogous to $c_{k,j}$ and c_k in the codeword c – see Fig. 6.

On input a coordinate $t \in [n]$, the corrector recovers $c(t) = (x_t, c_1(t), \ldots, c_{k_{\max}}(t))$ by recovering $c_k(t)$ for all k. For simplicity, we denote $c_0(t) = x_t$, $n_0 = 1$ and $d_1 = n_1$.

It holds that $c_k(t) = c_{k,j}(t - jn_k)$ for $j = \lceil t/n_k \rceil - 1$. We describe a recursive algorithm that recovers a field element in $c_{k,j}(t)$ for any $k \in \{0, \ldots, k_{\max}\}$, $j \in \{1, \ldots, \lceil n/n_k \rceil - 1\}$ and $t \in [n_k]$ (notice that $c_{k,j}(t)$ does not necessarily appear in c since the codeword may contain only a part of $c_{k,j}$). Recall that $c_{k,j}(t)$ consists of L_k elements, so the local corrector must apply the algorithm L_k times to recover each of them separately.

Recover a field element in $c_{k,j}(t) \in \mathbb{F}^{L_k}$:

1. If $j = 0$, output 0.
2. If $0 < j \leq \lfloor n/n_k \rfloor - 1$ (the codeword $c_{k,j}$ appears in c in its entirety):
   2.1. For $k' = k, \ldots, k_{\max}$, letting $j' = \lceil jn_k/n_{k'} \rceil$, until $k' = k_{\max}$ or $j' > \lfloor n/n_{k'} \rfloor - 1$ (i.e. $c_{k',j'}$ is partial in c):
   Recover the symbol in $c_{k,j}$ by applying the decomposition algorithm from Proposition 4.19 recursively $k' - k + 1$ times over $w_{k',j'}$ (notice that the string encoded by $c_{k,j}$ is a block of length n_k in the string supposedly encoded by $w_{k',j'}$). Instead of reading from $w_{k',j'}$ directly, however, use the local corrector for $\mathrm{C}_{k'}$ from Lemma 4.18 with $\lambda = \log^2(n)$.
   Denote by $y_{k'}$ the obtained value.
   2.2. Output the majority value of all $y_{k'}$.
3. If $j > \lfloor n/n_k \rfloor - 1$ (the codeword $c_{k,j}$ is partial in c):
   Recover the symbol in $c_{k,j}$ by applying the composition algorithm from Proposition 4.19 while reading one symbol from each of $c_{k-1,j'}$, for $j' = (d_k - 1) + 1, \ldots, jd_k$. Instead of reading directly from $c_{k-1,j'}$, however, recover their value recursively.

When $j = 0$, $c_{k,j}(t) = 0$ in any $c \in \mathrm{TC}$ by construction (Eq. (2)).

To see why the above corrector is successful in the second case ($0 < j \leq \lfloor n/n_k \rfloor - 1$), we follow a similar approach to the tree distance analysis in the proof of Proposition 4.13.

We show that the locations occupied by the $w_{k',j'}$ in w are a large fraction in a suffix of w starting at t. Since w is close to c, this means that most $w_{k',j'}$ are close to $c_{k',j'}$ and therefore applying the base corrector over these parts will return the correct value in the majority of cases.

Specifically, fix k and j and, for k', j' as in Step 2., let $S_{k'} = \{j' \cdot n_{k'} + 1, \ldots, (j'+1) \cdot n_{k'}\}$ denote the locations of $w_{k',j'}$ in w. Define S as follows:

1. Start with $S = \emptyset$ and $n^* = n$.
2. Let k^* be the largest $k' \geq k$ satisfying $(j_{k'} + 1) \cdot n_{k'} \leq n^*$.
3. Update $S \leftarrow S \cup S_{k^*}$ and $n^* = j_{k^*} n_{k^*}$, and repeat from Step 2.
4. If no such k^* exists, finish.

Observe that S is the union of all $S_{k'}$ such that $k' \geq k$ and $j' \leq \lfloor n/n_{k'} \rfloor - 1$ (where j' is defined as in Step 2.); this follows since any such k' satisfies $(j_{k'} + 1) \cdot n_{k'} \leq n$ and any two $S_{k'}$ sets are either disjoint or one is a subset of the other. Additionally, by the same reasoning as in the proof of Proposition 4.13, it holds that $|S| \geq \left\lfloor \frac{1}{2d_{k_{\max}+1}} \cdot (n - jn_k) \right\rfloor$, where, recall $d_{k_{\max}+1} = O(\log(n)^{1/\mu})$.

Hence, the blocks $\{w_{k',j'}\}$ over which we apply the local corrector for $\mathrm{C}_{k'}$ in Step 2. span over $\Omega(1/\log^{1/\mu}(n))$-fraction of some suffix of w. By Lemma 3.8, since $\Delta_{\mathsf{S}}(c, w) \leq \delta$, the relative Hamming distance between w and c over any suffix is at most $\delta' = H_n \cdot (\delta - o(1))$. By an averaging argument, then, at least 2/3-fraction of these blocks are in distance at most $O(\delta/\log^{(1/\mu)-1}(n))$ to their corresponding counterpart in c. By Lemma 4.18, this implies that for a sufficiently small $\delta = \Omega(1/\log^2(n))$, the base local corrector returns the correct value for the majority of k', with probability all but negligible.

In the third case, when $j > \lfloor n/n_k \rfloor - 1$, correctness follows immediately from the correctness of the composition of Proposition 4.19 and the correctness of recovery in the second case.

Let $q' = \mathrm{polylog}(n)$ be an upper bound on the query complexity of the local corrector for C, when applied over any block in w with parameter $\lambda = \log^2(n)$ (Lemma 4.18).

The recovery algorithm on input k enters Step 3. at most $k_{\max} = O(\log(n))$ in its recursion, since in each level of the recursion, at most one value of j' satisfies $j' > \lfloor n/n_k \rfloor - 1$). Each invocation of Step 3. incurs $d_k = O(\log(n)^{1/\mu})$ recursive calls. In total, then, Step 2. is performed at most polylog(n) times and, in each, the local corrector for C is called $k' - k + 1 \leq k_{\max} = O(\log(n))$ times. Overall, the query complexity of the recovery algorithm is polylogarithmic in n and, to recover the $k \cdot L_{k_{\max}}$ field elements in $c(t)$, the local corrector for TC reads $O(k_{\max} \cdot L_{k_{\max}} \cdot \text{polylog}(n)) = O(n^{\mu + o(1)})$ locations from w.

The runtime efficiency of the recovery follows from the efficiency of the composition and decomposition algorithms of Proposition 4.19. □

The local correctability of TC w.r.t. suffix distance (Lemma 4.20) implies its local correctability w.r.t. tree distance (Theorem 1.7) by Lemma 3.8.

As a corollary of Lemma 4.20 and Propositions 4.9 and 4.12, we conclude that the flattened code $\overline{\text{TC}^m}$, where TC is the tree code from Corollary 4.17 is both locally testable and has an incremental online encoding function that is robust against corruptions.

Corollary 4.21 (Locally Testable Tree Code with Robust Encoding)**.** *There exists a constant γ such that, for any $\mu > 0$ and $m \in \mathbb{N}$, the flattened code $\overline{\text{TC}^m}$ based on* TC *from Corollary 4.17 with parameter μ, is $(\overline{\Delta_{\mathsf{S}}}, q, \epsilon)$-locally testable with $q = O(\log(n)^{(2+1/\mu)m+2} \cdot n^{2/m})$ and $\epsilon(n) = \Omega(1/\log^m(n))$ and admits a $(\overline{\Delta_{\mathsf{S}}}, \delta, T, L)$-robustly incremental ensemble with $\delta(n) = \Omega(1/\log^{2m}(n))$, $T(n) = n^{4\mu + \gamma/m + o(1)}$ and $L(n) = n^{\mu + o(1)}$.*

In particular, the code $\overline{\text{TC}^m}$ admits a local corrector w.r.t. $\overline{\Delta_{\mathsf{S}}}$, and therefore w.r.t. tree distance (Proposition 4.10), that underlies the above incremental encoding, and is obtained by Lemma 4.11.

5 Constraint Evaluation under Codewords

We express the transition and consistency constraints over the CktReach witness, i.e. the assignments $A^1, \ldots, A^{3s}$ from Fig. 1, using a collection of well-structured constraints $\{P\}$ that satisfy:

1. (Correctness, (i)) Any assignments $A^1, \ldots, A^{3s}$ satisfy the CktReach constraints if and only if there exist witnesses $W^1, \ldots, W^K$ such that $P(A^1, \ldots, A^{3s}, W^1, \ldots, W^K) \equiv 0$ for all P.

In the above, we associate zero with TRUTH and this will be the case from now on. The output of P is a vector, which roughly corresponds to the truth table of the constraint (transition or consistency) when evaluated at all time coordinates $t = 1, \ldots, n$.

2. (Codeword Evaluation, (iii)) The structure of $\{P\}$ allows the verifier to evaluate them over any $A^1, \ldots, A^{3s}$ and $W^1, \ldots, W^K$ "underneath" their corresponding codewords. By this, we mean that given access to codewords $\widetilde{A}^1, \ldots, \widetilde{A}^{3s}, \widetilde{W}^1, \ldots, \widetilde{W}^K$ encoding the assignments and their witnesses, the verifier can simulate access to a codeword $\widetilde{E}$ encoding the evaluation vector
$$E = P(A^1, \ldots, A^{3s}, W^1, \ldots, W^K).$$
Jumping ahead, simulating access to $\widetilde{E}$, a redundant encoding of E, facilitates performing a zero test (Section 6) for checking that E is the all-zero string.

3. (Incrementality) For applicability to tree PCPs, we require that the witness and evaluation vectors, W^i and E, are incremental in the assignments $A = (A^1, \ldots, A^{3s})$ (Definition 3.2). Namely, that witnesses and evaluations corresponding to A of length n are an extension of the witnesses and evaluations corresponding to any prefix of A. Additionally, extending these vectors upon appending new values to the assignments can be done efficiently.

Notation. The statements made in this section are with respect to the tree code TC from Corollary 4.17 with parameter $0 < \mu < 1$, which is the instantiation of Fig. 6 with the block code C from Definition 4.16, and its derivations TC^m and $\overline{\mathrm{TC}^m}$ (Corollary 4.21). We use $\{\mathbb{F}(k)\}$ to refer to the finite fields from Definition 4.16.

5.1 The Transition Constraints

Evaluating the transition constraints (Fig. 1, Step 2.2) entails evaluating a given circuit C over assignments $A^1, \ldots, A^S$ in a pointwise manner .

Lemma 5.1 (Pointwise Circuit-Evaluation under Codewords)**.** *For any circuit $C : \{0,1\}^S \to \{0,1\}$, there exists a collection of constraints $\mathbf{Eval}(C)$ of size $\mathrm{poly}(|C|)$, where every constraint $P \in \mathbf{Eval}(C)$ is a function $P = \{P_n : (\mathbb{F}^n)^{S+K} \to \mathbb{F}^n\}$, for $\mathbb{F} = \mathbb{F}(0)$ and $K = \mathrm{poly}(|C|)$, that satisfies the following properties:*

- *(Correctness) The following two conditions are equivalent for any $A^1, \ldots, A^S : [n] \to \mathbb{F}$:*
  - *$A^1, \ldots, A^S$ are binary and $C(A^1(t), \ldots, A^S(t)) = 1$ for all $1 \le t \le n$.*
  - *There exist $W^1, \ldots, W^K : [n] \to \mathbb{F}$ such that*

    $$P(A^1, \ldots, A^S, W^1, \ldots, W^K) \equiv 0$$

    for all $P \in \mathbf{Eval}(C)$.

  We say that such $W^1, \ldots, W^K$ are witnesses *to $A^1, \ldots, A^S$.*
- *(Codeword Evaluation) There exists a linear tree code TC' with tree distance $\Omega(1/\log(n)^{1/\mu})$ (and no explicit encoding function) and a $O(n^{\mu+o(1)}) \cdot \mathrm{poly}(|C|)$-time deterministic algorithm $\mathcal{A}$ that satisfy:*

  For any $A^1, \ldots, A^S, W^1, \ldots, W^K : [n] \to \mathbb{F}$, there exists $\widetilde{E} \in \overline{(\mathrm{TC}')^m}(P(A^1, \ldots, A^S, W^1, \ldots, W^K))$ such that on any input $t \in [n]$, $\mathcal{A}$ computes $\widetilde{E}(t)$ by reading at most one location from each of $\{\widetilde{A^i} = \overline{\mathrm{TC}^m}(A^i)\}$ and $\{\widetilde{W^i} = \overline{\mathrm{TC}^m}(W^i)\}$.
- *(Incrementality) For any satisfying $A = (A^1, \ldots, A^S)$, there exist witnesses $W^1 = W^1_A, \ldots, W^K = W^K_A$ such that $\{W^i_A\}$, for any i, and $\{E_A = P(A^1, \ldots, A^S, W^1_A, \ldots, W^K_A)\}$, for any $P \in \mathbf{Eval}(C)$, are $(\mathrm{poly}(|C|), 1)$-incremental.*

To prove the lemma, we first turn the constraint $C(A^1, \ldots, A^S) \equiv 1$ into a collection of low-degree constraints over the finite field $\mathbb{F} := \mathbb{F}(0)$ by standard techniques. Then, we show how to evaluate low-degree polynomials over messages underlying codewords of $\overline{\mathrm{TC}^m}$.

The first step is to convert the circuit-satisfiability statement over C into a 3-SAT statement over a 3-CNF formula ψ_C over $S + K$ variables with L clauses, where $L, K = \mathrm{poly}(|C|)$ and ψ_C is satisfiable with input $A^1, \ldots, A^S$, $W^1, \ldots, W^K$ for some witness $W^1, \ldots, W^K$ if and only if C is satisfiable by $A^1, \ldots, A^S$. Then, to allow codeword evaluation, we arithmetize ψ_C and express it as a low-degree polynomial over $\mathbb{F}$: We associate each Boolean value in the assignment with the corresponding field element in $\{0,1\} \subseteq \mathbb{F}$ and obtain an assignment over $\mathbb{F}$. We represent each clause in the 3-CNF formula ψ_C by the degree-3 polynomial that agrees with it over $\{0,1\}^3$, where we associate a TRUE outcome with 0 and FALSE with 1. (For instance, $(\neg x_1 \vee \neg x_2 \vee x_3)$ is represented by the polynomial $x_1 x_2 (1 - x_3)$.) The formula ψ_C is then represented by the collection of the degree-3 polynomials corresponding to its clauses.

Proposition 5.2 (3-CNF Arithmetization)**.** *There exists a mapping from any 3-CNF formula ψ over K' variables with L clauses to a collection $\{p_1, \ldots, p_L\}$ of degree-3 K'-variate polynomials over $\mathbb{F}$, such that a Boolean assignment satisfies ψ if and only if its embedding in $\mathbb{F}^{K'}$ satisfies is a root of p_j for all j.*

In addition to the polynomials $p_1, \ldots, p_L$, we must check that $A^1, \ldots, A^S$ and $W^1, \ldots, W^K$ indeed encode Boolean assignments. To that end, we add the polynomials $z_1, \ldots, z_{S+K}$ to our collection of constraints, where

$$z_i(x_1, \ldots, x_{S+K}) = x_i(1 - x_i). \tag{4}$$

Evidently, the set roots of z_i is exactly all inputs where $x_i \in \{0, 1\}$.

Let us define $\mathbf{Eval}(C) = \{P_f \mid f \in \{p_1, \ldots, p_L, z_1, \ldots, z_{S+K}\}\}$, where $P_f(A^1, \ldots, A^S, W^1, \ldots, W^K)$ is the function that evaluates $f(A^1(t), \ldots, A^S(t), W^1(t), \ldots, W^K(t))$ at any $t \in [n]$. By the above, there exist $W^1, \ldots, W^S : [n] \to \mathbb{F}$ such that $P(A^1, \ldots, A^S, W^1, \ldots, W^K) \equiv 0$ for all $P \in \mathbf{Eval}(C)$ if and only if $A^1, \ldots, A^K$ are binary and $C(A^1(t), \ldots, A^K(t)) = 1$ for all t. Since any p_i or z_i is a polynomial over $\mathbb{F}$ of degree at most 3, it remains to show how to evaluate degree-3 polynomials, in a point-wise manner, underneath TC^m codewords.

In linear codes, by definition, adding the encoding of two messages gives the encoding of their sum. Thus, the only missing part for locally evaluating low-degree polynomials over codewords is to be able to *multiply* the variables they encode. Since our goal is to evaluate polynomials over variables coming from the same coordinate in different codewords, we are specifically interested in the point-wise product of two encoded messages. With a similar goal in mind, Meir [Mei13] formulates the notion of *multiplication codes* that precisely captures this capability. We restrict the definition to a simple special case of the notion defined in [Mei13], that is sufficient and achievable in our context.[10]

For simplicity, we restrict our definition to codes that have a well-defined encoding function. The corresponding multiplication code, however, may lack such an encoding function (in which case, recall, $\mathrm{C}(x)$ denotes a set of codewords).

Definition 5.3 (Multiplication Codes [Mei13])**.** *We say that a block code C over $\mathbb{F}$ with a well-defined encoding function is a M-*multiplication code*, for an integer $M \in \mathbb{N}$, if there exists another code C', which we refer to as the product code of C, such that*

$$\mathrm{C}(x_1) \odot \ldots \odot \mathrm{C}(x_M) \in \mathrm{C}'(x_1 \odot \ldots \odot x_M),$$

where $\odot$ denotes point-wise product over $\mathbb{F}$.

We extend the notion to linear tree codes $\mathrm{TC} = \{\mathrm{TC}_n : \mathbb{F}(0)^n \to \mathbb{F}(n)^{L(n)}\}$ where multiplication over $\mathbb{F}(n)^L$ is defined as the point-wise product.

Proposition 5.4 (C is a Multiplication Code)**.** *For any k, the block code C_k from Definition 4.16 is a 5-multiplication code. The product code of C_k is linear and has constant relative Hamming distance.*

Proof. The code is 5-multiplication by the fact that the point-wise multiplication of the evaluation table of two polynomials gives the evaluation table of their product. The product code has relative distance $\prod_{i=1}^{k}(1 - 5\mathbb{H}(i)/|\mathbb{F}(k)|) \geq (1 - 5/8k)^k = \Omega(1)$ since the individual degree blows up by at most 5 in the product polynomial. □

[10] We remark that similar notions existed in the literature, but our use here is closest to that of [Mei13].

Note that the multiplication code of C_k, denote it by C'_k, has the capacity to encode information of length $5n_k$. On the other hand, codewords that are the point-wise product of two codewords in C_k encode information of length n_k.

For instance, let $x_1, y_1 \in \mathrm{GF}(2)^{n_k}$ and $x_2, y_2 \in \mathbb{F}(2)^{n_k}$ denote two message pairs such that $x_1 \odot y_1 = x_2 \odot y_2 = z$. Then, $c_1 = \mathrm{C}_k(x_1) \odot \mathrm{C}_k(y_1)$ and $c_2 = \mathrm{C}_k(x_2) \odot \mathrm{C}_k(y_2)$ are both codewords in C'_k that encode z – they are the evaluation vectors of degree-$5n_k$ polynomials that evaluate z over $\mathbb{H}(1) \times \cdots \times \mathbb{H}(k)$ (see Definition 4.16) – but are not necessarily identical over all of $\mathbb{F}(k)$.

The extra redundancy makes the encoding function over the message space ambiguous but this is completely fine since C'_k is still a well-defined linear code (as a subset of strings) with constant relative Hamming distance. Further, every codeword $c \in \mathrm{C}'_k$ encodes a well-defined message in $\mathbb{F}(k)^{2^k}$ that can be efficiently decoded given c.

Remark 5.5. *The multiplication code C'_k, corresponding to C_k, satisfies $|\mathrm{C}'_k| > |\mathrm{C}_k|$ and does not induce a well-defined encoding function over the message space $\mathrm{GF}(0)^{n_k}$. In particular, any $x \in \mathrm{GF}(2)^{n_k}$ has many valid encodings under C'_k, which we denote by the set $\mathrm{C}'_k(x)$.*

Nevertheless, given a word $w \in \mathbb{F}(k)^{\mathbb{F}(k)^k}$ as input, it is possible to efficiently tell if $w \in \mathrm{C}'_k$ and, if that is the case, to find the well-defined message x such that $w \in \mathrm{C}'_k(x)$.

We note that 3-multiplication suffices for the proof of Lemma 5.1. However, a larger multiplication degree of 5 is required for evaluating the consistency constraints in Section 5.2.

The tree code TC from Fig. 6 generically inherits the multiplication property of the underlying block code C, if it exhibits any.

Proposition 5.6. *The tree code* TC *from Fig. 6 is M-multiplication assuming the underlying block codes $\{\mathrm{C}_k\}$ are M-multiplication. Further, instantiating the construction with the product code of* C *gives the product code of* TC *(Remark 4.14).*

Proof. The proposition follows from the observation that a codeword in TC is a concatenation of codewords from C that encode fixed sections of the input. Then, the multiplication property follows from the fact the, for any $x, y \in \mathbb{F}(0)^n$ and any (possibly overlapping) $I, J \subseteq [n]$,

$$\begin{aligned}&\big(\mathrm{C}_1(x_I), \mathrm{C}_2(x_J)\big) \odot \big(\mathrm{C}_1(y_I), \mathrm{C}_2(y_J)\big)\\ &= \big(\mathrm{C}_1(x_I) \odot \mathrm{C}_1(y_I), \mathrm{C}_1(x_J) \odot \mathrm{C}_2(y_J)\big) \in \{(c_1, c_2) \mid c_1 \in \mathrm{C}'_1\big((x \odot y)_I\big),\ c_2 \in \mathrm{C}'_2\big((x \odot y)_J\big)\}.\end{aligned}$$

□

Following Remark 5.5, the product code of TC does not induce a well-defined encoding function over the message space. However, it is still a well-defined tree code with tree distance $\Omega(1/\log^{1/\mu}(n))$, and allows to verify and decode codewords efficiently – see Remark 4.14 and Proposition 4.15 and the discussion in between.

As noted by [Mei13, Proposition 3.13], the tensor of a multiplication code is itself a multiplication code, where the product code is the tensor of the base product code. Thus, TC^m is a multiplication code if TC is. We additionally note that flattening the code TC^m preserves its multiplication property since we are merely re-organizing the codeword symbols. Overall, we obtain the following, which completes the proof of Lemma 5.1.

Proposition 5.7. *If a tree code* TC *is a M-multiplication code with product code* TC'*, then* TC^m *and* $\overline{\mathrm{TC}^m}$ *are M-multiplication codes with product codes* $(\mathrm{TC}')^m$ *and, respectively,* $\overline{(\mathrm{TC}')^m}$*.*

5.2 The Consistency Constraints

Next, we show how to evaluate the consistency constraints over two encoded assignments $A = A^i$ and $A' = A^j$ (Fig. 1, Step 2.3), akin to the evaluation of the transition constraints from above.

While the evaluation of the transition constraints given in Lemma 5.1 is quite straight-forward, we obtain weaker, more nuanced, guarantees from the evaluation of consistency constraints in Lemma 5.8 below, which are nevertheless still sufficient for a tree PCP. Before stating the lemma, let us highlight the differences compared to the statement from Lemma 5.1:

1. Unlike the witnesses for $\mathbf{Eval}(C)$, the length of witnesses W^i for the consistency constraints and the corresponding evaluation vectors E does not exactly match the length of the assignments n, but is still $N = O(n)$. While W^i and E are still monotone in the assignments A, A', they are not incremental in the sense of Definition 3.2 – it is no longer the case that extending A, A' by one additional coordinate requires adding few values to W^i and E. However, any new symbol in W^i and E can be computed by reading a few locations in the assignments and, therefore, we obtain an amortized notion of incrementality. In Section 7.3, we show how to de-amortize W^i and E to make them incremental, on par with the tree PCP requirements.

2. It does not hold here that an evaluation vector E is all-zeros when the corresponding constraint is satisfied, but rather that it is zeros over a certain subset of coordinates R. It is convenient to switch to a setting where we view E as an m-dimensional tensor over some rectangle $I = [n_1] \times \cdots \times [n_m]$, where the subset R is a sub-rectangle $R \subseteq I$. Consequently, every constraint P is associated with such an R, and we require $E_R \equiv 0$. Importantly, our zero test (Section 6) supports verifying such partial statements, as long as the "zero set" R is indeed a rectangle.

3. We do not cover all consistency constraints over a pair of assignments A, A', namely that $A(t) = A'(t-1)$ for all coordinates $t > 1$ (2.3). Instead, we guarantee that $A(t) = A'(t-1)$ *for almost all* t. Specifically, for all t except for an efficiently computable, constant-size set of coordinates which we denote by $\mathbf{Bad}(n)$. Since it has constant size, this exception does not constitute a big issue in our PCP construction: The verifier can individually verify each of the remaining consistency constraints in a straight-forward way using the local correctability of the code.

4. To simulate access to an evaluation codeword $\widetilde{E}$, we require access to an "extended" encoding of the witnesses $\{W^i\}$ under a flattened-tensor tree code $\overline{(\mathrm{TC}^+)^m}$, where TC^+ is an extension of TC. Importantly, TC^+ is also locally correctable, making the encodings robustly incremental.

Lemma 5.8 (Consistency Evaluation under Codewords)**.** *There exists an (infinite) collection of constraints* $\mathbf{EQ}$*, where every constraint* $(P, R) \in \mathbf{EQ}$ *consists of a function* $P = \{P_n : (\mathbb{F}^n)^2 \times \mathbb{F}^{N_1} \times \cdots \times \mathbb{F}^{N_K} \to \mathbb{F}^{I_P}\}$*, for* $\mathbb{F} = \mathbb{F}(0)$*,* $K := K(n) = \mathrm{polylog}(n)$*,* $N_i := N_i(n) = O(n)$ *and a rectangle* $I_P := I_P(n) \subset \mathbb{N}^m$ *of size* $O(n)$*, and* R *is a (possibly infinite) rectangle* $R \subseteq \mathbb{N}^m$*, that satisfies the following properties:*

- *(Size) For any* $n \in \mathbb{N}$*, the set* $\mathbf{EQ}(n) = \{(P, R) \in \mathbf{EQ} \mid I_P(n) \cap R \neq \emptyset\}$ *has size* $\mathrm{polylog}(n)$*.*
- *(Correctness) There exists an efficiently computable* $\mathbf{Bad}(n) \subseteq [n]$ *of size* $O(1)$*, such that the following two conditions are equivalent for any* $A, A' : [n] \to \mathbb{F}$*:*
  - A, A' *are binary and* $A(t) = A'(t-1)$ *for all* $1 < t \leq n$*,* $t \notin \mathbf{Bad}(n)$*.*

- *There exist $W^1, \ldots, W^K$ (henceforth witnesses for A, A') where $W^i : [N_i] \to \mathbb{F}$, such that for all $(P, R) \in \mathbf{EQ}$, letting $E = P(A, A', W^1, \ldots, W^K) \in \mathbb{F}^{I_P(n)}$, it holds that $E_R \equiv 0$.*

– *(Codeword Evaluation) There exists a linear tree code TC' (without explicit encoding) and a $(\Delta_{\mathsf{S}}, n^{\mu+o(1)}, \Omega(1/\log^{2m}(n)))$-locally correctable linear tree code TC^+, both with tree distance $\Omega(1/\log(n)^{1/\mu})$, and an $O(n^{\mu+o(1)})$-time deterministic algorithm $\mathcal{A}$ that satisfy:*

For any $A, A', W^1, \ldots, W^K$, there exists $\widetilde{E} \in (\mathrm{TC}')^m(P(A, \ldots, A', W^1, \ldots, W^K))$ such that on any input $(t_1, \ldots, t_m)$, $\mathcal{A}$ computes $\widetilde{E}(t_1, \ldots, t_m)$ by reading at most one location from each of $\{\widetilde{A}^i = \overline{\mathrm{TC}^m}(A^i)\}$ and $\{\widetilde{W}^i = \overline{(\mathrm{TC}^+)^m}(W^i)\}$.

– *(Amortized Incrementality) For any satisfying A, A', there exist witnesses $W^1 = W^1_A, \ldots, W^K = W^K_A$ (that depend only on A) such that $\{W^i_A\}$, for any i, and $\{E_{A,A'} = P(A, A', W^1_A, \ldots, W^K_A)\}$, for any $(P, R) \in \mathbf{EQ}$, are monotone ensembles. Additionally, any symbol in W^i_A or $E_{A,A'}$ can be efficiently computed by making $O(1)$ queries to A, A'.*

Arithmetization of equality between two variables x_i, x_j in $\{0, 1\} \subseteq \mathbb{F}$ can be obtained by the simple degree-2 function $\mathrm{EQ}(x_i, x_j) = (x_i - x_j)^2$. However, in contrast to the 3-CNF evaluations that we apply over variables coming from the same location in different encoded assignments (i.e. the same "row" in the columns $A^1, \ldots, A^S$), the consistency constraints involve a variable from any location t of some encoded assignment and a variable coming from location $t-1$ of another assignment. We can thus check consistency by evaluating EQ, in a point-wise manner, over some assignment column $A = A^i$ and a shift of another $A' = A^j$. Given the tools from the previous section, the remaining challenge lays in applying the shift. Roughly speaking, our goal then is, given an encoding of an assignment, to simulate access to the encoding of its shift.

We do not know if there exists a tree code that allows shifting a location t in the encoded message to location $t-1$. Instead, we show that our tree code construction (Corollary 4.17) allows for a different type of shifts, which are sufficient to simulate $t \mapsto t-1$ for all values of t using a small number of operations.

To describe the shift functions we realize, it is convenient to represent the coordinate set $[n]$ using $\lceil \log n \rceil$-bit labels, where each $t \in [n]$ is represented by the binary representation of $t-1$, which we denote by $\mathrm{b}(t)$ (e.g. $\mathrm{b}(1) = 0^{\lceil \log n \rceil}$). We look into shifts defined by the functions $\{\Gamma_i : \mathbb{N} \to \mathbb{N}\}$, where for all $i \in \mathbb{N}$, $\Gamma_i(t)$ is the coordinate with the label obtained by flipping the i least significant bits in the label of t. That is, $\Gamma_i(t) = \mathrm{b}^{-1}(\mathrm{b}(t) \oplus (0, \ldots, 0, 1^i))$. For every $i \in \mathbb{N}$, let Λ_i be the set of all $t \in \mathbb{N}$ where i is the biggest integer such that 2^{i-1} divides $t-1$ (in other words, if the i least significant bits in the label of t are 10^{i-1}). Observe that if $t \in \Lambda_i$, then $\Gamma_i(t) = t-1$ (see Fig. 3 for illustration). Since $\Lambda_1, \ldots, \Lambda_{\lceil \log n \rceil}$ cover $[n]$, we may write

$$\forall t, \quad \mathrm{EQ}(A(t), A'(t-1)) = 0 \iff \forall i \in \lceil \log n \rceil\,,\ t \in \Lambda_i, \quad \mathrm{EQ}(A(t), A'(\Gamma_i(t))) = 0 \tag{5}$$

for any assignments $A, A' : [n] \to \{0, 1\}$.

For technical convenience, we additionally extend the above definitions to $i = 0$, where Γ_0 is the identity function and $\Lambda_0 = \mathbb{N}$.

5.2.1 Shifting under Codewords

Having reduced our goal to dealing with shifts by the functions Γ_i, we next show that the tree code $\overline{\mathrm{TC}^m}$ indeed allows us to locally simulate access to codewords of shifted messages. For this to be possible, we require that the block code underlying the base tree code construction (C from Definition 4.16) exhibits such a property. Prior works on PCP use low-degree extensions to attain

similar structural properties that are useful for arithmetization (so-called affine invariance). Some even specifically consider the type of shifts we are interested in [Spi95, PS94, BGH+06].

Proposition 5.9 (Shifting under C)**.** *There exists a deterministic algorithm $\mathcal{A}$ that for any $x : [n_k] \to \mathbb{F}(k)$ and any input $i \in [k]$, simulates a query to $\hat{c} = \mathrm{C}(x \circ \Gamma_i)$ by making a single query to $c = \mathrm{C}(x)$.*

Proof. Recall every coordinate $t \in [n_k]$ is represented by $(\widetilde{t}_1, \ldots, \widetilde{t}_k) \in \mathbb{H}(1) \times \cdots \times \mathbb{H}(k)$ in the encoding under C_k (Definition 4.16). For any i, the transformation from $(\widetilde{t}_1, \ldots, \widetilde{t}_k)$ to the representation of $\Gamma_i(t)$ in $\mathbb{H}(1) \times \cdots \times \mathbb{H}(k)$ is affine over $\mathbb{H}(1) \times \cdots \times \mathbb{H}(k)$ and, therefore, over $\mathbb{F}(k)^k$. Hence, the codeword encoding $x \circ \Gamma_i(\cdot)$ is a low-degree extension that can be rewritten as $\widetilde{x}(\alpha \cdot (\widetilde{t}_1, \ldots, \widetilde{t}_k) + \beta)$ for some $\alpha, \beta \in \mathbb{F}(k)^k$. □

Unlike the multiplication property that seamlessly propagates from the underlying block code C all the way to the flattened-tensor code $\overline{\mathrm{TC}^m}$, extending Proposition 5.9 to $\overline{\mathrm{TC}^m}$ requires a much more subtle treatment. For start, we show that given codewords of C can be "shifted" by Γ_i as implied by the proposition, then codewords of the base tree code TC can be similarly "shifted" albeit with some extra "help". In more details, recall that a codeword $c \in \mathrm{TC}$ is the "vertical" concatenation of $k_{\max} = O(\log(n))$ "threads" c_k (see Figs. 6 and 7). Every c_k is itself a "horizontal" concatenation of $\lceil n/n_k \rceil - 1$ codewords of C, of total length $n'(k) = n_k \cdot \lceil n/n_k \rceil$ over $\mathbb{F}^\ell$, which possibly exceeds n (note the dotted red line in Fig. 7). When $n'(k) > n$, the codeword c does not contain all of c_k. To locally simulate the shifts, however, we require access to any c_k in its entirety and not only the parts composing c. (This will not be a problem for us since any c_k will eventually be a part of the growing codeword at time at most $d_{k_{\max}} \cdot n = O(n \log^{1/\mu}(n))$; it will merely require the PCP to include parts from future codeword symbols in advance.)

It is convenient to view such an extension of c as coming from a tree code that extends TC.

Definition 5.10 (The Code TC^+)**.** *We define TC^+ to be the linear code that maps any x to the concatenation of x with $\{c_k \mid 1 \le k \le k_{\max}\}$, where c_k and $k_{\max}$ are as defined in Fig. 6 w.r.t. the tree code* TC *from Corollary 4.17.*

TC^+ maps any $x \in \mathrm{GF}(2)^n$ to $\mathrm{TC}^+(x) \in \mathbb{F}(k_{\max})^{n^+}$, where $\mathbb{F} = \mathbb{F}(k_{\max})$ is of size $\mathrm{polylog}(n)$ and $n^+ = k_{\max} \cdot L_{k_{\max}} \cdot n'(k_{\max}) = O(n^{1+1/\mu+o(1)})$. While TC^+ satisfies the syntax of tree code (Definition 3.3) with some choice of alphabet, for simplicity we diverge from that syntax and do not split an encoding of length-n message into n codeword symbols. Rather, we think of codewords in TC^+_n as being over the alphabet of $\mathrm{TC}_{n'}$, where $n' = n'(k_{\max}) = O(n \log^{1/\mu}(n))$. (Formally speaking, one can view TC^+ as an infinite collection of block codes.) Note that TC^+_n is a restriction of $\mathrm{TC}_{n'}$ as a vector space of $\mathbb{F}(k_{\max})$, and we pad codeword symbols with zero field elements where necessary. With this view of TC^+, it follows that TC^+_n immediately inherits the minimum tree distance of $\mathrm{TC}_{n'}$. It also inherits local correctability by naively adjusting the correction algorithm from Lemma 4.20. Consequently, $\overline{(\mathrm{TC}^+)^m}$ exhibits a robust online encoding following Proposition 4.12.

Lemma 5.11. *The code $\mathrm{TC}^+ = \{\mathrm{TC}^+_n\}$ has tree distance $\Omega(1/\log^{1/\mu}(n))$ and is $(\Delta_{\mathsf{S}}, q, \delta)$-LCC with $q = n^{\mu+o(1)}$ and $\delta = \Omega(1/\log^2(n))$. Consequently, there exists a constant γ such that for any $m \in \mathbb{N}$, the codewords of $\overline{(\mathrm{TC}^+)^m}$ form a $(\overline{\Delta_{\mathsf{S}}}, \delta^m, T, L)$-robustly incremental ensemble with $T(n) = n^{4\mu+\gamma/m+o(1)}$ and $L(n) = n^{\mu+o(1)}$.*

A range of coordinates $[n]$ is closed under Γ_i only when n is divisible by 2^i. Consequently, we are able to simulate access to encoded assignments shifted by Γ_i only when their length is such an n. This suffices for our eventual goal, as we shall see in the next section.

Lemma 5.12 (Shifting under TC^+)**.** *There exists a deterministic algorithm $\mathcal{A}$ that takes as input an integer $i \geq 0$ and coordinate t and, for any $x : [n] \to \mathbb{F}$ s.t. n is divisible by 2^i, outputs the t^{th} field element in $\hat{c} = \mathrm{TC}^+(x \circ \Gamma_i)$ by reading one field element from $c = \mathrm{TC}^+(x)$.*

Proof. Let $\hat{c}_k$ be the component of $\hat{c}$ as defined in Fig. 6. It suffices to show how to compute a field element in $\hat{c}_k(t)$ for all $k = 0, \ldots, k_{\max}$. (Computing the "message part", i.e. the first field element in any $\hat{c}(t)$ equal to $x_{\Gamma_i(t)}$, is straight-forward). Assume $t > n_k$ (otherwise $\hat{c}_k(t) = 0^\ell$). Letting $j = \lceil t/n_k \rceil - 1$ and $t' = t \mod n_k$, it holds that $\hat{c}_k(t) = \hat{c}_{k,j}(t')$ where $\hat{c}_{k,j} \in \mathrm{C}_k$ is as defined in Fig. 6. Our goal then is to compute a field element from $\hat{c}_{k,j}$ which, by construction, is the encoding of $(\hat{x}((j-1)n_k + 1), \ldots, \hat{x}(jn_k))$ under C_k, where $\hat{x} = x \circ \Gamma_i$.

Recall that $n_k = 2^h$ for $h = \sum_{i=1}^{k} h_i$. It follows that $\mathrm{b}((j-1)n_k + 1) = (\mathrm{b}(j), 0^h)$ and $\mathrm{b}(jn_k) = (\mathrm{b}(j), 1^h)$. If $i \leq h$, then by these observations it holds that $\hat{x}((j-1)n_k + z) = x((j-1)n_k + \Gamma_i(z))$ for all $z \in [n_k]$, hence $\hat{c}_{k,j}$ can be simulated by accessing $c_{k,j}$ due to Proposition 5.9. If $i > h$, then it holds that $\hat{x}((j-1)n_k + z) = x((\Gamma_{i-h}(j) - 1)n_k + \Gamma_i(z))$ and we can use $c_{k,\Gamma_{i-h}(j)}$ to simulate; notice that Γ_{i-h} maps $[n/n_k]$ to itself and, therefore, $\Gamma_{i-h}(j) \leq n/n_k$ and $c_{k,\Gamma_{i-h}(j)}$ is a part of c_k and thus of $\mathrm{TC}^+(x)$.

□

Shifting messages encoded by TC^+ allows us to shift m-dimensional messages encoded by $(\mathrm{TC}^+)^m$ along each of the m dimensions, or even a subset of the dimensions. For any $j \in [m]$, define

$$\Gamma_i^j(t_1, \ldots, t_m) = (t_1, \ldots, \Gamma_i(t_j), \ldots, t_m). \tag{6}$$

More generally, for $(i_1, \ldots, i_m) \in (\mathbb{N} \cup \{0\})^m$, we denote $\Gamma_{i_1,\ldots,i_m} = \Gamma_{i_1}^1 \circ \cdots \circ \Gamma_{i_m}^m$ (note the definition is invariant to the order of composition). Letting $J = \{j \mid i_j \neq 0\}$ and $1_J \in \{0,1\}^m$ denote the binary vector that is 1 at any $j \in J$, we have that

$$\forall j \in [m],\ t_j \in \Lambda_{i_j} \implies \Gamma_{i_1,\ldots,i_m}(t_1, \ldots, t_m) = (t_1, \ldots, t_m) - 1_J. \tag{7}$$

(Recall Γ_0 is identity and $\Lambda_0 = \mathbb{N}$.)

Lemma 5.13 (Shifting under $(\mathrm{TC}^+)^m$)**.** *There exists a deterministic algorithm $\mathcal{A}$ that takes as input integers $i_1, \ldots, i_m \geq 0$ and coordinate $(t_1, \ldots, t_m)$ and, for any $x : [n_1] \times \cdots \times [n_m] \to \mathbb{F}$ s.t. n_j is divisible by 2^{i_j}, outputs $\hat{c}(t_1, \ldots, t_m)$ where $\hat{c} = (\mathrm{TC}^+)^m(x \circ \Gamma_{i_1,\ldots,i_m})$, by reading one field element from $c = (\mathrm{TC}^+)^m(x)$.*

Proof. It is sufficient to show how to apply a shift when $i_2 = \cdots = i_m = 0$, namely only along the first dimension, and compute one element from $\hat{c} = (\mathrm{TC}^+)^m(x \circ \Gamma_{i_1,0,\ldots,0})$ by reading one element from c. By performing this operation recursively, every time shifting along the next dimension $j = 2, \ldots, m$, the algorithm obtains a field element in $(\mathrm{TC}^+)^m(x \circ \Gamma_{i_1,\ldots,i_m})$ for any $i_1, \ldots, i_m$.

Let $d : [n_1] \times [n_2^+] \times \cdots \times [n_m^+] \to \mathbb{F}$ denote the m-dimensional tensor obtained by applying TC^+ over x along all dimensions except the first, namely $d = (I \otimes (\mathrm{TC}^+)^{m-1})(x)$ (here, $n_j^+ = O(n_j^{1+1/\mu+o(1)})$ is the length of a codeword in $\mathrm{TC}_{n_j}^+$).

Let $d_1 : [n_1] \to \mathbb{F}$, $c_1 : [n_1^+] \to \mathbb{F}$ and $\hat{c}_1 : [n_1^+] \to \mathbb{F}$ be the columns in d, c and, respectively, $\hat{c}$, that are parallel to the first axis and intersect $(t_1, \ldots, t_m)$.

It holds that $c_1 = \mathrm{TC}^+(d_1)$ and $\hat{c}_1 = \mathrm{TC}(d_1 \circ \Gamma_i^m)$. Hence, $\mathcal{A}$ may apply the simulator for TC^+ from Lemma 5.12 over c_1^+ with input t_1 to compute $\hat{c}_1(t_1) = \hat{c}(t_1, \ldots, t_m)$.

□

5.2.2 Checking Consistency in Flattened Codewords

In the above we show how to shift a message underlying a codeword of TC^m given (its extension from $(\mathrm{TC}^+)^m$) as oracle. The shifts we can perform are Γ_i^j, over the m-dimensional coordinate space $\mathbb{N}^m$, along any dimension $j \in [m]$ in the tensor (see Eq. (6)). In contrast, our goal of evaluating the consistency constraints over codewords of the flattened tensor tree code $\overline{\mathrm{TC}^m}$ involve shifts over a 1-dimensional coordinate space $[n] \subset \mathbb{N}$.

Recall the flattening of the tensor code is carried using the mapping $\varphi : \mathbb{N} \to \mathbb{N}^m$ from Fig. 4. The shift $t \mapsto t-1$ over the encoded message coordinates translates, then, to a shift w.r.t. φ over the message coordinates when "lifted" back to m dimensions, which we denote by $\mathrm{pre}_\varphi(t_1, \ldots, t_m) = \varphi(\varphi^{-1}(t_1, \ldots, t_m) - 1)$. While this function is different, in general, than shifting $(t_1, \ldots, t_m)$ along a certain dimension, our efforts in the previous section are not in vain. On a closer observation, it holds that for *most* of the coordinates $(t_1, \ldots, t_m) \in \mathbb{N}^m$, $\mathrm{pre}_\varphi(t_1, \ldots, t_m)$ is exactly $(t_1, \ldots, t_j - 1, \ldots, t_m)$ for some $j \in [m]$ and, therefore, can be emulated by Γ_i^j. We can also precisely specify the value of j corresponding to such a given coordinate $(t_1, \ldots, t_m)$. For that, let us define the predicate $\Psi_j : \mathbb{N}^m \to \{0, 1\}$ that is 1 on input $(t_1, \ldots, t_m)$ if and only if j is the smallest integer satisfying $t_j = \min_{j'} t_{j'}$.

Lemma 5.14. *Let $(t_1, \ldots, t_m) \in \mathbb{N}^m$ be such that $t_j > 1$ for all j. Let $j \in [m]$ be the integer satisfying $\Psi_j(t_1, \ldots, t_m) = 1$. Then, it holds that $\mathrm{pre}_\varphi(t_1, \ldots, t_m) = (t_1, \ldots, t_j - 1, \ldots, t_m)$.*

Proof. Assume $(t_1, \ldots, t_m)$ is "visited" by the recursion in Fig. 4 in step 2 of a recursive call φ_n^r that is restricted to a subset of r dimensions $D \subseteq [m]$ (recall each recursive call made in step 1.3. restricts the traversal to a subset of dimensions $D = [m] \setminus J$). If $n = 1$, then $(t_1, \ldots, t_m)$ contains a 1. Otherwise, the last coordinate visited by the traversal before $(t_1, \ldots, t_m)$ is $\mathbf{n}_j^{(1)}$ in the recursive call φ_{n-1}^1 restricted to some dimension $j \subseteq D$. It holds that $\Psi_j(t_1, \ldots, t_m) = 1$ since $D = \{j \mid t_j = \min_{j'} t_{j'}\}$ and the last subset of D iterated over by the loop is the last in lexicographic order, which contains all but the smallest j in D. It also holds that this last coordinate is $\mathbf{n}_j^{(1)} = (t_1, \ldots, t_j + 1, \ldots, t_m)$ by definition. □

Given the above, we may represent the consistency constraint over coordinates $t = (t_1, \ldots, t_m)$ such that $t_j > 1$ for all j by constraint (EQ1) in Fig. 9.

We complement our understanding of the function pre_φ by specifying its behavior on all coordinates that are 1 along at least one axis, namely, on *"axes-adjacent" coordinates*. Let $(t_1, \ldots, t_m)$ be such a coordinate with ones at some non-empty $D \subset [m]$. Our observation is that although the predecessor $\mathrm{pre}_\varphi(t_1, \ldots, t_m)$ might be unreachable form $(t_1, \ldots, t_m)$ by a small number of shifts $t_j \mapsto t_j - 1$, it is always reachable from some point on the D-parallel hyperplane that intersects $(t_1, \ldots, t_m)$. We give a precise characterization in the following.

Lemma 5.15. *Let $(t_1, \ldots, t_m) \in \mathbb{N}^m \setminus \{(1, \ldots, 1)\}$ be such that $D := \{j \mid t_j = 1\} \neq \emptyset$. Let $n = \min_{j \notin D} t_j$ and $H = \{j \mid t_j = n\} \cup D$, and let D' denote the subset $D' \subset H$ such that $H \setminus D'$ is the predecessor of $H \setminus D$ in the lexicographic order over subsets of H (used in Fig. 4). Let $(t'_1, \ldots, t'_m)$ be defined by*

$$t'_j = \begin{cases} n-1 & j \in D \cap D' \\ n & j \in D \setminus D' \\ t_j & j \notin D. \end{cases} \tag{8}$$

Then, it holds that $\mathrm{pre}_\varphi(t_1, \ldots, t_m) = (t'_1, \ldots, t'_m) - 1_{D' \setminus D}$.

Consistency Constraints over $A, A' : [n] \to \mathbb{F}$

For all $I \in \mathbf{Rect}(n)$:

(EQ1) $\forall (t_1, \ldots, t_m) \in I$ s.t. $\min_j t_j > 1$,

$$\sum_{j=1}^{m} \Psi_j(t_1, \ldots, t_m) \cdot \mathrm{EQ}\Big(A_I(t_1, \ldots, t_m), A'_I\big((t_1, \ldots, t_m) - 1_j\big)\Big) = 0$$

(EQ2) $\forall$ non-empty $D \subset [m]$, $j \in D$, $(t_1, \ldots, t_m) \in I$ s.t. $\min_{j' \in D} t_{j'} > 1$,

$$\mathrm{EQ}\Big(B_D(t_1, \ldots, t_m), B_D\big((t_1, \ldots, t_m) - 1_j\big)\Big) = 0$$

(EQ3) $\forall$ non-empty $D \subset [m]$, $(t_1, \ldots, t_m) \in I$ s.t. $\forall j \in D, t_j = 1$,

$$\mathrm{EQ}\big(B_D(t_1, \ldots, t_m), A_I(t_1, \ldots, t_m)\big) = 0$$

(EQ4) $\forall$ non-empty $D \subset [m]$, $(t_1, \ldots, t_m) \in I$,

$$\sum_{D' \subseteq [m]} \Phi_{D,D'}(t_1, \ldots, t_m) \cdot \mathrm{EQ}\Big(B_D(t_1, \ldots, t_m), A'_I\big((t_1, \ldots, t_m) - 1_{D' \setminus D}\big)\Big) = 0$$

- $\Psi_j(t_1, \ldots, t_m)$ outputs 1 if and only if j is the smallest integer satisfying $t_j = \min\{t_1, \ldots, t_m\}$.
- $\Phi_{D,D'}(t_1, \ldots, t_m)$ outputs 1 if only if $D \cap D' = \{j \mid t_j = \min\{t_1, \ldots, t_m\}\}$ and $D, D' \subseteq H := \{j \mid t_j = \min\{t_1, \ldots, t_m\} + 1\}$ and $H \setminus D'$ is the predecessor of $H \setminus D$ in the lexicographic order over the subsets of H.

Figure 9: Testing that $A(t) = A'(t-1)$ for all $1 < t \leq n$ using constraints over the lifting of A, A' to m-dimensions.

For intuition, note that $(t'_1, \ldots, t'_m)$ is the coordinate that, at any dimension $j = 1, \ldots, m$, takes the maximal value among $(t_1, \ldots, t_m)$ and $\mathrm{pre}_\varphi(t_1, \ldots, t_m)$.

Proof. Such $(t_1, \ldots, t_m)$ is the first coordinate visited by a recursive call to $\varphi_{n-1}^{|D|}$ over the restriction to D, which we denote by $\mathbf{n}_J^{(0)}$ in Fig. 4. Further, the call originates from a higher level in the recursion (either directly from the parent level or indirectly from a higher level) by the traversal restricted to some $H \supset D$. In particular, in this "ancestor" level, $\mathbf{n}_J^{(0)}$ is set to be $(t_1, \ldots, t_m)$ for $J = H \setminus D$.

The last coordinate visited before $(t_1, \ldots, t_m)$ is thus the last coordinate in a recursive call $\varphi_{n-1}^{|D'|}$ made in the ancestor level over the restriction to the subset D', where $H \setminus D'$ precedes J in the loop. This coordinate is $\mathbf{n}_{D'}^{(1)}$ by the notation of Fig. 4. The lemma follows since $\mathbf{n}_{D'}^{(1)}$ is precisely $(t'_1, \ldots, t'_m) - 1_{D' \setminus D}$ by definition. □

Given the above, our strategy to test consistency over coordinates $(t_1, \ldots, t_m)$ with some $t_j = 1$ is to let the prover provide auxiliary variables that form a "bridge" between $(t_1, \ldots, t_m)$

and $\mathrm{pre}_\varphi(t_1,\ldots,t_m)$ through the hyperplane that intersects $(t_1,\ldots,t_m)$ and is reachable from $\mathrm{pre}_\varphi(t_1,\ldots,t_m)$. By Lemma 5.15, there exists such a hyperplane for any $(t_1,\ldots,t_m)$, which is the hyperplane parallel to $D = \{j \mid t_j = 1\}$. Since D comes from a constant-size space (of size $2^{[m]}$), we can "pack" all of these bridges in a constant number of tensors, each containing bridges parallel to some D. Specifically, given an assignment $A : [n] \to \mathbb{F}$, we define for any D the word $B_D : I(n) \to \mathbb{F}$, which satisfies the following for any rectangle $I \subseteq I(n)$:

$$B_D(t_1,\ldots,t_m) = A_I(t'_1,\ldots,t'_m), \quad \text{where } t'_j = \begin{cases} 1 & j \in D \\ t_j & j \notin D. \end{cases} \tag{9}$$

In words, in B_D, we "spread" the A-value from any $(t_1,\ldots,t_m)$ that is "adjacent to the D-axes", i.e. where $\{j \mid t_j = 1\} = D$, along the D-parallel hyperplane that contains it, i.e. the hyperplane $\{(t'_1,\ldots,t'_m) \mid \forall j \notin D, t'_j = t_j\}$.

Given the "bridges" B_D, we check consistency between the remaining coordinates in A_I and A'_I by the following tests, which we include in Fig. 9:

(EQ2) Test that any B_D is constant over any D-parallel hyperplane.

(EQ3) Test that B_D and A_I are equal over the coordinates adjacent to the axes of D.

(EQ4) Test that when shifting A'_I according to Lemma 5.15, we obtain a coordinate on the D-parallel hyperplane that is equal to B_D. For any non-empty $D \subset [m]$ and $D' \subseteq [m]$, we define a function $\Phi_{D,D'} : \mathbb{N}^m \to \{0,1\}$ that, on input $(t'_1,\ldots,t'_m)$ outputs 1 if and only if, letting $n = \min_j t'_j + 1$ and $H = \{j \mid t'_j \le n\}$, it holds that: (i) $\{j \mid t'_j = n-1\} = D \cap D'$, (ii) $D, D' \subseteq H$, and (iii) $H \setminus D'$ is the predecessor of $H \setminus D$ in the lexicographic order over subsets of H. By Lemma 5.15, for any $(t'_1,\ldots,t'_m)$ such that $\Phi_{D,D'}(t'_1,\ldots,t'_m) = 1$, it holds that $\mathrm{pre}_\varphi(t_1,\ldots,t_m) = (t'_1,\ldots,t'_m) - 1_{D'\setminus D}$, where $(t_1,\ldots,t_m)$ is defined by $t_j = 1$ at $j \in D$ and $t_j = t'_j$ at $j \notin D$. Hence, we conclude the test with the constraint.

We conclude with the following lemma, which is implicitly implied by the above discussion.

Lemma 5.16 (Flattened Consistency Constraints)**.** *The following two conditions are equivalent for any $A, A' : [n] \to \mathbb{F}$:*

- *$A(t) = A'(t-1)$ for all $1 < t \le n$ for which there exists a rectangle $I \in \mathbf{Rect}(n)$ that contains both $\varphi(t)$ and $\varphi(t-1)$.*
- *There exist $\{B_D : I(n) \to \mathbb{F} \mid D \subset [m], D \neq \emptyset\}$ such that for every rectangle $I \in \mathbf{Rect}(n)$, all constraints in Fig. 9 hold for A_I and A'_I.*

Further, if the former condition holds, then the latter holds with $\{B_D\}$ defined by Eq. (9).

Proof. Suppose there exists t such that $A(t) \neq A'(t-1)$ and let I be any rectangle in $\mathbf{Rect}(n)$ that contains both $\varphi(t) = (t_1,\ldots,t_m)$ and $\varphi(t-1) = \mathrm{pre}_\varphi(t_1,\ldots,t_m)$. Then, it holds that $A_I(t_1,\ldots,t_m) \neq A'_I(\mathrm{pre}_\varphi(t_1,\ldots,t_m))$.

If $\min_j t_j > 1$, then by Lemma 5.14 and Eq. (7), (EQ1) is not satisfied by $(t_1,\ldots,t_m)$ w.r.t. A_I, A'_I (note for every $(t_1,\ldots,t_m)$ there is a unique j such that $\Psi_j(t_1,\ldots,t_m) = 1$).

If $D = \{j \mid t_j = 1\}$ is non-empty, let $(t'_1,\ldots,t'_m)$ and D' be the coordinate and subset defined by Lemma 5.15 satisfying $\mathrm{pre}_\varphi(t_1,\ldots,t_m) = (t'_1,\ldots,t'_m) - 1_{D'\setminus D}$. Note that $(t'_1,\ldots,t'_m)$ is also contained in I since it is bounded by $(t_1,\ldots,t_m)$ or $\mathrm{pre}_\varphi(t_1,\ldots,t_m)$ at any dimensions $j \in [m]$. Then, it either holds that (i) $A_I(t_1,\ldots,t_m) \neq B_D(t_1,\ldots,t_m)$, or (ii) $B_D(t_1,\ldots,t_m) \neq B_D(t'_1,\ldots,t'_m)$, or (iii) $B_D(t'_1,\ldots,t'_m) \neq A'_I((t'_1,\ldots,t'_m) - 1_{D'\setminus D})$.

In the first case, (EQ3) is not satisfied.

In the second case, since $(t_1, \ldots, t_m)$ and $(t'_1, \ldots, t'_m)$ reside on the same D-parallel hyperplane, then there exists $(t''_1, \ldots, t''_m)$ on the hyperplane where $B_D(t''_1, \ldots, t''_m) \neq B_D(t''_1, \ldots, t''_j - 1, \ldots, t''_m)$ for some $j \in D$ as otherwise all values on the hyperplane are equal. Hence, Item (EQ2) does not hold.

Lastly, in the case where $B_D(t'_1, \ldots, t'_m) \neq A'_I((t'_1, \ldots, t'_m) - 1_{D' \setminus D})$, (EQ4) is not satisfied since $\Psi_{D,D'}(t'_1, \ldots, t'_m) = 1$ and $\Psi_{D,D''}(t'_1, \ldots, t'_m) = 0$ for any other $D'' \neq D'$ (as D' is determined uniquely by $(t'_1, \ldots, t'_m)$ and D).

The the other direction follows by inspection: assuming that the first condition in the lemma holds and let $\{B_D\}$ be as defined in Eq. (9), then (EQ1) follows by Lemma 5.14, (EQ2) and (EQ3) by the definition of B_D and (EQ4) by Lemma 5.15. □

We address two minor gaps that are left by Lemma 5.16 towards realizing our goal of consistency evaluation under codewords.

The first gap is that, in the lemma, we give a representation of the consistency constraint over all coordinates, except those where no rectangle $I \in \mathbf{Rect}(n)$ contains both them and their predecessor under φ. This will be the set of coordinates $\mathbf{Bad}(n)$ that we do not cover in the statement of Lemma 5.8. In the following, we show that the size of $\mathbf{Bad}(n)$ is bounded by a constant.

Lemma 5.17. *Let* $\mathbf{Bad}(n)$ *be the set of all coordinates* $1 < t \leq n$ *such that* $\{\varphi(t), \varphi(t-1)\} \not\subseteq I$ *for all* $I \in \mathbf{Rect}(n)$*. Then,* $|\mathbf{Bad}(n)| = O(1)$*.*

Proof. Let $(n_1, \ldots, n_m) = \varphi(n)$. Let $t \in [n]$ and $(t_1, \ldots, t_m) = \varphi(t)$. If $\min_j t_j > 1$, then, by Lemma 5.14, $\varphi(t-1)$ is contained in any $I \in \mathbf{Rect}(n)$ that contains $\varphi(t)$.

Otherwise, $D = \{j \mid t_j = 1\} \neq \emptyset$. Let $n^*_1 > n^*_2 > \ldots$ denote the distinct values in $(n_1, \ldots, n_m)$ from largest to smallest, and let $J_i = \{j \mid n_j = n^*_i\}$. Let $J'_i = \{j \mid t_j = n^*_i\}$.

Let i be the smallest integer such that J'_i precedes J_i, in the lexicographic order over subsets of $[m] \setminus (\bigcup_{i'<i} J_i)$ (followed in Fig. 4). Note that since $\varphi(t) \leq \varphi(n)$, it must hold that $J'_{i'} = J_{i'}$ for all $i' < i$.

Let $n' = \min_{j \notin D} t_j$ be the second smallest value in $(t_1, \ldots, t_m)$ and let $(t'_1, \ldots, t'_m)$ be as defined in Eq. (8). (Recall $t'_j \leq n'$ at all locations j where $t_j \leq n'$ and is equal to t_j otherwise.)

If $n' < n^*_i$, then $(t'_1, \ldots, t'_m)$ also precedes $(n_1, \ldots, n_m)$ in the mapping φ, therefore $(t'_1, \ldots, t'_m) \in I(n)$ and any rectangle in $\mathbf{Rect}(n)$ that contains t' contains both $\varphi(t)$ and $\varphi(t-1)$ by definition.

The case where $n' = n^*_i$ corresponds to a coordinate $(t_1, \ldots, t_m)$ where: At $j \in \bigcup_{i'<i} J_i$, $t_j = n_j$ and, at any other j, $t_j \in \{1, n^*_i\}$. The number of such coordinates is then bounded by the number of choices for i' and a subset of $[m] \setminus \bigcup_{i'<i} J_i$. Hence, it depends only on m and is constant in n. □

The second gap is due to the fact that, on the one hand, we know how to evaluate shift functions Γ^j_i only when the length of the tensor at the j^{th} dimension, namely n_j, is divisible by 2^i (Lemma 5.13), whereas on the other hand, the constraints in Fig. 9 seem to require applying shifts over assignments of arbitrary length. (Note that padding the assignments will break monotonicity and is therefore out of the question.)

We resolve this issue by letting the prover add another set of auxiliary variables (similarly to $\{B_D\}$ from above) that are always of length suitable for applying the shifts.

Let A_I be an assignment over some $I = [n_1] \times \cdots \times [n_m]$. For any $i \in \{0, \ldots, \lceil \log n_j \rceil\}$, let $n_j(i) = 2^i \cdot \lfloor n_j / 2^i \rceil$.

For any $(i_1, \ldots, i_m)$, where $i_j \in \{0, \ldots, \lceil \log n_j \rceil\}$, let $G_{i_1,\ldots,i_m} := G_{i_1,\ldots,i_m}(A_I) \in \mathbb{F}^{n_1(i_1)\times\cdots\times n_m(i_m)}$ be defined as follows:

$$G_{i_1,\ldots,i_m}(t_1, \ldots, t_m) = \begin{cases} A(\varphi^{-1}(t_1, \ldots, t_m)) & \forall j,\ t_j \in \Lambda_{i_j} \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

Notice that $G_{i_1,\ldots,i_m}$ is well-defined since if $t_j \le n_j(i_j)$ and $t_j \in \Lambda_{i_j}$, then $t_j \le n_j$. (Recall $t \in \Lambda_i$ means 2^{i-1} divides $t-1$.)

Given $G_{i_1,\ldots,i_m}$, we may rewrite the consistency constraint between A_I and the shift of any other A'_I along dimensions $J \subseteq [m]$, over coordinates $(t_1, \ldots, t_m) \in \Lambda_{i_1} \times \cdots \times \Lambda_{i_m}$ where $i_j = 0$ for all $j \in J$, as follows:

$$\begin{aligned} &\mathrm{EQ}\Big(A_I(t), A'_I\big(t - 1_J\big)\Big) \\ &= \mathrm{EQ}\big(A_I(t), A'_I(\Gamma_{i_1,\ldots,i_m}(t))\big) \\ &= 1 - \Big(1 - \mathrm{EQ}\big(A_I(t), G_{i_1,\ldots,i_m}(t)\big)\Big) \cdot \Big(1 - \mathrm{EQ}\big(A'_I(t), G_{i_1,\ldots,i_m}(\Gamma_{i_1,\ldots,i_m}(t))\big)\Big). \end{aligned} \tag{11}$$

Importantly, the above transformation is sound: If the left-hand side is non-zero (meaning inequality), then the right-hand side is non-zero even when replacing $G_{i_1,\ldots,i_m}$ with any arbitrary binary function. The above observations imply the following lemma.

Lemma 5.18. *For any $A, A' : [n] \to \{0,1\}$, $I \in \mathbf{Rect}(n)$ and $i_1, \ldots, i_m$ such that $i_j \in \{0, \ldots, \lceil \log n_j \rceil\}$, Eq. (11) holds for $G_{i_1,\ldots,i_m}$ as defined in Eq. (10). Further, if the LHS of the equation is 1, then the RHS is 1 for any choice of $G_{i_1,\ldots,i_m}$.*

5.2.3 Evaluating The Coefficients Ψ and Φ

In Fig. 9, we formulate the consistency constraints over assignments as low-degree constraints over their corresponding lifting to m dimensions and their shifts. The low-degree constraints involve coefficients defined by the binary functions $\{\Psi_j\}$ and $\{\Phi_{D,D'}\}$ over the coordinate space. For the verifier to locally evaluate the constraint-evaluation codewords as desired (Lemma 5.8), he must be able to locally evaluate the codewords encoding these coefficients. (Given the code $\overline{\mathrm{TC}^m}$ is linear, multiplication (Propositions 5.4 and 5.7) and allows evaluating shifts (Lemma 5.13), this is also the only remaining component.)

Although the coefficients are independent in the prover's statement and, in fact, can be locally computed by the verifier (by simply computing the above predicates on any given coordinate), it is not clear if the verifier can locally compute any location in a $\overline{\mathrm{TC}^m}$-codeword encoding the coefficients. Instead, we let the prover provide these encodings to the verifier, together with a proof of their validity.

For a function $f : [n_1] \times \cdots \times [n_m] \to \mathbb{F}$ (think of $f \in \{\Psi_j, \Phi_{D,D'}\}$), let C_f be the circuit that takes as input $(\mathrm{b}(t_1), \ldots, \mathrm{b}(t_m)) \in \{0,1\}^M$ where $M = \sum_j \lceil \log n_j \rceil$ – recall this is the binary representation of $(t_1 - 1, \ldots, t_m - 1)$ – and a value $F \in \{0,1\}$ and outputs 1 if and only if $f(t_1, \ldots, t_m) = F$.

To validate that a certain codeword $F : [n_1]\times\cdots\times[n_m] \to \mathbb{F}$ encodes the output of C_f, it suffices to verify satisfiability of C_f under all assignments $(\mathrm{b}(t_1), \ldots, \mathrm{b}(t_m), F(t_1, \ldots, t_m))$. Lemma 5.1 already provides us with a set of constraints $\mathbf{Eval}(n, C_f)$ that express satisfiability of C_f by a given assignment and can be evaluated "under codewords". Equipped with this machinery, to prove that a codeword $\widetilde{F} : [n] \to \Sigma$ encodes $F : [n_1] \times \cdots \times [n_m] \to \mathbb{F}$, the prover additionally encodes:

- $T^1, \ldots, T^M : [n_1]\times\cdots\times[n_m] \to \mathbb{F}$ where $(T^1(t_1, \ldots, t_m), \ldots, T^M(t_1, \ldots, t_m)) = (\mathrm{b}(t_1), \ldots, \mathrm{b}(t_m))$.

- $W_f^1, \ldots, W_f^K : [n_1] \times \cdots \times [n_m] \to \mathbb{F}$, witnesses for $\mathbf{Eval}(n, C_f)$ computed w.r.t. assignments $(T^1, \ldots, T^M, F)$, as induced by Lemma 5.1.

It remains to show how to test that the encoded $T^1, \ldots, T^M$ indeed compose the binary representation of the coordinate space.

Let $\mathrm{b}^i(t)$ denote the i^{th} least significant bit in $\mathrm{b}(t)$. We want to test whether $T(t_1, \ldots, t_m) \equiv \mathrm{b}^i(t_j)$, for a given $T : [n] \to \mathbb{F}$ and $j \in [m]$, $i \lceil \log n_j \rceil$. We express this equality using constraints that again involve the degree-2 function EQ and the shift functions Γ_i^j. To that end, we observe that $\mathrm{b}^i(t)$ is the unique binary function over $\mathbb{N}$ that: (i) evaluates 0 at $t = 1$, (ii) gives equal values at t and $\Gamma_{i'}(t)$ for any t and $i' < i$, and (iii) gives distinct values at t and $\Gamma_{i'}(t)$ for any t and $i' \geq i$.

T-Constraints for $T : [n_1] \times \cdots \times [n_m] \to \{0, 1\}$

(T1) $\forall (t_1, \ldots, t_m) \in [n_1] \times \cdots \times [n_m]$ s.t. $t_j = 1$,

$$T(t_1, \ldots, t_m) = 0$$

(T2) For $i' = 1, \ldots, i-1$, $\forall (t_1, \ldots, t_m) \in [n_1] \times \cdots \times [n_m]$,

$$\mathrm{EQ}\Big(T(t_1, \ldots, t_m), T\big(\Gamma_{i'}^j(t_1, \ldots, t_m)\big)\Big) = 0$$

(T3) For $i' = i, \ldots, \lceil \log n_j \rceil$, $\forall (t_1, \ldots, t_m) \in [n_1] \times \cdots \times [n_m]$,

$$\mathrm{EQ}\Big(T(t_1, \ldots, t_m), 1 - T\big(\Gamma_{i'}^j(t_1, \ldots, t_m)\big)\Big) = 0$$

Figure 10: Testing that $T(t) = \mathrm{b}^i(t_j)$ for all $t = (t_1, \ldots, t_m)$.

Lemma 5.19 (T-Constraints)**.** *Let $T : [n_1] \times \cdots \times [n_m] \to \{0, 1\}$. Then, $T(t_1, \ldots, t_m) = \mathrm{b}^i(t_j)$ for all $(t_1, \ldots, t_m) \in [n_1] \times \cdots \times [n_m]$ if and only if T satisfies all constraints in Fig. 10.*

Proof. It is by inspection that $T \equiv b^i(t_j)$ satisfies all constraints in Fig. 10. For the other direction, assume $T(t_1, \ldots, t_m) \neq \mathrm{b}^i(t_j)$ at some $t = (t_1, \ldots, t_m)$. Consider the sequence of coordinates $t^r = (t_1, \ldots, t_j^r, \ldots, t_m)$ for $r = \lceil \log t_j \rceil + 1, \ldots, 1$, where t_j^r is defined inductively as follows:

1. $t_j^{\lceil \log t_j \rceil + 1} = t_j$.
2. $t_j^r = \Gamma_r(t_j^{r+1})$ if $\mathrm{b}^r(t_j^{r+1}) = 1$ and $t_j^r = t_j^{r+1}$ otherwise.

It holds that $t_j^1 = 1$ and therefore $\mathrm{b}^i(t_j^1) = 0$. If $T(t^1) \neq \mathrm{b}^i(t_j^1)$, then $T(t^1) \neq 0$ and (T1) does not hold. Otherwise, let i' be the largest value of r such that $T(t^r) = \mathrm{b}^i(t_j^r)$. If $\mathrm{b}^{i'}(t_j^{i'+1}) = 0$ then $t^{i'} = t^{i'+1}$ by definition and $T(t^{i'+1}) = T(t^{i'}) = \mathrm{b}^i(t_j^{i'}) = \mathrm{b}^i(t_j^{i'+1})$, in contradiction to the maximality of i'. If $\mathrm{b}^{i'}(t_j^{i'+1}) = 1$ then $t^{i'} = \Gamma_{i'}^j(t^{i'+1})$ and we again split to two cases: If $i' < i$, then $\mathrm{b}^i(t^{i'}) = \mathrm{b}^i(t^{i'+1})$ and, by definition of i', $T(t^{i'+1}) \neq T(t^{i'})$ thus breaking (T2). If $i' \geq i$, then $\mathrm{b}^i(t^{i'}) = 1 - \mathrm{b}^i(t^{i'+1})$ and $T(t^{i'+1}) = T(t^{i'})$, breaking (T3). □

5.2.4 Proof of Lemma 5.8

Before laying down the set of constraints **EQ**, let us first describe the variables $W^1, \ldots, W^K$ over which they are applied, besides the variables A, A'. Although the W^i are defined in the lemma to be over $[N_i]$, it is more natural to think about some of them as functions over m-dimensional domains of size N_i. We also describe the values $W^1_A, \ldots, W^K_A$ that these variables take to attest that a given pair of assignments A, A' is consistent (note the witnesses are a function of A alone and some of them do not even depend on A, in which case the same witness can be used for all assignments):

1. For any non-empty $D \subseteq [m]$,
$$B_D : I^B_D(n) \to \mathbb{F},$$
where $I^B_D(n) = \bigcup_{[n_1]\times\cdots\times[n_m]\in\mathbf{Rect}(n)} [\hat{n}_1] \times \cdots \times [\hat{n}_m]$, for $\hat{n}_j = n_j$ for $j \notin D$ and, for $j \in D$, $\hat{n}_j$ is the smallest power of 2 that is at least n_j.
Given A, we define $B_D := B_D(A)$ as in Eq. (9) (note it is well-defined over $I^B_D(n)$).

2. For any $i_1, \ldots, i_m$ such that $i_j \in \{0, \ldots, \lceil \log n'_j \rceil\}$ for $n'_j = \max_{(t_1,\ldots,t_m)\in I(n)} t_j$,
$$G_{i_1,\ldots,i_m} : I^G_{i_1,\ldots,i_m}(n) \to \mathbb{F},$$
where $\hat{I}_{i_1,\ldots,i_m}(n) = \bigcup_{[n_1]\times\cdots\times[n_m]\in\mathbf{Rect}(n)} [n_1(i_1)] \times \cdots \times [n_m(i_m)]$ and, recall, $n_j(i) = 2^i \cdot \lfloor n_j/2^i \rceil$.
Given A, we define $G_{i_1,\ldots,i_m} := G_{i_1,\ldots,i_m}(A)$ to be consistent with Eq. (10) for all $I \in \mathbf{Rect}(n)$.

3. $T^1, \ldots, T^M : [N]^m \to \mathbb{F}$, where N is the smallest power of 2 that such that $I(n) \subseteq [N]^m$ and $M = m \cdot \lceil \log N \rceil$ (by Proposition 4.8, it holds that $N \leq 2\lceil n^{1/m} \rceil = O(n^{1/m})$).
The value of any T^r in the witness is independent in A, A' and is always $T^r(t_1, \ldots, t_m) = \mathrm{b}^i(t_j)$, where $j = \lfloor r / \lceil \log N \rceil \rfloor$ and $i = r \mod \lceil \log N \rceil$.

4. For any $f \in \{\Psi_j \mid j \in [m]\} \cup \{\Phi_{D,D'} \mid D, D' \subseteq [m]\}$ (defined in Fig. 9):

   4.1. $F_f : [N^m] \to \mathbb{F}$.
   The value of F_f in the witness is independent in A, A' and satisfies $F_f(t) = f(\varphi(t))$ for all $t \in [N^m]$ (recall $I(N^m) = [N]^m$).

   4.2. $W^1_f, \ldots, W^{K'}_f : [N^m] \to \mathbb{F}$.
   The value of $(W^1_f, \ldots, W^{K'}_1)$ in the witness is independent in A, A' and is the witness for constraints $\mathbf{Eval}(C_f)$ w.r.t. assignment $(T^1, \ldots, T^M, f(T^1, \ldots, T^M))$ (Lemma 5.1).

Note that the number $K = K(n)$ of additional variables we have introduced is polylogarithmic in n (and exponential in the constant m) since $n'_j \leq \lceil n^{1/m} \rceil$ (Proposition 4.8) and the circuit that computes Ψ_j and $\Phi_{D,D'}$ is of size polynomial in its input. Further, the size of each of the new variables is at most $O(n)$.

Next, we list the constraints in **EQ**. To that end, we introduce the following notation which roughly generalizes $\mathbf{Rect}(n)$: For any function X, we denote by $\mathbf{Rect}(X)$ the set of all maximal rectangles $I \in \mathrm{dom}(X)$, i.e. where any rectangle $I' \subseteq \mathrm{dom}(X)$ is contained in some $I \in \mathbf{Rect}(X)$ and, further, any $I' \in \mathbf{Rect}(X)$ is not contained in any other rectangle in $\mathbf{Rect}(X)$ but itself. Crucially to us, for any W^i in the list of witnesses above, $\mathbf{Rect}(W^i) \leq 2^m$ due to Proposition 4.8. (In the following, we define the rectangle R in any $(P, R) \in \mathbf{EQ}$ as a function of n. However, the construction implicitly defines an infinite rectangle R that complies with the syntax of the lemma.)

1. (Z-Constraints) For any $X \in \{A, A'\} \cup \{G_{i_1,\dots,i_m}\} \cup \{T^i\}$ and any $I \in \mathbf{Rect}(X)$,

$$(Z_{X,I}, \mathbb{N}^m) \in \mathbf{EQ},$$

where $Z_{X,I}(A, A', W^1, \dots, W^K) : I \to \mathbb{F}$ is the function that evaluates $X_I(t)(1 - X_I(t))$ at all $t \in I$. Recall this is the degree-2 polynomial that is identically zero if and only if X_I is Boolean (similarly to Eq. (4)).

2. (EQ-Constraints) We add the constraints derived from Fig. 9, via Eq. (7) and Lemma 5.18:

(EQ1) For any $I = [n_1] \times \cdots \times [n_m] \in \mathbf{Rect}(n)$ and $i = (i_1, \dots, i_m)$ s.t. $0 \le i_j \le \lceil \log(n_j) \rceil$,

$$(EQ^{(1)}_{I,i_1,\dots,i_m}, R^{(1)}_{i_1,\dots,i_m}) \in \mathbf{EQ},$$

where $EQ^{(1)}_{I,i_1,\dots,i_m}(A, A', W^1, \dots, W^K) : I \to \mathbb{F}$ evaluates

$$\sum_{j=1}^{m} (F_{\Psi_j})_I(t) \cdot \Big(1 - \Big(1 - \mathrm{EQ}\big(A_I(t), G_{0,\dots,i_j,\dots,0}(t)\big)\Big) \cdot \Big(1 - \mathrm{EQ}\big(A'_I(t), G_{0,\dots,i_j,\dots,0}(\Gamma_{0,\dots,i_j,\dots,0}(t))\big)\Big)\Big)$$

at any $t \in I$, and $R^{(1)}_{i_1,\dots,i_m} = \{(t_1, \dots, t_m) \mid t_j \in \Lambda_{i_j},\ t_j > 1\ \forall j\}$.

(EQ2) For any non-empty $D \subset [m]$, any $I = [n_1] \times \cdots \times [n_m] \in \mathbf{Rect}(B^D)$, any $j \in D$, and any $0 \le i_j \le \log(n_j)$ (recall n_j is a power of 2),

$$(EQ^{(2)}_{I,D,j,i_j}, R^{(2)}_{D,j,i_j}) \in \mathbf{EQ},$$

where $EQ^{(2)}_{I,D,j,i_j}(A, A', W^1, \dots, W^K) : I \to \mathbb{F}$ evaluates

$$\mathrm{EQ}\Big(B_D(t), B_D\big(\Gamma_{0,\dots,i_j,\dots,0}(t)\big)\Big)$$

at any $t \in I$, and $R^{(2)}_{D,j,i_j} = \{(t_1, \dots, t_m) \mid t_j \in \Lambda_{i_j},\ t_{j'} > 1\ \forall j' \in D\}$.

(EQ3) For any $I \in \mathbf{Rect}(n)$ and non-empty $D \subset [m]$,

$$(EQ^{(3)}_{I,D}, R^{(3)}_{D}) \in \mathbf{EQ},$$

where $EQ^{(3)}_{I,D}(A, A', W^1, \dots, W^K) : I \to \mathbb{F}$ evaluates $\mathrm{EQ}\big(B_D(t), A_I(t)\big)$ at any $t \in I$, and $R^{(3)}_D = \{(t_1, \dots, t_m) \mid t_j = 1\ \forall j \in D\}$.

(EQ4) For any $I \in \mathbf{Rect}(n)$, non-empty $D \subset [m]$ and $i = (i_1, \dots, i_m)$ s.t. $0 \le i_j \le \lceil \log(n_j) \rceil$,

$$(EQ^{(4)}_{I,D,i_1,\dots,i_m}, R^{(4)}_{i_1,\dots,i_m}) \in \mathbf{EQ},$$

where, letting $i(J) = (i'_1, \dots, i'_m)$ be $i'_j = i_j$ if $j \in J$ and $i'_j = 0$ otherwise, the function $EQ^{(4)}_{I,D,i_1,\dots,i_m}(A, A', W^1, \dots, W^K) : I \to \mathbb{F}$ evaluates

$$\sum_{D' \subseteq [m]} (F_{\Phi_{D,D'}})_I(t) \cdot \Big(1 - \Big(1 - \mathrm{EQ}\big(B_D(t), G_{i(D' \setminus D)}(t)\big)\Big) \cdot \Big(1 - \mathrm{EQ}\big(A'_I(t), G_{i(D' \setminus D)}(\Gamma_{i(D' \setminus D)}(t))\big)\Big)\Big)$$

at any $t \in I$, and $R^{(4)}_{i_1,\dots,i_m} = \{(t_1, \dots, t_m) \mid t_j \in \Lambda_{i_j}\ \forall j\}$.

3. (T-Constraints) For any $r \in [M]$, we add to $\mathbf{EQ}$ the constraints from Fig. 10 over T^r (where $n_1 = \cdots = n_m = N$), with $j = \lfloor r / \lceil \log N \rceil \rfloor$ and $i = r \mod \lceil \log N \rceil$. Note the figure defines $1 + \lceil \log N \rceil$ constraints over T^r in total.

4. (Coefficient Evaluation Constraints) For $f \in \{\Psi_j \mid j \in [m]\} \cup \{\Phi_{D,D'} \mid D, D' \subseteq [m]\}$, we add all constraints from $\mathbf{Eval}(N^m)$ over $T^1, \ldots, T^M$, F_f and $W_f^1, \ldots, W_f^{K'}$.

Correctness (both completeness and soundness) follows, by inspection, from Lemmas 5.1, 5.16, 5.18 and 5.19.

In any constraint $(P, R) \in \mathbf{EQ}(n)$, the function $E = P(A, A', W^1, \ldots, W^K) : I \to \mathbb{F}$ has the form $E(t) = p(A_I(t), A'_I(t), \hat{W}^1(t), \ldots, \hat{W}^K(t))$, where p is a polynomial over $\mathbb{F}$ of degree at most 5 and $\hat{W}^i = W^i \circ \Gamma_{i_1,\ldots,i_m}$ for some $i_1, \ldots, i_m$ where 2^{i_j} divides the length of the j^{th} dimension in the tensor W^i for all j. This allows applying the Γ shifts over W^i under their $(\mathrm{TC}^+)^m$ encoding using Lemma 5.13 separately on each of the $O(n^{\mu+o(1)})$ field elements in the codeword symbol, to obtain any symbol from the encoding of any $\hat{W}^i$ under TC^m (which is a restriction of $(\mathrm{TC}^+)^m$). Given additionally the linearity of the code and its multiplication property (Propositions 5.4, 5.6 and 5.7), which allow us to evaluate degree-5 polynomials, we obtain the codeword evaluation algorithm.

6 The Zero Test

In Section 5, we express the transition and consistency constraints over the assignments in the tree PCP (Fig. 1) as a conjunction of constraints of the form $E_R \equiv 0$, where E is an m-dimensional tensor over a rectangle $I \subset \mathbb{N}^m$ and $R \subseteq I$ is a sub-rectangle. We now show how to efficiently test statements of the form $E_R \equiv 0$ given access to the codeword $\widetilde{E}$.

Our goal in general is to design a "zero test" for tree code tensors. In the test, the verifier is given oracle access to a codeword $c \in \mathrm{TC}^m$, that systematically encodes a message $x : [n_1] \times \cdots \times [n_m] \to \mathbb{F}$ (see Section 3.2), and a proof π which allows him to test whether x is all zeros over a certain rectangle $T = T_1 \times \cdots \times T_m$, i.e. if $x_T \equiv 0$.

We require a strong notion of soundness that makes accepting proofs for the zero test robustly incremental (Definition 3.2) and thus applicable to tree PCPs. We require that an accepting proof must be close to the "correct" proof, prescribed by an honest prover algorithm. In particular, any oracle that is far from the prescribed proof must be rejected by the verifier, even when the statement is correct. This is similar to the notion of unambiguity that was first considered in [RRR16] in the context of interactive proofs.

Second, we require that the prescribed proof is robustly incremental, namely that a proof for a statement (c, T), even if slightly corrupted, can be extended to the prescribed proof for any "extension" of the statement, i.e. (c', T') such that $c'(T') \equiv 0$ where $c \preceq c'$, $T \subseteq T'$, and $T' \setminus T$ does not contain elements in the domain of c. Notice that these two conditions imply incrementality of accepting proofs as required by the tree PCP definition (Definition 1.4): If a proof is accepting, then it is equal to the prescribed proof up to a few corruptions and can be therefore extended.

Recall that $\widetilde{E}$ from Lemmas 5.1 and 5.8 is a codeword in a *multiplication code* that does not have an explicit encoding function. Consequently, we shall not assume in the following such an encoding function. However, the message x encoded by a given codeword c is well-defined (Remark 4.14 and Propositions 5.6 and 5.7), and we denote $c \in \mathrm{TC}(x)$ (or $c \in \mathrm{TC}^m(x)$ in the tensor case).

As usual, we think of the codeword oracle as a function $c : [n_1] \times \cdots \times [n_m] \to \mathbb{F}^{L(n_1) \times \cdots \times L(n_m)}$. We denote for brevity $L_j := L(n_j)$ and $n := \max_j n_j$.

Lemma 6.1 (Zero Test)**.** *Let $\mathbb{F} = \{\mathbb{F}(n)\}$ be such that $\mathbb{F}(n-1) \subseteq \mathbb{F}(n)$ and let $m \geq 2$ be a constant such that $|\mathbb{F}(0)| > m^2/(m-1)$.*

Let $\mathrm{TC} = \{\mathrm{TC}_n \subset (\mathbb{F}(n)^{L(n)})^n\}$ *be a systematic linear tree code over* $\mathbb{F}$ *and let* $\widehat{\mathrm{TC}} : \mathbb{F}(0)^n \to (\mathbb{F}(n)^{\hat{L}(n)})^n$ *be a linear tree code with explicit polynomial-time encoding function. Assume the tree distance of* TC *and* $\widehat{\mathrm{TC}}$ *is at least* δ.

Than, there exists an ensemble of oracles $\{\pi_{c,T} \mid c : [n_1] \times \cdots \times [n_m] \to \mathbb{F}(n)^{L_1 \times \cdots \times L_m},\ T = T_1 \times \cdots \times T_m \subset [n_1] \times \cdots \times [n_m]\}$, *a distance function* Δ, *and a probabilistic verifier* V *that takes as input a rectangle* $T \subset [n_1] \times \cdots \times [n_m]$ *and has oracle access to* (c, π), *that satisfy:*

- *(Prescribed Completeness) For any* $c \in \mathrm{TC}^m(x)$ *and* T, *with probability all but negligible,* $V^{(c,\pi_{c,T})}(T)$ *outputs* 1 *if and only if* $x_T \equiv 0$.
- *(Unambiguity) There exists* $\delta' = O((\delta(n)/H_n)^{2m})$, *such that for any* c, T *and* π *where* $\Delta(\pi, \pi_{c,T}) \geq \delta'$, $\Pr[V^{c,\pi}(T) = 1] < n^{-\omega(1)}$.
- *(Query Complexity)* $V^{c,\pi}$ *makes a total of* $O\Big((n_1 + \cdots + n_m) \cdot (\log(n)/\delta(n))^{3m+2}\Big)$ *queries to its oracles.*
- *(Robust Incrementality) Assume* $\widehat{\mathrm{TC}}$ *is* $(\Delta_{\mathsf{S}}, q, \delta'')$*-LCC. Then, there exists a constant* γ *such that, for any (possibly infinite) rectangle* $T \subseteq \mathbb{N}^m$, *the ensemble* $\{\pi_{c,T} \mid c \in \mathrm{TC}^m(x)\ s.t. x_{T \cap \mathrm{dom}(x)} \equiv 0\}$ *is* $(\Delta, \delta''(n)^m, L(n)^m \hat{L}(n)^m \cdot n^{\gamma/m}, m \cdot L(n)^m \hat{L}(n)^m)$*-robustly incremental.*

Following classic PCP constructions [BFL90, BFLS91], our zero test proof oracle is derived by an interactive protocol between a verifier and a prover for proving $x_T \equiv 0$, which is based on the *sumcheck protocol* [LFKN92]. Jumping ahead, we will later "flatten" this interactive protocol into a tree PCP. In the protocol, the statement $x_T \equiv 0$ is first reduced to a statement of the form

$$\sum_{t_1 \in T_1} \alpha_{1,t_1} \cdots \sum_{t_m \in T_m} \alpha_{m,t_m} \cdot c(t_1, \ldots, t_m) = u, \tag{12}$$

for some $\alpha_{j,t_j} \in \mathbb{F}$ and $u \in \mathbb{F}^{L_1 \times \cdots \times L_m}$. Indeed, think of a verifier that at the beginning sends uniformly random $\{\alpha_{j,t_j}\} \leftarrow \mathbb{F}$ and asks the prover to prove Eq. (12) with u that has a zero in the systematic part (which is a single coordinate over $\mathbb{F}$). If $x_T \equiv 0$, then all symbols in the sum have a zero in the systematic coordinate and, therefore, so has their sum. If $x_T \not\equiv 0$, then Eq. (12) holds with probability at most $1 - (1 - 1/|\mathbb{F}|)^m \approx m/|\mathbb{F}|$ over the choice of random coefficients and u. (Crucially to our final goal, we will later see how to sample good-enough coefficients using much less randomness.)

To prove Eq. (12), the prover and verifier engage in a sumcheck protocol. The sumcheck protocol we build on is an adaptation of standard sumcheck, specifically its generic form for general tensor codes from [Mei13]. It was shown in [RRR16] that, beyond the standard notion of soundness, the sumcheck protocol also satisfies unambiguity. We devise an analogous protocol for tree code tensors, keeping in mind the eventual goal of attaining an incremental proof oracle for the zero test.

6.1 Warm-up: Sumcheck for Tree Code Tensors

In our sumcheck protocol for tree code tensors, the verifier is given oracle access to a codeword $c \in \mathrm{TC}^m$ and the goal of the prover is to prove Eq. (12) holds with $\{\alpha_{j,t_j}\}$ that are given as input to both parties.

The protocol consists of m rounds of interaction where, at the j^{th} round, the sum over T_j is replaced by a sum over a small set $R_j \subset [n_j]$ of size $\lambda = (2H_n \log(m))/\delta'$ for $\delta' = \delta - \epsilon$, where $\epsilon > 0$

is an arbitrarily small constant. Concretely, we reduce a statement of the form

$$\sum_{r_1 \in R_1} \beta_{1,r_1} \cdots \sum_{r_{j-1} \in R_{j-1}} \beta_{j-1,r_{j-1}} \cdot \sum_{t_j \in T_j} \alpha_{j,t_j} \cdots \sum_{t_m \in T_m} \alpha_{m,t_m} \cdot c(r_1, \ldots, r_{j-1}, t_j, \ldots, t_m) = u_j \quad (13)$$

to a smaller statement of the form

$$\sum_{r_1 \in R_1} \beta_{1,r_1} \cdots \sum_{r_{j-1} \in R_{j-1}} \beta_{j-1,r_{j-1}} \cdot \sum_{r_j \in R_j} \beta_{j,r_j} \cdots \sum_{t_m \in T_m} \alpha_{m,t_m} \cdot c(r_1, \ldots, r_j, t_{j+1}, \ldots, t_m) = u_{j+1}, \quad (14)$$

We start with $u_1 = u$ and, after the m^{th} round, we are left with the following statement

$$\sum_{r_1 \in R_1} \beta_{1,r_1} \cdots \sum_{r_m \in R_m} \beta_{m,r_m} \cdot c(r_1, \ldots, r_m) = u_{m+1},$$

which V can verify with λ^m queries to c.[11]

We now describe the j^{th} round of the protocol. Recall, by this point, we have a statement of the form of Eq. (13) in hand. In particular, the value u_j, the sets $R_1, \ldots, R_{j-1}$ and the elements $\{\beta_{j,r_j}\}, \ldots, \{\beta_{j-1,r_{j-1}}\}$ have been already determined and are known to both parties.

1. P computes the truth table of the function $d_j : [n_j] \to \mathbb{F}^{L_1 \times \cdots \times L_m}$ defined as

$$d_j(t) = \sum_{r_1 \in R_1} \beta_{1,r_1} \cdots \sum_{r_{j-1} \in R_{j-1}} \beta_{j-1,r_{j-1}} \cdot \sum_{t_{j+1} \in T_{j+1}} \alpha_{j+1,t_{j+1}} \cdots \sum_{t_m \in T_m} \alpha_{m,t_m} \cdot c(r_1, \ldots, r_{j-1}, t, t_{j+1}, \ldots, t_m), \quad (15)$$

   and sends it to V. Denote the prover's message by $\overline{d}_j$.

2. V verifies that $\sum_{t \in T_j} \alpha_{j,t} \overline{d}_j(t) = u_j$ and, in the partition of $\overline{d}_j$ to $L' = \prod_{j' \neq j} L_{j'}$ functions $\overline{d}_j^1, \ldots, \overline{d}_j^{L'} : [n_j] \to \mathbb{F}^{L_j}$ such that $\overline{d}_j(t) = (\overline{d}_j^1(t), \ldots, \overline{d}_j^{L'}(t))$, any $\overline{d}_j$ is a codeword in TC.

3. V samples R_j as a subset of λ i.i.d. random coordinates $r \leftarrow \sigma_{n_j}$ (Definition 3.7) and, for any $r \in R_j$ samples uniform $\beta_{j,r} \leftarrow \mathbb{F}$ and sends it to P.

4. Proceed to prove the statement in Equation (14) w.r.t. R_j and $\{\beta_{j,r}\}$ chosen above and

$$u_{j+1} = \sum_{r \in R_j} \beta_{j,r} \cdot \overline{d}_j(r). \quad (16)$$

Completeness follows easily since, for any $j \in [m]$, an honest $\overline{d}_j = d_j$ is indeed a concatenation of codewords in TC due to the structure of tensor codes (Remark 4.2) and their linearity.

Soundness is by the fact that if Equation (13) does not hold at some round j, then Equation (14) holds only with low probability over the choice of the random R_j and β_{j,r_j} from Step 3 and u_j as defined in Equation (16). Given this, soundness follows by a union bound over $j = 1, \ldots, m$.

To see why the implication holds, notice that $\overline{d}_j$ is a (pointwise) concatenation of $L' = \prod_{j' \neq j} L_{j'}$ codewords of TC (otherwise, V rejects) and therefore the tree distance between $\overline{d}_j$ and d_j over one of the L' "slices" is at least δ_j. (The two columns are necessarily distinct over one of the

[11] Here lays the main difference from classical sumcheck, where $\lambda = 1$. Per-round amplification is needed here since, in probabilistically testing tree distance using the suffix distribution, we inherently loose a $H_n = \omega(1)$ factor (by the tightness of Lemma 3.8). This is in contrary to testing Hamming distance where the probability of disagreement at a uniformly random location is exactly the Hamming distance. Since we need the soundness error at any round to be small enough for a non-trivial union bound over the m rounds, $\omega(1)$ loss is unaffordable.

slices since, otherwise, V rejects at Step 2.) By Lemma 3.8, this implies that the suffix distance between d_j and $\overline{d}_j$ over the slice is at least $(\delta(n_j) - o(1))/H_{n_j}$. Hence, by definition, it holds that $\Pr[d_j(r) = \overline{d}_j(r)] \leq 1 - (\delta(n) - o(1))/H_n$ for a coordinate r sampled from the suffix distribution σ_{n_j}. Since R_j is a subset of λ i.i.d. such coordinates, it holds that

$$\Pr[d_j(R_j) = \overline{d}_j(R_j)] \leq (1 - (\delta(n) - o(1))/H_n)^{\lambda} < e^{-\delta'\lambda/H_n} < 1/m^2. \tag{17}$$

Now, assuming $d_j(R_j) \neq \overline{d}_j(R_j)$ and, hence, $d_\Delta = d_i(T_i) - \overline{d}_i(T_i) \not\equiv 0$, it holds that a uniform linear combination of the coordinates in d_Δ gives zero with probability at most $1/|\mathbb{F}|$. Therefore,

$$\Pr[\sum_{r_j \in R_j} \beta_{j,R_j} \cdot d_j(r_j) = u_{i+1}] = 1/|\mathbb{F}|.$$

By the above and Equation (17), we conclude that the soundness error in one round is less than $1/m^2 + 1/|\mathbb{F}|$ and, therefore, at most $1/m + m/|\mathbb{F}|$ overall, which is a constant smaller than 1.

6.2 The Zero Test Proof Oracle

The general idea for turning the interactive zero test into a proof oracle is to write down all possible transcripts of the protocol, corresponding to any configuration of the verifier's random coins. The size of the proof here grows exponentially with the randomness complexity of the verifier. In the interactive zero test, the verifier samples three types of challenges: the uniform coefficients $\{\alpha_{j,t_j}\}$ in the first round (before sumcheck) and the set of locations R_j and coefficients $\{\beta_{j,r_j}\}$ at round $j = 1, \ldots, m$ of the sumcheck. In a naive implementation, the number of all possible configurations these values can take is far bigger than what we can afford to write in our proof oracle. We are nevertheless able to obtain the desired proof size by two observations.

First, we observe that transcripts corresponding to different values of R_j and $\{\beta_{j,r_j}\}$ may be represented by a small basis of transcripts that span them linearly. Including the basis in the oracle is enough since the verifier may simulate access to any possible transcript by querying few locations across the basis. Second, we revisit the goal of the coefficients $\{\alpha_{j,t_j}\}$ and recall that they serve to reduce the statement $x_T \equiv 0$ to a sumcheck statement. Using uniformly random coefficients is wasteful here. What we essentially want is a relatively small set of vectors $\{\boldsymbol{\alpha}_j = (\alpha_{j,1}, \ldots, \alpha_{j,|T_j|})\}$ such that $\Pr_j[\boldsymbol{\alpha}_j \cdot x' = 0]$ is small for all $x' \neq 0^{|T_j|}$. This is equivalent to the existence of linear error-correction codes with good rate and distance. Letting the $\boldsymbol{\alpha}_j$'s be the rows of a generator matrix of a block error-correction code with rate ρ and relative Hamming distance δ, gives a set of size $|T_j|/\rho$ and probability of $1 - \delta$ to "miss a zero". (Conversely, a good derandomization set implies a good error-correction code.) This solution could have worked for us, except we do not know how to directly use it to obtain a monotone proof oracle. Not surprisingly, this can be made possible by replacing the block error-correction code with a tree code that has an explicit efficient encoding function.

We now formally describe the prescribed proof oracle $\pi = \pi_{c,T}$, for $c \in \mathrm{TC}^m$ and rectangle $T = T_1 \times \cdots \times T_m$ that satisfy $x_T \equiv 0$. Let $G = \{G_n \in \mathbb{F}^{(\hat{L}(n)\cdot n)\times n}\}$ be a (block lower-triangular) generator matrix of $\widehat{\mathrm{TC}}$ (Remark 3.6) and view it as $G_n : [n]^2 \to \mathbb{F}^{\hat{L}(n)}$ by splitting its columns into $\hat{L}(n)$-size blocks (that correspond to different symbols in a codeword). Let $t_j^1 < t_j^2 < \cdots < t_j^{|T_j|}$ denote the elements of T_j. The prescribed proof π is composed by oracles $\pi_1, \ldots, \pi_{m-1}$ where, for

$j = 1, \ldots, m-1$, (letting $\hat{L}_j = \hat{L}(|T_j|)$)

$$\pi_j : [n_1] \times \cdots \times [n_j] \times [|T_{j+1}|] \times \cdots \times [|T_m|] \to \mathbb{F}^{\hat{L}_{j+1} \times \cdots \times \hat{L}_m \times L_1 \times \cdots \times L_m},$$

$$\pi_j(t_1, \ldots, t_j, i_{j+1}, \ldots, i_m) = \sum_{k_{j+1}=1}^{|T_{j+1}|} G(i_{j+1}, k_{j+1}) \otimes \cdots \otimes \sum_{k_m=1}^{|T_m|} G(i_m, k_m) \otimes c(t_1, \ldots, t_j, t_{j+1}^{k_{j+1}}, \ldots, t_m^{k_m}). \tag{18}$$

Notice that $\pi_m = c$. The total size of the proof π is bounded by $m \cdot (\prod_{j=1}^{m} L_j \hat{L}_j) n_j$ field elements. To see why the construction satisfies robust incrementality, observe that at any π_j is essentially an encoding of x by a tensor tree code. For $j = 1, \ldots, m$, let TC'_j denote the tree code that on input message $y : [n] \to \mathbb{F}^{L_j}$, slices it into L_j messages $y_1, \ldots, y_{L_j} : [n] \to \mathbb{F}$ and applies the tree code $\widehat{\mathrm{TC}}$ over the restriction of any y_r to T_j to obtain $w_r : [|T_j|] \to \mathbb{F}^{\hat{L}_j}$ and their (point-wise) concatenation $w : [|T_j|] \to \mathbb{F}^{\hat{L}_j \times L_j}$. It holds that $w(i) = \sum_{k=1}^{|T_j|} G(i,k) \otimes y(t_j^k)$. We may write, then

$$\pi_j = \left(I^j \otimes \mathrm{TC}'_{j+1} \otimes \cdots \otimes \mathrm{TC}'_m\right)(c). \tag{19}$$

While Definition 4.1 refers to m-fold tensor products that involve the same code, the definition straight-forwardly generalizes to the tensor product of different tree codes, exhibiting similar algebraic and combinatorial structural properties (Remark 4.2). In particular, such a tensor has a robustly incremental encoding function when the base codes are locally correctable, by direct generalization of Proposition 4.12: For $\pi = (\pi_1, \ldots, \pi_{m-1})$ and $\pi' = (\pi'_1, \ldots, \pi'_{m-1})$, we define $\Delta(\pi, \pi')$ to be the maximum of $\Delta_{\mathsf{S}}(\pi_j, \pi'_j)$ over all $j = 1, \ldots, m-1$. Then, assuming $\widehat{\mathrm{TC}}$, and therefore TC', is $(\Delta_{\mathsf{S}}, q, \delta'')$-LCC, it holds that the prescribed proof oracle $\pi_{c,T}$ is robustly incremental with the parameters in the lemma.

We next describe the verifier. In fact, the following is a "base verifier" that suffers from large soundness error. We amplify soundness by repeating the base verification $\lambda' = O((H_n/\delta(n))^{3m+2})$ times with i.i.d. randomness.

The verifier views its oracle as a composition of m oracles $(\pi, c) = (\pi_1, \ldots, \pi_m)$. For $j = 1, \ldots, m$, the verifier samples $i_j \leftarrow \sigma_{|T_j|}$, a random subset R_j that consists of λ i.i.d. $r \leftarrow \sigma_{n_j}$ and uniformly random coefficients $(\beta_{j,r})_{r \in R_j} \leftarrow \mathbb{F}^{\lambda}$. Let $\overline{d}_j$ denote the following function over $[n_j]$:

$$\overline{d}_j(t) = \sum_{r_1 \in R_1} \beta_{1,r_1} \cdots \sum_{r_{j-1} \in R_{j-1}} \beta_{j-1,r_{j-1}} \cdot \pi_j(r_1, \ldots, r_{j-1}, t, i_{j+1}, \ldots, i_m).$$

Note that the verifier can read $\overline{d}_j$ entirely using $\lambda^{j-1} n_j$ queries to π_j and, in particular, can read $\overline{d}_m$ using $\lambda^{m-1} n_m$ queries to c. The verifier accepts if and only if all of the following hold:

(i) For $j = 1, \ldots, m-1$, $\overline{d}_j$ is a concatenation of $\prod_{j'<j} L_j \prod_{j'>j} \hat{L}_j L_j$ codewords in TC (akin to Step 2 in the interactive sumcheck).

(ii) For $j = 2, \ldots, m$,

$$\sum_{k=1}^{|T_j|} G(i_j, k) \otimes \overline{d}_j(t_j^k) = \sum_{r \in R_{j-1}} \beta_{j-1,r} \cdot \overline{d}_{j-1}(r). \tag{20}$$

(iii) Letting

$$u = \sum_{k=1}^{|T_j|} G(i_1, k) \otimes \overline{d}_1(t_j^k) \quad \in \mathbb{F}^{\hat{L}_1 \times \cdots \times \hat{L}_m \times L_1 \times \cdots \times L_m}, \tag{21}$$

that u has $0^{\hat{L}_1 \times \cdots \times \hat{L}_m}$ in the systematic coordinate of TC^m.

In any repetition of the above verification, the verifier reads at most $\sum_{j=1}^{m} \lambda^{j-1} n_j$ locations from its oracles c and π. The query complexity of λ' repetitions is then $O\Big(m\lambda' \cdot \lambda^m \cdot (n_1 + \cdots + n_m)\Big) = O\Big((m \log m) \cdot (H_n/\delta(n))^{4m+2} \cdot (n_1 + \cdots + n_m)\Big)$.

Standard completeness follows by inspection. Conditions (i) and (ii) hold with the prescribed proofs by construction. For (iii), the main observation is that the systematic coordinate in u from Eq. (21) is the $(i_1, \ldots, i_m)^{th}$ symbol in the $(\mathrm{TC}')^m$-codeword that encodes the systematic coordinates of $c(T)$, which contain precisely x_T (Eq. (19)). Hence, if $x_T \equiv 0$, u must contain a zero symbol.

Moreover, by the distance of TC', if $x_T \not\equiv 0$, then the systematic coordinate in u is a symbol in a $(\mathrm{TC}')^m$-codeword that has high suffix weight at least $(\delta(n)/H_n)^m$ (Proposition 4.4). This implies prescribed completeness: Since $(i_1, \ldots, i_m)$ is sampled from the suffix distribution $\sigma_{|T_1|} \times \cdots \times \sigma_{|T_m|}$, the probability that u contains a zero is at most $1-(\delta(n)/H_n)^m$. Consequently, condition (iii) holds in all of the λ' repetitions with probability at most $(1-(\delta(n)/H_n)^m)^{\lambda'} = n^{-\omega(1)}$.

For unambiguity, assume the verifier is given access to an oracle $\pi' = (\pi'_1, \ldots, \pi'_{m-1})$ and define $\pi'_m = \pi_m = c$. Observe that, if the verifier accepts with non-negligible probability, then, for any $j = 1, \ldots, m-1$:

(a) Most lines ("in suffix weight") in π'_j that are parallel to the j^{th} axis are codewords in TC. Formally, letting $r_{j'} \leftarrow \sigma_{n_{j'}}$, for $j' < j$, and $i_{j'} \leftarrow \sigma_{|T_{j'}|}$, it holds that $\pi'_j(r_1, \ldots, r_{j-1}, \cdot, i_{j+1}, \ldots, i_m)$ is a concatenation of codewords in TC, with probability at least $1-(\delta(n)/H_n)^{3m}$. Otherwise, the verifier catches a non-codeword in condition (i) in one of the λ' repetitions with probability all but negligible.

(b) $\Delta_{\mathsf{S}}((I^j \otimes \mathrm{TC}' \otimes I^{m-j-1})(\pi'_{j+1}), \pi'_j) < (\delta(n)/H_n)^{3m}$. Otherwise, condition (ii) fails to hold in one of λ' repetitions with probability all but negligible.

We argue that $D(j) := \Delta_{\mathsf{S}}(\pi_j, \pi'_j) = O((\delta(n)/H_n)^{2m})$ for all j by induction. It holds that $D(m-1) < (\delta(n)/H_n)^{2m}$ by (b) – recall $\pi'_m = c$ and $\pi_{m-1} = (I^{m-1} \otimes \mathrm{TC}')(c)$. For smaller j, by an averaging argument over suffix distance (Definition 3.7), we have that π'_{j+1} is with high probability close to π_{j+1} over a random axis-parallel line sampled by suffix distribution. In particular,

$$\Pr_{r_{j'} \leftarrow \sigma_{n_{j'}}, i_{j'} \leftarrow \sigma_{|T_{j'}|}} \Big[\Delta_{\mathsf{S}}\left(\pi_{j+1}(r_1, \ldots, r_j, \cdot, i_{j+2}, \ldots, i_m),\ \pi'_{j+1}(r_1, \ldots, r_j, \cdot, i_{j+2}, \ldots, i_m)\right) < \delta(n)/H_n\Big] > 1 - (H_n/\delta(n)) \cdot D(j+1).$$

By the minimum distance of TC, if such a line in π'_{j+1} happens to be also a codeword in TC, then it must be identical to the corresponding line in π_{j+1}. By the above and (a), then,

$$\Pr_{r_{j'} \leftarrow \sigma_{n_{j'}}, i_{j'} \leftarrow \sigma_{|T_{j'}|}} \Big[\pi_{j+1}(r_1, \ldots, r_j, \cdot, i_{j+2}, \ldots, i_m) \equiv \pi'_{j+1}(r_1, \ldots, r_j, \cdot, i_{j+2}, \ldots, i_m)\Big] > 1 - (H_n/\delta(n)) \cdot D(j+1) - (\delta(n)/H_n)^{3m}.$$

It follows that the encoding of such two identical lines under TC' is identical and, therefore,

$$\Delta_{\mathsf{S}}\left((I^j \otimes \mathrm{TC}' \otimes I^{m-j-1})(\pi_{j+1}), (I^j \otimes \mathrm{TC}' \otimes I^{m-j-1})(\pi'_{j+1})\right) < (H_n/\delta(n)) \cdot D(j+1) + (\delta(n)/H_n)^{3m}.$$

This implies $D(j) < (H_n/\delta(n)) \cdot D(j+1) + 2(\delta(n)/H_n)^{3m} = O((\delta(n)/H_n)^{2m})$ by triangle inequality since $(I^j \otimes \mathrm{TC}' \otimes I^{m-j-1})(\pi_{j+1}) = \pi_j$ and $(I^j \otimes \mathrm{TC}' \otimes I^{m-j-1})(\pi'_{j+1})$ is $(\delta(n)/H_n)^{3m}$-close to π'_j by (b).

7 The Tree PCP

We put together the components from Sections 4 to 6 to construct a tree PCP and prove our main result from Theorem 1.5.

Let $(C, a, n) \in$ CktReach be a circuit reachability instance (Eq. (1)). We describe an accepting proof for (C, a, n), which can be efficiently computed given a witness $(a_1, w_1, \ldots, a_n, w_n)$ such that $C(a_{t-1}, a_t, w_t) = 1$ for all $1 \le t \le n$ (where $a_0 = 0^s$), and also satisfies monotonicity. Then, we describe the PCP verifier and prove its soundness.

The tree PCP is parametrized by a constant $m \in \mathbb{N}$ which can be arbitrarily increased to reduce asymptotic complexity. In particular, for sublinear complexity (queries and proof length per step), we shall choose $m \ge 5$.

In the construction below, we use TC to denote the tree code from Corollary 4.17 with a small enough constant parameter $0 < \mu < 1$, and recall its extension TC^+ (Definition 5.10) and their tensor products TC^m, $(\mathrm{TC}^+)^m$ and their flattening $\overline{\mathrm{TC}^m}$, $\overline{(\mathrm{TC}^+)^m}$ (Definitions 4.1 and 4.6 and Corollary 4.21).

7.1 The Proof Oracle

Let $A_t = (a_{t-1}, a_t, w_t) \in \{0,1\}^{3s}$ and denote by $A^i : [n] \to \mathbb{F}$ the column satisfying $A^i(t) = A_t(i)$ for all $t \in [n]$. Let $W = \{W^1, \ldots, W^K\}$ be the set that contains the witnesses for $\mathbf{Eval}(C)$ corresponding to $A^1, \ldots, A^{3s}$ (induced by Lemma 5.1) and, for $i = 1, \ldots, s$, the witnesses for $\mathbf{EQ}(n)$ corresponding to the pair A^i and A^{i+s} (Lemma 5.8). It holds that $K = \mathrm{poly}(|C|) + s \cdot \mathrm{polylog}(n)$. We denote by $N = O(n)$ the upper bound on the length of any W^i and assume for simplicity and w.l.o.g. that $W^i : [N] \to \mathbb{F}$ for all i (smaller witnesses can be padded with zeros).

An accepting proof π for (C, a, n) consists of the following:

1. For $i = 1, \ldots, 3s$ and $j = 1, \ldots, K$,
$$\widetilde{A}^i = \overline{\mathrm{TC}^m}(A^i) \qquad\qquad \widetilde{W}^j = \overline{(\mathrm{TC}^+)^m}(W^j).$$

2. For any constraint $P \in \mathbf{Eval}(C)$, letting[12] $E = P(A^1, \ldots, A^{3s}, W) \in \mathbb{F}^I$ and $\widetilde{E} \in (\mathrm{TC}')^m(E)$ be obtained by the codeword evaluation in Lemma 5.1, an accepting zero-test proof for $(\widetilde{E}, I)$ (Lemma 6.1), namely $\pi_{\widetilde{E},I}$ from the lemma.

3. For any constraint $(P, R) \in \mathbf{EQ}(n)$ and any $i = 1, \ldots, s$, letting[12] $E = P(A^i, A^{i+s}, W) \in \mathbb{F}^I$ and $\widetilde{E} \in (\mathrm{TC}')^m(E)$ be obtained by the codeword evaluation in Lemma 5.8, an accepting zero-test proof for $(\widetilde{E}, R)$, namely $\pi_{\widetilde{E},R}$. We use $\widehat{\mathrm{TC}} = \mathrm{TC}$ for the zero tests.

Overall, the PCP for (C, a, n) consists of: $3s \cdot K = \mathrm{poly}(\log(n), |C|)$ codewords in $\overline{\mathrm{TC}^m}$ of length $O(n)$ over an alphabet of size $O(n^{\mu+o(1)})$, and $\mathrm{poly}(\log(n), |C|)$ zero-test proof oracles over statements of overall size $O(n^{1+2\mu+o(1)})$ (there are $\mathrm{poly}(\log(n), |C|)$ constraints in $\mathbf{Eval}(C) \cup \mathbf{EQ}(n)$ in total). Its total length, then, is $n^{1+2\mu+o(1)} \cdot \mathrm{poly}(\log(n), |C|)$.

The PCP is also robustly incremental in an amortized sense: The proof for (C, a_n, n) can be generated by computing a proof for $(C, a_{n-1}, n-1)$ then extending it, and this takes time $O(n^{1+\gamma} \cdot \mathrm{poly}(|C|))$ in total. This is due to the incrementality of the witnesses W^i and the evaluation vectors E (Lemmas 5.1 and 5.8), and the robust incrementality of the zero tests (Lemma 6.1, given the local correctability of TC from Lemma 4.20) and of tensor tree codes (Proposition 4.12).

[12] More accurately, P takes only a subset of W as input.

In fact, the only reason the tree PCP does not attain incrementality in the strict sense of Theorem 1.5 is that the witnesses W^i and the evaluation vectors E, corresponding to **EQ**, are not incremental in the assignments. If they were, we would obtain a properly incremental tree PCP by the incrementality of all other components.

While W^i, E from **EQ** are not incremental, they are monotone and efficient to compute given the assignments (Lemma 5.8): Every new symbol can be computed by accessing a single coordinate in $A^1, \ldots, A^{3s}$. The only problem is that many symbols may be added to any such W^i, E at once, upon appending a single new value to each of $A^1, \ldots, A^{3s}$. Indeed, their length is only proportional to n and does not match it exactly. If we manage to de-amortize the extension of these W^i, E and extend them by, say, $O(1)$ new symbols at every step, then we can obtain an $(n^{\gamma/m} \cdot \mathrm{poly}(\log n, |C|), n^{\mu+o(1)}\mathrm{poly}(|C|))$-incremental PCP for a constant γ independent in m, matching the theorem. In Section 7.3, we show how to de-amortize any W^i and E and, therefore, the tree PCP construction.

7.2 The Verifier

Given an input (C, a, n) and access to a proof π, the verifier V performs the following:

1. Perform the local test T from Proposition 4.9 over $\widetilde{A}^i$, for all $i = 1, \ldots, 3s$, and $\widetilde{W}^j$, for all $j = 1, \ldots, K$, $\lambda = \log^{(1+1/\mu)m+3}(n)$ times each.
   If any of the tests rejects, the verifier rejects. Otherwise, the verifier runs the following while simulating any query to any $F \in \{\widetilde{A}^i, \widetilde{W}^j\}$ using the (relaxed) local corrector $\mathcal{C}$ from (Lemma 4.11 or Lemma A.1) over F_I, for some $I \in I(|F|)$ that contains the queried location. If $\mathcal{C}$ outputs $\perp$ at any of its invocations, the verifier rejects.

2. For $i = 1, \ldots, s$, check that the first symbol in the message encoded by $\widetilde{A}^i$ is 0 (by reading the systematic part in the first symbol). For $i = s+1, \ldots, 2s$, check that the last symbol in the message encoded by $\widetilde{A}^{s+i}$ is $a(i)$. If either conditiones do not hold, reject.

3. For all $t \in \mathbf{Bad}(n)$ (Lemma 5.8) and all $i = 1, \ldots, s$, check that the t^{th} symbol in the message encoded by $\widetilde{A}^i$ is equal to the $(t-1)^{th}$ symbol in the message encoded by $\widetilde{A}^{s+i}$. (Again, this is done by reading the systematic parts in the respective codeword symbols.)

4. For any zero-test proof oracle $\pi_{\widetilde{E},R}$ in the PCP, perform the zero test from Lemma 6.1 to verify the statement $E_R \equiv 0$.
   To read from the codeword $\widetilde{E}$ that encodes the evaluation of some constraint P from $\mathbf{Eval}(C)$ or $\mathrm{EQ}(n)$ the assignments, the verifier uses the evaluation algorithm from Lemmas 5.1 and 5.8.
   If any of the zero tests rejects, the verifier rejects. Otherwise, the verifier accepts.

Query Complexity. In Step 1, the verifier performs $\lambda \cdot (3s + K) = \mathrm{poly}(\log(n), |C|)$ local tests over alleged codewords of length $O(n)$, that are over alphabet of size $n^{\mu+o(1)}$. By Proposition 4.9, each of the tests has query complexity $n^{2/m} \cdot \mathrm{polylog}(n)$ codeword symbols. In analyzing the complexity of the remaining steps, we shall take into account an overhead of $n^{1/m} \cdot \mathrm{polylog}(n)$ due to the use of the local corrector $\mathcal{C}$ to simulate queries to the codewords (recall, by Proposition 4.8, the length of any rectangle $I \subseteq I(n)$ does not exceed $\lceil n^{1/m} \rceil$ at any dimension).

In Steps 2 and 3, $O(s)$ field elements are read in total. In Step 4, the verifier performs $\mathrm{poly}(\log(n), |C|)$ zero tests to verify statements of size $O(n)$. By Lemma 6.1, each such test has query complexity at most $n^{1/m} \cdot \mathrm{polylog}(n)$.

Completeness. Completeness of V follows by inspection from the completeness of the local test (Proposition 4.9 and Corollary 4.21), the local corrector (Lemma 4.11 or Lemma A.1), the constraints $\mathbf{Eval}(n, C)$ (Lemma 5.1) and $\mathbf{EQ}(n)$ (Lemma 5.8) and the zero tests (Lemma 6.1).

Soundness. Soundness also follows from the respective soundness guarantees of these components: First, we argue that if, with non-negligible probability, all local tests over some $\widetilde{F} \in \{\widetilde{A^i}, \widetilde{W^j}\}$ in Step 1 accept then $\widetilde{F}$ is $(\delta/2H_n)^m$-close to its respective code ($\overline{\mathrm{TC}^m}$ ot $\overline{(\mathrm{TC}^+)^m}$) in $\overline{\Delta_{\mathsf{S}}}$ (Proposition 4.9) and $\delta = \Omega(1/\log^{1/\mu}(n))$ is the tree distance of TC (Corollary 4.17). Assuming the contrary, a local test over $\widetilde{F}$ rejects with probability at least $\epsilon = \Omega(1/\log^{(1+1/\mu)m+1}(n))$ by Proposition 4.9. The probability that $\lambda = \log^{(1+1/\mu)m+3}(n)$ such tests accept is then at most $(1-\epsilon)^\lambda = e^{-\epsilon\lambda} = n^{-\Omega(\log(n))}$.

Assuming the local tests pass with non-negligible probability, for any $\widetilde{F} \in \{\widetilde{A^i}, \widetilde{W^j}\}$, we have $\overline{\Delta_{\mathsf{S}}}(\widetilde{F}, \overline{\mathrm{TC}}^m) < (\delta/2H_n)^m$ and by definition there exists F such that, for any I, $\Delta_{\mathsf{S}}(\widetilde{F_I}, \mathrm{TC}^m(F_I)) < (\delta/2H_n)^m$. Therefore, with probability all but negligible, the local corrector $\mathcal{C}$, when invoked over $\widetilde{F_I}$, either outputs $\perp$ (in which case V rejects) or simulates access to $\mathrm{TC}^m(F_I)$ (Lemma A.1). Hence, we may assume that in Steps 2 to 4, the oracles $\widetilde{A^1}, \ldots, \widetilde{A^{3s}}, \widetilde{W^1}, \ldots, \widetilde{W^K}$ that the verifier reads from are indeed codewords in $\overline{\mathrm{TC}}^m$, respectively $\overline{\mathrm{TC}^+}^m$, that encode some $A^1, \ldots, A^{3s}, W^1, \ldots, W^K$ (and correspond to the unique closest codewords to the actual oracles given in the proof).

If $(C, a, n) \notin$ CktReach, then either:

1. $A^i(1) \neq 0$ for some $i \in \{1, \ldots, s\}$, or $A^i(n) \neq a(i)$ for some $i \in \{s+1, \ldots, 2s\}$, or
2. $A^i(t) \neq A^{i+s}(t-1)$ for some $i \in \{s+1, \ldots, 2s\}$ and $1 < t \leq n$, or
3. $A^1, \ldots, A^{3s}$ are not binary or $C(A^1(t), \ldots, A^{3s}(t)) \neq 1$ for some $1 \leq t \leq n$.

If the first condition holds, then V rejects already in Step 2. If the second condition holds with $t \in \mathbf{Bad}(n)$, then V rejects in Step 3. If it holds with $t \notin \mathbf{Bad}(n)$ or the third condition holds then, by Lemmas 5.1 and 5.8 (resp.), there exists $(P, R) \in \mathbf{Eval}(C) \cup \mathbf{EQ}(n)$ such that $E = P(A^1, \ldots, A^{3s}, W) \in \mathbb{F}^I$ satisfies $E_R \not\equiv 0$ (in the constraints from $\mathbf{Eval}(C)$, we have $R = I$). We claim that the zero-test from Step 4, corresponding to the broken constraint, rejects with probability all but negligible.

Soundness of the tree PCP then follows from the zero soundness of the zero tests (Lemma 6.1) and the correctness of the codeword evaluation algorithms (Lemmas 5.1 and 5.8).

7.3 De-Amortization

To finish the proof of Theorem 1.5, we explain how to de-amortize the extension of any witness W^i for $\mathbf{EQ}$. Identical techniques apply for de-amortizing the evaluation vector $E = P(A, A', W^1, \ldots, W^K)$ for any $(P, R) \in \mathbf{EQ}$.

The witnesses $W^1, \ldots, W^K$ from Lemma 5.8 come in four types: B_D for some $D \subseteq [m]$ (1), $G_{i_1,\ldots,i_m}$ for $i_j \leq \lceil \log n_j \rceil$, where $n_j = \max_{(t_1,\ldots,t_m)\in I(n)}(t_j)$ (2), T^r for $r \in [M]$ (3) or their corresponding witnesses for $\mathbf{Eval}(C_f)$ (4).

The easiest to amortize are T^r for $r = 1, \ldots, M$. Recall that these encode the binary representation of the coordinates in the domain and do not depend on the assignments. They are defined over $[N]^m$, where N is the smallest power of 2 satisfying $I(n) \subseteq [N]^m$. It holds that $N \leq 2\lceil n^{1/m} \rceil$ (Proposition 4.8) and hence $N^m \leq 2^m n$. Therefore, we can de-amortize any T^r by extending it by 2^m at every step following, for example, the embedding φ (Fig. 4). Consequently, by Lemma 5.1, we can de-amortize the witnesses for $\mathbf{Eval}(C_f)$ corresponding to $T^1, \ldots, T^M$.

Straight-forward de-amortization applies also for any B_D. There, the domain is rounded up to the next power of 2 only along dimensions $j \in D$. Upon extending the corresponding assignment A with the n^{th} value, letting $(n_1, \ldots, n_m) = \varphi(n)$, we extend B_D to any coordinate $(t_1, \ldots, t_m)$ where $t_j = n_j$ for $j \notin D$ and $t_j \in \{2n_j - 1, 2n_j\}$ for $j \in D$. Crucially, these coordinates in B_D, $2^{|D|} \leq 2^m$ in total, are already defined at this point given A.

Lastly, we show how to de-amortize any $G_{i_1,\ldots,i_m}$. Recall $G_{i_1,\ldots,i_m}$ is defined over $I^G_{i_1,\ldots,i_m}(n)$ which is the union of all rectangles $[n_1(i_1)] \times \cdots \times [n_m(i_m)]$ where $[n_1] \times \cdots \times [n_m] \in \mathbf{Rect}(n)$ and $n_j(i) = 2^i \left\lfloor n/2^i \right\rceil$.

Upon input a new value at coordinate $(t_1, \ldots, t_m)$ in the "lifting" of A to m dimension (via φ), where $t_j = k_j \cdot 2^{i_j} + t'_j$ for some k_j and $1 \leq t'_j \leq 2^{i_j - 1}$, we extend $G_{i_1,\ldots,i_m}$ by computing the symbols at coordinates $(k_1 \cdot 2^{i_1} + t''_1, \ldots, k_m \cdot 2^{i_m} + t''_m)$ for $t''_j \in \{2t'_j - 1, 2t'_j\}$.

On a close observation, this shows, for any rectangle $I = [n_1] \times \cdots \times [n_m] \subseteq I(n)$, how to incrementally compute $G_{i_1,\ldots,i_m}$ at $I^G = [n_1(i_1)] \times \cdots \times [n_m(i_m)]$, $O(1)$ symbols at a time, whenever A_I extends by one symbol. This seems to achieve what we want except it is not clear how to carry on the computation given that $G_{i_1,\ldots,i_m}$ at $(k_1 \cdot 2^{i_1} + t''_1, \ldots, k_m \cdot 2^{i_m} + t''_m)$ is not always fully defined given A up to coordinate t. Specifically, in any such block of size $2^{i_1} \times \cdots \times 2^{i_m}$ (defined by fixing $k_1, \ldots, k_m$), there exists exactly one coordinate t^* where $G_{i_1,\ldots,i_m}$ is not always zero and takes the value at the same location in A (this is the coordinate in $\Lambda_{i_1} \times \cdots \times \Lambda_{i_m}$, see Eq. (10)). Since this value is unknown by the time it must be added to $G_{i_1,\ldots,i_m}$ in the above incremental procedure, we let the proof contain the two possible versions of $G_{i_1,\ldots,i_m}$ corresponding to the two different Boolean values it might take at t^*. By the time t^*, when the value becomes known, the computation of $G_{i_1,\ldots,i_m}$ for this block is finished, the proof "consolidates" the correct version (by standard pointer techniques), and proceeds. Importantly, any part of $G_{i_1,\ldots,i_m}$ is contained in I^G only after it is consolidated.

One last issue is that, in a naive implementation of the above, the proof is extended by an infinite amount of symbols since there are infinitely many oracles $G_{i_1,\ldots,i_m}$ which we add to. Notice, however, that $G_{i_1,\ldots,i_m}$ is all-zeros over $[2^{i_1-1}] \times \cdots \times [2^{i_m-1}]$ and, due to the linearity of the tree code, so is its corresponding codeword. Hence, values in $G_{i_1,\ldots,i_m}$ and its codeword are not actually written to the proof before the walk φ reaches a coordinate $(n_1, \ldots, n_m)$ where $n_j > 2^{i_1-1}$, i.e. $i_j < \log(n_j) + 1$.

Acknowledgments

We thank Salil Vadhan and Siu On Chan for insightful discussions and Nikolaj Schwartzbach for advice on exposition. We additionally thank Rafail Ostrovsky and Daniel Wichs for pointing out the amortized tree PCP construction and Omer Paneth for drawing our attention to related work on incrementally verifiable computation.

References

[ALM+92] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. In *33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, Pennsylvania, USA, 24-27 October 1992*, pages 14–23. IEEE Computer Society, 1992.

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788. Springer, 2018.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 111–120. ACM, 2013.

[BCG24] Annalisa Barbara, Alessandro Chiesa, and Ziyi Guan. Relativized succinct arguments in the ROM do not exist. Cryptology ePrint Archive, Paper 2024/728, 2024.

[BCN21] Inbar Ben Yaacov, Gil Cohen, and Anand Kumar Narayanan. Candidate tree codes via pascal determinant cubes. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021, August 16-18, 2021, University of Washington, Seattle, Washington, USA (Virtual Conference)*, volume 207 of *LIPIcs*, pages 54:1–54:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[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*, pages 16–25 vol.1, 1990.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, STOC '91, page 21–32, New York, NY, USA, 1991. Association for Computing Machinery.

[BGH+06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006.

[BLR93] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. Comput. Syst. Sci.*, 47(3):549–595, 1993.

[BS04] Eli Ben-Sasson and Madhu Sudan. Robust locally testable codes and products of codes. In Klaus Jansen, Sanjeev Khanna, José D. P. Rolim, and Dana Ron, editors, *Approximation, Randomization, and Combinatorial Optimization, Algorithms and Techniques, 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2004, and 8th International Workshop on Randomization and Computation, RANDOM 2004, Cambridge, MA, USA, August 22-24, 2004, Proceedings*, volume 3122 of *Lecture Notes in Computer Science*, pages 286–297. Springer, 2004.

[BS08] Eli Ben-Sasson and Madhu Sudan. Short PCPs with polylog query complexity. *SIAM J. Comput.*, 38(2):551–607, 2008.

[CHK+19] Arka Rai Choudhuri, Pavel Hubácek, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, and Guy N. Rothblum. Finding a nash equilibrium is no easier than breaking fiat-shamir. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1103–1114. ACM, 2019.

[CHS18] Gil Cohen, Bernhard Haeupler, and Leonard J. Schulman. Explicit binary tree codes with polylogarithmic size alphabet. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 535–544. ACM, 2018.

[CKO14] Nishanth Chandran, Bhavana Kanukurthi, and Rafail Ostrovsky. Locally updatable and locally decodable codes. In Yehuda Lindell, editor, *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*, volume 8349 of *Lecture Notes in Computer Science*, pages 489–514. Springer, 2014.

[CL20] Alessandro Chiesa and Siqi Liu. On the impossibility of probabilistic proofs in relativized worlds. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, volume 151, pages 57:1–57:30, 2020.

[DGKV22] Lalita Devadas, Rishab Goyal, Yael Kalai, and Vinod Vaikuntanathan. Rate-1 non-interactive arguments for batch-np and applications. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 1057–1068. IEEE, 2022.

[DGMV20] Nico Döttling, Sanjam Garg, Giulio Malavolta, and Prashant Nalini Vasudevan. Tight verifiable delay functions. In *International Conference on Security and Cryptography for Networks*, pages 65–84. Springer, 2020.

[Din07] Irit Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3):12, 2007.

[EFKP20] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Sparks: Succinct parallelizable arguments of knowledge. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 707–737. Springer, 2020.

[FGL+96] Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *J. ACM*, 43(2):268–292, 1996.

[Gel17] Ran Gelles. Coding for interactive communication: A survey. *Found. Trends Theor. Comput. Sci.*, 13(1-2):1–157, 2017.

[GRR20] Tom Gur, Govind Ramnarayan, and Ron Rothblum. Relaxed locally correctable codes. *Theory Comput.*, 16:1–68, 2020.

[GS92] Peter Gemmell and Madhu Sudan. Highly resilient correctors for polynomials. *Inf. Process. Lett.*, 43(4):169–174, September 1992.

[GS06] Oded Goldreich and Madhu Sudan. Locally testable codes and PCPs of almost-linear length. *J. ACM*, 53(4):558–655, 2006.

[HN23] Mathias Hall-Andersen and Jesper Buus Nielsen. On valiant's conjecture - impossibility of incrementally verifiable computation from random oracles. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 438–469. Springer, 2023.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 723–732. ACM, 1992.

[KMRS17] Swastik Kopparty, Or Meir, Noga Ron-Zewi, and Shubhangi Saraf. High-rate locally correctable and locally testable codes with sub-polynomial query complexity. *J. ACM*, 64(2):11:1–11:42, 2017.

[KT00] Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 80–86, 2000.

[LFKN92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, October 1992.

[Mei09] Or Meir. Combinatorial construction of locally testable codes. *SIAM J. Comput.*, 39(2):491–544, 2009.

[Mei13] Or Meir. IP = PSPACE using error-correcting codes. *SIAM J. Comput.*, 42(1):380–403, 2013.

[Mic95] Silvio Micali. Computationally-sound proofs. In Johann A. Makowsky and Elena V. Ravve, editors, *Proceedings of the Annual European Summer Meeting of the Association of Symbolic Logic, Logic Colloquium 1995, Haifa, Israel, August 9-18, 1995*, volume 11 of *Lecture Notes in Logic*, pages 214–268. Springer, 1995.

[MRR25] Tamer Mour, Alon Rosen, and Ron Rothblum. Locally testable tree codes. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 5523–5559. SIAM, 2025.

[MS14] Cristopher Moore and Leonard J. Schulman. Tree codes and a conjecture on exponential sums. In Moni Naor, editor, *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 145–154. ACM, 2014.

[NPR19] Moni Naor, Omer Paneth, and Guy N. Rothblum. Incrementally verifiable computation via incremental PCPs. In Dennis Hofheinz and Alon Rosen, editors, *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 552–576. Springer, 2019.

[OW25] Rafail Ostrovsky and Daniel Wichs. Personal communication at the Proofs workshop at Simons Institute, July 14-18, 2025.

[PP22] Omer Paneth and Rafael Pass. Incrementally verifiable computation via rate-1 batch arguments. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 1045–1056. IEEE, 2022.

[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, page 194–203, New York, NY, USA, 1994. Association for Computing Machinery.

[Pud13] Pavel Pudlák. Linear tree codes and the problem of explicit constructions. *CoRR*, abs/1310.5684, 2013.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, page 49–62, New York, NY, USA, 2016. Association for Computing Machinery.

[RS96] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996.

[Sch93] Leonard J. Schulman. Deterministic coding for interactive communication. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 747–756. ACM, 1993.

[Sch94] Leonard J. Schulman. Postscript from 21 september 2003 to "coding for interactive communication", 1994. Online at http://www.cs.caltech.edu/~schulman/Papers/intercodingpostscript.txt.

[Spi95] Daniel Alan Spielman. *Computationally efficient error-correcting codes and holographic proofs*. PhD thesis, USA, 1995. AAI0576626.

[Sud95] Madhu Sudan. *Efficient Checking of Polynomials and Proofs anf the Hardness of Approximation Problems*, volume 1001 of *Lecture Notes in Computer Science*. Springer, 1995.

[Sud01] Madhu Sudan. Algorithmic introduction to coding theory (lecture notes). 2001. Online at http://people.csail.mit.edu/madhu/FT01/.

[Sud04] Madhu Sudan. Probabilistically checkable proofs. In Steven Rudich and Avi Wigderson, editors, *Computational Complexity Theory*, volume 10 of *IAS / Park City mathematics series*, pages 349–389. AMS Chelsea Publishing, 2004.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*, volume 4948 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008.

[Vid15] Michael Viderman. A combination of testability and decodability by tensor products. *Random Struct. Algorithms*, 46(3):572–598, 2015.

A Relaxed Local Correctability of Tensor Tree Codes

Gur *et al.* [GRR20] (building on [BGH+06]) define a relaxed notion of local correctability, where the correction procedure is allowed to abort in case it detects that the given word is corrupt (i.e., is not a codeword). They additionally show that standard tensor codes are locally correctable in this relaxed sense. In the following lemma, we adapt their proof to tensor tree codes, and show that they are relaxed locally correctable w.r.t. suffix distance (Definition 3.7).

Lemma A.1 (Relaxed Local Correctability of TC^m)**.** *Let $m \in \mathbb{N}$ and let* TC *be a linear tree code with constant tree distance δ. Then, there exists a corrector algorithm $\mathcal{C}$ that has oracle access to a word $w : [n_1] \times \cdots \times [n_m] \to \mathbb{F}^{L^m}$ and takes as input a coordinate $(t_1, \ldots, t_m) \in [n_1] \times \cdots \times [n_m]$, and satisfies the following properties:*

- *(Completeness) If $w \in \mathrm{TC}^m$, then $\Pr[\mathcal{C}^w(t_1, \ldots, t_m) = w(t_1, \ldots, t_m)] = 1$.*
- *(Soundness) There exists a negligible function ϵ such that, letting $n = \max_j n_j$, if $\Delta_{\mathsf{S}}(w, c) \leq (\delta/2H_n)^m$ for some $c \in \mathrm{TC}^m$, then*

$$\Pr\left[\mathcal{C}^w(t_1, \ldots, t_m) \in \{c(t_1, \ldots, t_m), \perp\}\right] \geq 1 - \epsilon(n).$$

- *(Query Complexity) $\mathcal{C}$ makes at most $O(n \cdot \log^{3m}(n))$ queries to w.*

Proof. We prove the lemma by induction. We assume that TC^{m-1} is a relaxed locally correctable code with correction distance $(\delta/2H_n)^{m-1}$ and show that $\mathrm{TC}^m = \mathrm{TC}^{m-1} \otimes \mathrm{TC}$ is relaxed locally correctable as well, with correction distance $(\delta/2H_n)^m$ and a small blow-up in query complexity.

Given access to w, the corrector for TC^m performs the following:

1. Read the entire "row" that contains $(t_1, \ldots, t_m)$, namely $w' : [n_m] \to \mathbb{F}^{L^m}$ defined by $w'(x) = w(t_1, \ldots, t_{m-1}, x)$. If $w' \notin \mathrm{TC}$, output $\perp$ and abort.
2. Repeat the following $\lambda = \log^3(n)/\delta$ times:
   - 2.1 Sample $i \leftarrow \sigma_{n_m}$ (recall this is the suffix distribution, see Definition 3.7) and let $w_i : [n_1] \times \cdots \times [n_{m-1}] \to \mathbb{F}^{L^m}$ be the "column" defined by $w_i(x_1, \ldots, x_{m-1}) = w(x_1, \ldots, x_{m-1}, i)$.
   - 2.2 Invoke the corrector of TC^{m-1} over w_i with input coordinate $(t_1, \ldots, t_{m-1})$ to retrieve a symbol y_i.
   - 2.3 If $y_i \neq w'(i)$, output $\perp$ and abort.
3. Output $w'(t_m)$.

Now, if $w \in \mathrm{TC}^m$, then $w' \in \mathrm{TC}$ and $w_i \in \mathrm{TC}^{m-1}$, implying $y_i = w_i(t_1, \ldots, t_{m-1}) = w'(i)$ for all i by the completeness of the corrector for TC^{m-1}.

If $w' \notin \mathrm{TC}$, then the corrector aborts. Otherwise, if $w \notin \mathrm{TC}^m$ but $w' \in \mathrm{TC}$, let c' be the codeword row $c'(x) = c(t_1, \ldots, t_{m-1}, x)$. If $w' = c'$, then the corrector either aborts or outputs $w'(t_m) = c'(t_m) = c(t_1, \ldots, t_m)$. If $w' \neq c'$, then $\Delta_{\mathsf{T}}(w', c') > \delta$ by the distance of the code and, by Lemma 3.8,

$$\Pr_{i \leftarrow \sigma_{n_m}}[w'(i) \neq c'(i)] = \Delta_{\mathsf{S}}(w', c') \geq \delta/H_n.$$

Additionally, letting c_i be the column $c_i(x_1, \ldots, x_{m-1}) = c(x_1, \ldots, x_{m-1}, i)$, then by assumption

$$\mathbb{E}_{i \leftarrow \sigma_{n_m}}[\Delta_{\mathsf{S}}(w_i, c_i)] = \Delta_{\mathsf{S}}(w, c) \leq (\delta/2H_n)^m.$$

and, therefore, $\Pr_{i\leftarrow\sigma_{n_m}}[\Delta_{\mathsf{S}}(w_i, c_i) \geq (\delta/2H_n)^{m-1}] < 1 - \delta/2H_n$. Hence,

$$\Pr_{i\leftarrow\sigma_{n_m}}[w'(i) \neq c'(i),\ \Delta_{\mathsf{S}}(w_i, c_i) < (\delta/2H_n)^{m-1}] \geq \delta/H_n - \delta/2H_n = \delta/2H_n. \tag{22}$$

The probability that at least one i sampled by the corrector satisfies the two events in Eq. (22) is at least $1 - (1 - \delta/2H_n)^{\lambda} = 1 - e^{-\Omega(\log^2 n)}$. In such a case, by the soundness of the corrector for TC^{m-1}, y_i is either $\perp$ or $c'(i) \neq w'(i)$.

The query complexity of the corrector for TC^m can be bounded by the recursive equation as $Q(m) = n + \lambda \cdot Q(m-1) = O(\lambda^m \cdot n)$. □

B Suffix Distance is Bounded by Flattened Suffix Distance

In the following simple lemma, we upper bound suffix distance (Definition 3.7) between two words by their flattened suffix distance (defined in Proposition 4.9).

Lemma B.1. *For any $w, w' \in \Sigma^m$,*

$$\overline{\Delta_{\mathsf{S}}}(w, w') \geq \frac{1}{m} \cdot \Delta_{\mathsf{S}}(w, w')$$

Proof. First, we show that suffix distance between flattened words w and w' is bounded (up to a factor of m) by the suffix distance over the restriction to any rectangle $I \in \mathbf{Rect}(n)$, i.e. w_I and w'_I, where we consider the one-dimensional suffix distance over the flattening of the rectangles, rather than the high-dimensional metric.

Formally, let φ_I denote the restriction of the mapping φ from Fig. 4 to a rectangle $I \in \mathbf{Rect}(n)$ in the following sense: $\varphi_I(t)$ is the t^{th} coordinate in the sequence $\varphi(1), \varphi(2), \ldots$ that is contained in I. For any $w \in \Sigma^n$ and $I \in \mathbf{Rect}(n)$, we define $\overline{w_I}$ to be the word defined by $w(t) = w_I(\varphi(t))$. It holds that, for any $t' \in [n]$, if t is such that $\varphi_I(t) = \varphi(t')$, then $\sigma(t') \leq \sigma(t)$ (recall Definition 3.7). It follows, then, that

$$\Delta_{\mathsf{S}}(w, w') \leq \sum_{I\in\mathbf{Rect}(n)} \Delta_{\mathsf{S}}(\overline{w_I}, \overline{w_I}') \leq m \cdot \max_{I\in\mathbf{Rect}(n)} \Delta_{\mathsf{S}}(\overline{w_I}, \overline{w_I}').$$

Next, by the monotonicity of φ, for any rectangle $I = [n_1] \times \cdots \times [n_m] \in \mathbf{Rect}(n)$ and $1 \leq t \leq |I|$, letting $(t_1, \ldots, t_m) = \varphi_I(t)$, we have that $|I| - t + 1 \geq \prod_{i=1}^{m}(n_i - t_i + 1)$ and, therefore, $\sigma_{|I|}(t) \leq \sigma_{n_1,\ldots,n_m}(t_1, \ldots, t_m)$ and

$$\Delta_{\mathsf{S}}(\overline{w_I}, \overline{w_I}') \leq \Delta_{\mathsf{S}}(w_I, w'_I).$$

The lemma follows by the above two equations. □

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il