STARKS

Christian Reitwießner
chris@ethereum.org

Ingredients for STARKs:

Encode the computational task into a polynomial problem.

IOP: Prover and verifier exchange messages. In many cases, prover sends gigantic message, but verifier only reads a tiny bit. In such a setting, prover can merkelize, only send root hash and verifier can request parts of the message plus Merkle proof.

(Probabilistically Checkable) Proof of Proximity: Ensures that the polynomial used by the prover is "close" to a low-degree polynomial.

From "Computational integrity with a public random string from quasi-linear PCPs.":

Encode the computation trace as a sequence of N elements from $\mathcal{F}_{2^64}$ and encode this in a polynomial a of degree $N-1$ over $\mathcal{F}_{2^64}$.

ACSP (algebraic constraint satisfaction problem) - NTIME(n)-complete. Instances: $(\mathcal{F}, \{AFF_1, \ldots, AFF_k\}, H, C)$, where $\mathcal{F}$ finite field, AFF_i is affine map on $\mathcal{F}$, i.e. $ax+b$, $a, b \in \mathcal{F}$, $H \subseteq \mathcal{F}$, $C\colon \mathcal{F}^k \to \mathcal{F}$ is a poly of degree at most

Compute $b = \Phi(a)$, where Φ is an ACSP instance and append PCPPs π_a, π_b for both a and b.

Verifier: Input is Φ, oracle access to a, b and π_a, π_b. Run RS-PCPP on (a, π_a) and (b, π_b) and then use sampling to check that the two polynomials a and $\Phi(b)$ are equal.

Interesting: This sampling could be done by outside actors. If an error is found, it is reported.

Reed-solomon-Codes: Univariate polynomial of high degree.

PCPPs only have oracle access to the statement and proof and check that the proof is "close" to a true statement. In contrast, PCPs have full access to the statement and oracle access to the proof and check that the statement is true.

Low-degree test: Procudure to check if a function we have oracle access to (i.e. can ask to be evaluated) is a multivariate polynomial of "low" degree.

Zero-tester: efficient procedure to verify if a function given by an oracle is close to a multivariate polynomial that is zero on every point in a prespecified subset of its domain

New approach in BS08: - reduce SAT to zero-testing. Reduce zero-testing to low-degree testing.

In BS08, all oracle queries are non-adaptive

Definition 0.1. $\mathrm{RS}[\mathbb{F}, S, \rho] = \{f \colon S \to \mathbb{F} \mid$ *there is a poly* p *of degree at most* $\rho\|S\|$ *such that* $p(x) = f(x)$ *on* $S\}$

Definition 0.2. *IOP - interactive oracle proof: IP for a nondeterministic language, where the prover's input includes the witness, the verifier has oracle access to the prover's messages. Proof length* $\ell(x)$ *includes the messages, query complexity* $\mathrm{q}(x)$ *only the number of queries. Perfect completeness and* ε*-soundness. Prover complexity includes the witness size as input, verifier complexity only the input.*

Definition 0.3. *STIK - scalable transparent IOP of Knowledge - soundness against unbounded provers. Language* L*, witness relation* R

- *transparent: all verifier messages and queries are public random coins*
- *scalable: verifier complexity is poly in* n*, polylog in the ntime-complexity of* L *and polylog in* $\frac{1}{\varepsilon}$*. Prover complexity is linear in the ntime-complexity of* L *times the bounds for the verifier*
- *proof of knowledge: there exists a knowledge error function* $\varepsilon'(n)$ *and a randomized extractor* E *that, given oracle access to any prover* $\mathrm{P}*$ *that causes the verifier to accept* x *with probability at least* $\varepsilon'(|x|)$*, outputs in expected time poly of something a witness* w *such that* $(x, w) \in R$*.*
- *privacy preservation: there exists a randomized simulator that samples perfectly the distribution on transcripts of interactions between verifier and prover and runs in time poly* $T(n)$*.*

Complexity in the following is arithmetic complexity.