Oded (April 1, 2024): Arithmetization in probabilistic proof systems

This brief note refers to two fundamental theorems regarding probabilistic proof systems that are proved by using "arithmetization". We refer to co$\mathcal{NP}$ (and even $\#\mathcal{P}$) having interactive proof systems and to $\mathcal{NP}$ having low complexity PCPs (i.e., PCPs of polynomial length and polylogarithmic query complexity). In both cases, an input CNF formula ϕ is being "arithmetized" and a probabilistic proof system is applied to the derived arithmetic expression, via the celebrated sum-check process. However, the similarity is rather superficial. First, the arithmetization is radically different, and second the proof system employed is fundamentally different.

The first case: $\#\mathcal{P} \subseteq \mathcal{IP}$. In this case, the input is a CNF formula ϕ and an integer N, and the task is verifying that ϕ has exactly N satisfying assignments. Here verification should be performed in polynomial-time with the "assistance" of an "untrusted" prover.

Viewing ϕ, which has n variables, as a function from $\{0,1\}^n$ to $\{0,1\}$, we first consider a low-degree extension of ϕ to a finite field $\mathcal{F}$ of size greater than 2^n (but smaller than, say, 2^{n+1}). Denoting this low-degree extension by $\widehat{\phi} : \mathcal{F}^n \to \mathcal{F}$, the sum-check protocol is used to verify that

$$\sum_{\alpha \in \{0,1\}^n} \widehat{\phi}(\alpha) \;=\; N, \tag{1}$$

where $\alpha \in \{0,1\}^n$ is viewed as an n-long sequence over $\mathcal{F}$. The verification process involves n rounds of interaction, and at its end the verifier computes $\widehat{\phi}(r)$, for a value $r \in \mathcal{F}^n$ that was determined during the interactive process, and compares it to a value $v \in \mathcal{F}$ that was also determined during this process.

The second case: $\mathcal{NP} \subseteq \mathcal{PCP}[\log, \operatorname{poly}\log]$. In this case, the input is a 3CNF formula ϕ, and the task is verifying that ϕ is satisfiable. Here verification should be performed in polynomial-time with the "assistance" of a polynomially-long alleged proof that may be queries only at few (i.e., poly-logarithmically many) points.

Here we consider an arithmetization of a function that describes the 3CNF formula ϕ; that is, assuming that ϕ has n variables and $m = O(n^3)$ clauses, we consider the function $D_\phi : [m] \times [2n]^3 \to \{0,1\}$ such that $D_\phi(i, j_1, j_2, j_3)$ indicates whether, for every $k \in [3]$, the k^{th} literal of the i^{th} clause of ϕ is the i_k^{th} literal (i.e., is the $(i_k/2)^{\text{th}}$ variable if i_k is even, and the negation of the $((i_k+1)/2)^{\text{th}}$ variable otherwise). Next, we observe that, for any truth assignment $A : \{0,1\}^n \to \{0,1\}$, it holds that A satisfies ϕ if and only if for every $i \in [m]$ the following holds (over the integers)

$$\sum_{j_1,j_2,j_3 \in [2n]} D_\phi(i, j_1, j_2, j_3) \cdot \prod_{k \in [3]} (1 - A'(j_k)) \;=\; 0, \tag{2}$$

where $A'(j) = A(j/2)$ if the j^{th} literal is non-negated and $A'(j) = 1 - A((j+1)/2)$ otherwise (i.e, if $j = 2j' + b$, where $b \in \{0,1\}$, then $Q'(j) = (1-b) \cdot A(j') + b \cdot (1 - A(j))$). At this point, letting $\ell = \log_2(\max(m, 2n))$, we consider a low-degree extension of $D_\phi : \{0,1\}^{4\ell} \to \{0,1\}$ to a finite field $\mathcal{F}$ of $\operatorname{poly}(\log n)$-size. Denoting this low-degree extension by $\widehat{D} : \mathcal{F}^{4\ell} \to \mathcal{F}$ and viewing A and A' as functions from $\mathcal{F}^\ell$ to $\mathcal{F}$, we wish to verify

$$(\forall i \in [m]) \sum_{j_1,j_2,j_3 \in [2n]} \widehat{D}_\phi(i, j_1, j_2, j_3) \cdot \prod_{k \in [3]} (1 - A'(j_k)) \;=\; 0, \tag{3}$$

where $i \in [m]$ and $j_1, j_2, j_3 \in [2n]$ are viewed as ℓ-long sequences over $\mathcal{F}$. To verify Eq. (3), the verifier selects $r \in \mathcal{F}^m$ from a small-biased sample space (of size $\mathrm{poly}(m)$) such that $R(i)$ the i^{th} element in r, and uses the sum-check process to verify that

$$\sum_{i\in[m]} R(i) \cdot \sum_{j_1,j_2,j_3\in[2n]} \widehat{D}_\phi(i,j_1,j_2,j_3) \cdot \prod_{k\in[3]} (1 - A'(j_k)) \;=\; 0, \tag{4}$$

The verification process involves 4ℓ steps, in which the verifier obtains adequate answers by queries to the oracle. After these steps, the verifier needs to verify a claim of the type $\widehat{D}_\phi(i,j_1,j_2,j_3) \cdot \prod_{k\in[3]}(1 - A'(j_k)) \;=\; v$, for some $i, j_1, j_2, j_3 \in \mathcal{F}^\ell$ and $v \in \mathcal{F}$ that were determined by the process. In addition, the verifier should check that A (which determines A') is a low degree polynomial (i.e., it needs to employ a "good" low degree tester).

(Recall that the soundness of the sum-check process relies on the assumption that the expressions that occur in it are low degree polynomials (i.e., have degree $o(|\mathcal{F}|/t)$, where t is the number of iterations). This condition holds trivially in the case that the verifier determines by itself all expressions in the original sum (as in Eq. (1)), but this is not the case in Eq. (4): Indeed, the verifier does determine $\widehat{D}_\phi$ but it has no control over A.)

We warn that the foregoing description yields a verifier of randomness complexity $O(\ell \log |\mathcal{F}|) = \widetilde{O}(\log n)$, which falls (slightly) short of our goal of having logarithmic randomness complexity, which corresponds to polynomial proof length. (In contrast, the query complexity is also $O(\ell \log |\mathcal{F}|)$, which meets our goal.) To obtain the desired randomness complexity, we view the original functions (i.e., D_ϕ and A) as functions over a larger alphabet; specifically, as functions over an alphabet of size $s = O(\log n)$. This allows to use $\ell = O((\log n)/\log\log n)$ (since $s^\ell \geq \max(2n, m)$), and in that case $O(\ell \log |\mathcal{F}|) = O(\log n)$, as desired. Note that this will increase the query complexity of each iteration by a factor of s, which is fine (i.e., we will get query complexity $O(\log n)^2$).

Needless to say, the same idea (i.e., using a larger alphabet) can be employed in the context of $\#\mathcal{P} \subseteq \mathcal{IP}$ (cf., Eq. (1)), but the effect there is less dramatic. Specifically, using an alphabet of size s will reduce the number of iterations by a factor of $\log_2 s$, but will increase the total communication complexity by a factor of $s/\log s$.

Digest. As evident from the foregoing description, the arithmetization in the two cases is fundamentally different. The common theme is that in both cases a Boolean formula is being arithmetized, an assertion regarding the number of evaluation-points on which the Boolean formula evaluates to 1 (resp., to 0) is reduced to an assertion regarding a corresponding sum of the arithmetic expression, and the sum-check process is employed in order to verify the latter assertion.

In the first case (of $\#\mathcal{P} \subseteq \mathcal{IP}$), *the Boolean formula being arithmetized is the input CNF formula*, and the evaluation-points are its truth assignments. In the second case (of $\mathcal{NP} \subseteq \mathcal{PCP}[\log, \mathrm{poly}\log]$), *the Boolean formula being arithmetized is a function that describes the clauses of the input 3CNF formula*, and the evaluation-points are all 4-tuples (i, j_1, j_2, j_3) such that i is an index of a clause and j_k is an index of a literal (i.e., a variable and a bit indicating possible negation). Furthermore, the arithmetic expression also refers to a function that is *supposedly* a low degree extension of an assignment that satisfies the input 3CNF formula, and the verifier is given oracle access to this function. Hence, on top of employing the sum-check process, the verifier needs to *verify that the foregoing function is indeed a low degree polynomial.*