Electronic Colloquium on Computational Complexity, Report No. 153 (2020)

# TOTAL FUNCTIONS IN THE POLYNOMIAL HIERARCHY

ROBERT KLEINBERG[*], DANIEL MITROPOLSKY[†], AND CHRISTOS PAPADIMITRIOU[†]

Abstract. We identify several genres of search problems beyond **NP** for which existence of solutions is guaranteed. One class that seems especially rich in such problems is **PEPP** (for "polynomial empty pigeonhole principle"), which includes problems related to existence theorems proved through the union bound, such as finding a bit string that is far from all codewords, finding an explicit rigid matrix, as well as a problem we call Complexity, capturing Complexity Theory's quest. When the union bound is generous, in that solutions constitute at least a polynomial fraction of the domain, we have a family of seemingly weaker classes $\alpha$-**PEPP,** which are inside $\mathbf{FP}^{\mathbf{NP}}|$poly. Higher in the hierarchy, we identify the constructive version of the Sauer-Shelah lemma and the appropriate generalization of **PPP** that contains it. The resulting total function hierarchy turns out to be more stable than the polynomial hierarchy: it is known that, under oracles, total functions within **FNP** may be easy, but total functions a level higher may still be harder than $\mathbf{FP}^{\mathbf{NP}}$.

## 1. Introduction

The complexity of total functions has emerged over the past three decades as an intriguing and productive branch of Complexity Theory. Subclasses of **TFNP**, the set of all total functions in **FNP**, have been defined and studied: **PLS, PPP, PPA, PPAD, PPADS**, and **CLS**. These classes are replete with natural problems, several of which turned out to be complete for the corresponding class, see e.g. [4, 5].

Each of these classes corresponds naturally to a very simple existential argument. For example, **PLS** is the class of all total functions whose proof of totality relies on the fact that *every finite dag must have a sink*, while **PPAD** captures this true existential statement: *"If a finite directed graph has an unbalanced node (i.e., a node whose in-degree differs from its out-degree), then it must have another unbalanced node."* The class of total functions **PPP** (for "polynomial pigeonhole principle") captures the well known fact that *"if there are more pigeons than pigeonholes, there must be a pigeonhole with two or more pigeons."* This latter complexity class has attracted much attention due to its close connections to cryptography, and there has been recent progress towards natural complete problems [2, 8, 13].

More recently a logic-inspired class **PTFNP** (for "provable **TFNP**") was identified containing all of the above classes [6], its definition motivated by the existential proof point of view described above. It was also pointed out in [6] that finitariness is necessary for the definition of a meaningful class of total functions, in that any non-finitary existence theorem — that is, one that also holds for infinite structures — results in a computational problem that is provably easy. Also recently, an intriguing link between the possibility of **TFNP**-hardness and average-case hardness was discovered [7].

The simple statement on which **PPP** is based has a very natural "dual" variant, call it the *empty pigeonhole principle,* namely: *"if there are more pigeonholes than pigeons, then there must*

[*] Cornell University.
[†] Columbia University.

ISSN 1433-8092

*be an empty pigeonhole."* Concretely, given a circuit $C$ mapping $[2^n - 1]$ to $[2^n]$[1], find a bit string of length $n$ that is not in $C$'s range. Call this problem Empty. One could even define a class based on the empty pigeonhole principle, call it **PEPP** (for "polynomial empty pigeonhole principle," the set of all total function problems polynomial-time reducible to Empty). At first sight, **PEPP** may seem very close to **PPP** — identical, perhaps? — *until one notices that* **PEPP** *is not obviously in* **NP!** For **PPP,** one can guess and check the offending pigeonhole *and* the two pigeons in it — but for **PEPP?** Once the empty pigeonhole has been guessed, proving it is empty requires one to look at all pigeons. An alternation of quantifiers appears to be at work!

*In this paper we introduce a hierarchy of total search problems analogous to the polynomial hierarchy of decision problems.* **TFNP** is the first level of the hierarchy, and the class **PEPP** just defined is at the second level of this hierarchy, denoted $\mathbf{TF\Sigma}_2$. Actually, we shall soon see that there is a natural and interesting search problem occupying the third level of the hierarchy. (For the formal definition of $\mathbf{TF\Sigma}_i$ and some basic facts about this hierarchy, see the Appendix.)

The first result we prove in this direction is that, despite the apparent similarity and "symmetry" outlined above, **PEPP** contains **PPP** — *and in fact, all of* **FNP** (Theorem 1; the proof is easy).

Empty and **PEPP** are closely associated with the familiar probabilistic argument known as *the union bound.* There is a formal way to see this: Consider a generic instance of Empty, that is, a circuit $C$ mapping $[2^n - 1]$ to $[2^n]$; the task is to find a possible output in $[2^n]$ not in the circuit's range. Interpret now an input $x$ as $x = yz$, where $|z| = n - m$ and $|y| = m$, and where $y$ encodes a "bad event" — in the sense of the union bound — with probability $2^{-m}$ ($2^{-m} - 2^{-n}$ for one of the events), while $z$ indexes the $2^{n-m}$ (respectively, $2^{n-m} - 1$) elements of the whole probability space of size $2^n$ that constitute the bad event. Hence, the empty pigeonhole principle can be interpreted as the union bound. Many of the important natural problems in **PEPP** correspond to existential proofs through the union bound, or more generally through counting.

One of these problems is Remote Point: Given a code — generically, a circuit mapping $[2^k]$ to $[2^n]$ where $k < n$ and the codewords are the range of the circuit — find an $n$-bit string that is far from all codewords (as far in Hamming distance, that is, as is guaranteed by the union bound). It is not hard to see that Remote Point is **PPEP**-complete. The important open problem here is the complexity of the special case of Remote Point in which the circuit is a linear function in $\mathbf{GF}_2$; this is a much studied problem [1].

Other natural problems in **PEPP** appear to capture interesting aspects of *complexity.* To start with the more indirect one, Rigid Matrix Completion is the following problem: find a rigid matrix in $\mathbf{GF}_2^{n\times n}$ (that is, an $n \times n$ matrix whose rank cannot collapse to something tiny by manipulating very few entries, details supplied later) given several of its entries. Rigid matrices have been shown [14] to be abundant, and to capture logarithmic-depth circuit complexity, while their explicit construction has remained an important open problem since Valiant's paper. We define the relevant problem as a *completion* variant of the explicit construction problem — that is, part of the matrix is specified — to overcome a familiar impediment: without a binary input of some substantial length, one is dealing with a *sparse* problem, and current techniques seem unable to fathom the complexity of such problems. A second problem in the same vein, Ramsey-Erdős completion, embodies Erdős's famous 1959 proof that the $n$-th Ramsey number is at least $2^{\frac{n}{2}}$, historically one of the first applications of the probabilistic method.

[1]We denote the set $\{0, 1, \cdots, M - 1\}$ by $[M]$. We shall see that it is easy to construct circuits with arbitrary integer domains and ranges.

Complexity is a problem asking, given a bit string of length $n$, to find an explicit Boolean function with $\log n$ inputs which requires $\Omega(\frac{n}{\log^2 n})$ gates — that is, an explicit exponential lower bound. The problem is, again, defined through a circuit. The circuit interprets its input gates as the representation of a circuit with $\log n$ inputs and $O(\frac{n}{\log^2 n})$ gates, where besides the usual Boolean gates we also allow *oracle gates* with fan-in $\log n$. The output of the circuit is the Boolean function computed by this circuit, encoded as a bit string of length $2^{\log n} = n$. The input to the problem (on the basis of which the circuit is constructed and the computation of the circuit is carried out) is interpreted as an *oracle,* encoding in its $n$ bits the answers to all possible oracle inputs. The task is to discover an $n$-bit string that is not in the range of this circuit under this oracle — that is to say, a Boolean function with $\log n$ variables which therefore requires $\Omega(\frac{n}{\log^2 n})$ gates to be computed with the given oracle. The oracle here is needed, again, to render a unary function binary.

Is this problem **PEPP**-complete, or otherwise hard in a demonstrable sense? This is the most important problem left open in this paper. We are aware of one immediate obstacle: it turns out that many of the problems we discussed above, Complexity among them, belong in a significantly weakened subclass of **PEPP**. Let $\alpha$ be a positive quantity, possibly a function of $n$, and define the class $\alpha$-**PEPP** (pronounced *abundant* **PEPP**) to be the variant in which the given circuit does not map $[2^n]$ to $[2^n + 1]$, but instead $[2^n]$ to $[(1 + \alpha) \cdot 2^n + 1]$; evidently, $\mathbf{PEPP} =$ 0-**PEPP**, while many of the problems in **PEPP** discussed are known to belong to $\alpha$-**PEPP** for some constant $\alpha$.

We prove two theorems on $\alpha$-**PEPP**. First, we establish that the precise value of $\alpha$ is in some sense irrelevant, in that any class $\alpha$-**PEPP** with $\alpha$ between $\frac{1}{\text{poly}}$ and poly can be reduced to any other such class *through* $\mathbf{FP}^{\mathbf{NP}}$ *reductions* (Theorem 7; it is not known whether polynomial time reductions are possible here). Second, it turns out that for any problem in $\alpha$-**PEPP** with $n$ input gates there is a small set of outputs (strings of length $\lceil n \log(1+\alpha)\rceil$) such that, for any input, one of them is an empty pigeonhole. (The proof is by — what else? — the union bound.) It follows that $\alpha$-**PEPP** is contained in $\mathbf{FP}^{\mathbf{NP}}|\text{poly}$, $\mathbf{FP}^{\mathbf{NP}}$ with polynomial advice; we see no reason why **PEPP** should be so confined.

So far we have been discussing problems and classes in $\mathbf{TF\Sigma_2}$, the next level after **TFNP** of what can be called the polynomial total function hierarchy. It turns out that there is at least one interesting problem further up. Shattering is the following problem: we are given a circuit $C$ with $k$ input gates and $n$ output gates, which is supposed to represent a family of $2^k$ subsets of $[n]$. We must return either a collision in this circuit, establishing that the family has fewer than $2^n$ distinct sets; or otherwise a $d$-subset of $[n]$, call it $D$, which is *shattered* by the family — that is, every subset of $D$ can be written as $D \cap C(x)$ for some set $C(x)$ in the family; such a set is guaranteed to exist by the Sauer-Shelah lemma [10, 12, 15], as long as $C$ has no collisions and $k$ is large enough as a function of $n$ and $d$. Notice immediately that there are two alternations of quantifiers in this existential result: *there is* a set $D$ such that *for every* subset $G$ of $D$ *there is* an output $C(x)$ of $C$ such that $G = D \cap C(x)$: we are in the class $\mathbf{TF\Sigma_3}$! In fact, we show that Shattering belongs to a very natural subclass of $\mathbf{TF\Sigma_3}$: it belongs to $\mathbf{PPP}^{\mathbf{\Sigma_2}}$, the pigeonhole principle class when the function mapping pigeons to pigeonholes can use a $\mathbf{\Sigma_2}$ oracle in its computations.

## 2. The Problems in **PEPP**

Empty is the following search problem: Given a circuit $C$ with Boolean gates mapping $[2^n - 1]$ to $[2^n]$, find a $y \in [2^n]$ such that $y \neq C(x)$ for all $x \in [2^n - 1]$.

**Remark:** In this paper, we shall blur the distinction between bitstrings and binary integers. Our Boolean circuits have a domain and range whose cardinality is not necessarily a power of two, which may seem peculiar. In this paper we shall consider Boolean circuits mapping $[M]$ to $[N]$, where $M, N$ are arbitrary integers greater than one. Such a circuit $C$ has $\lceil \log M \rceil$ inputs and $\lceil \log N \rceil$ outputs, and for all $x$ $C(x)$ is defined to be $C(M-1)$ if $x \geq M$, and also for all $x$ $C(x) = N - 1$ whenever the value computed by $C$ on input $x$ (or on input $M - 1$ if $x \geq M$) is at least $N$. Hence, Empty can be defined in terms of any circuit $C : [M] \mapsto [M+1]$ — or even $C : [M] \mapsto [N]$ as long as $M < N$. For larger $N$ the problem may be easier, but it is reducible to Empty (as long as $\log N \leq \text{poly} \log M$).

Coming back to Empty, we can now define a class of total functions **PEPP** as all total functions that are polynomial-time reducible to Empty. One rather immediate — and yet a little surprising — fact to observe about **PEPP** is the following:

**Theorem 1. FNP $\subseteq$ PEPP.**

*Proof.* We prove that Sat can be reduced to Empty. Let $\phi$ be a CNF formula with $n$ variables, without loss of generality not satisfied by the all-true truth assignment. Consider now the following polynomially computable function $C$ from $[2^n - 1]$ to $[2^n]$: For every truth assignment $t$ different from the all-true one $1^n$, $C$ tests whether $t$ satisfies $\phi$. If it does, then $C(t) = 1^n$, and if it does not then $C(t) = t$. Now, if we could solve Empty, that is, if we could find a solution $s \in [2^n]$ not in the range of $C$, then we would have solved the Sat problem for $\phi$: If $s \neq 1^n$ then $\phi$ is satisfiable and $s$ satisfies it; otherwise, $\phi$ is unsatisfiable. $\square$

This result suggests that **PEPP** is genuinely a subclass of $\mathbf{TF\Sigma_2}$, the generalization of **TFNP** to the first level of the polynomial hierarchy. Once we are dealing with $\mathbf{TF\Sigma_2}$, it is tempting to define classes such as **PEPP** as the set of all problems that can be reduced through $\mathbf{FP^{NP}}$ reductions — not just polynomial-time reductions — to a specific problem, such as Empty in the case of **PEPP**. This option becomes relevant when dealing with $\alpha$-**PEPP** in the next subsection.

As we sketched in the introduction, Empty and **PEPP** can be alternatively thought as a computationally constructive form of the union bound. A most prominent and early use of the union bound is in Shannon's work on codes, and this is captured by the following problem, Remote Point: given a code, which generically means a circuit $C$ mapping $[M]$ to $[N]$ with $N > M$, find a bitstring $x \in [N]$ whose Hamming distance from any codeword $y$, that is, any $y$ such that $y = C(z)$ for some $z \in [M]$ is at least $d$, where $d$ is the largest integer such that the Hamming ball of radius $d - 1$ has fewer than $N/M$ elements.

**Proposition 2.** Remote Point *is in* **PEPP**.

*Proof.* Its proof of totality is an application of the union bound. $\square$

In fact, Remote Point is strictly speaking **PEPP**-complete, because any instance of Empty is also an instance of Remote Point with $d = 1$.

Next we introduce two problems in **PEPP** capturing two other classical applications of the union bound. In $\delta$-Rigid Matrix Completion, where $0 < \delta < \frac{1}{3}$, we are given the first $\lceil \log n \rceil$ rows of an $n \times n$ matrix with elements in $\mathbf{GF}_2$. We seek to complete this to a full matrix in

$\mathbf{GF}_2^{n\times n}$ that is *$\delta$-rigid:* it cannot be turned into a matrix of rank $\leq \delta n$ by changing $n^\delta$ or fewer entries in each row.

Why do we have to phrase the quest for the rigid matrix as a completion problem? The reason is that the alternative ("Given $n$, find an $n \times n$ rigid matrix") is a *sparse* problem, that is, it has a polynomially (in $n$) many instances of length $\leq n$, which places it in complexity limbo, see e.g. [9]; alternatively, if $n$ is given in binary, then the problem is even more ill-posed since an exponentially long output is required[2].

To see that $\delta$-Rigid Matrix Completion reduces to Empty, consider the circuit $C : [M] \mapsto [N]$ where $N = 2^{n(n-\log n)}$ is the total number of possible completions of the matrix, and $[M]$ is the total number of matrices of the form $(L + S)$, where $L$ is a $\delta n$-rank $n \times n$ extension (there are $2^{2\delta n^2 - n\log n}$ such matrices), and $S$ is a matrix that has at most $n^\delta$ ones per row (there are at most $\binom{n}{n^\delta} \leq 2^{n^{2\delta}}$ such matrices. Thus, there are at most $2^{2\delta n^{1+2\delta} - n\log n} < 2^{n^2 - n\log n}$ sums, and thus $M < N$.

Ramsey-Erdős completion is the problem of finding an $n$-node graph with no independent set of size $2\lceil\log n\rceil$ and no clique of this size, given the connectivity of $\lceil \log n\rceil$ nodes in the graph. There are at least $2^{n^2 - 2n\lceil\log n\rceil} > M$ such completions, while the number of objects of the form $(A, G)$, where $A$ is either a clique or an independent set of size $2\lceil\log n\rceil$ and $G$ is the remaining connectivity of the graph is $N = 2^{\binom{n}{2} - \binom{2\lceil\log n\rceil}{2}}$. It follows from the standard calculation that $M < N$. We have established this result:

**Proposition 3.** *$\delta$-*Rigid Matrix Completion *and* Ramsey-Erdős completion *are in* **PEPP**.

Of these problems Ramsey-Erdős completion seems the easiest computationally, as it belongs in a variant of **BPP** in which $n^{O(\log n)}$ computations are allowed.

## 2.1. The Problem Complexity.

The field of Complexity Theory is about identifying a Boolean function with $v$ variables requiring a number of gates that grows faster than polynomially in $v$. It is well known since Shannon's union bound proof [11] that almost all Boolean functions with $v$ variables have complexity at least $2^{\frac{cv}{\log v}}$ for some $c > 0$; however, no explicit function of complexity that is not $O(v)$ is known.

We can now define Complexity: given a bitstring $x$ of length $n$, find a Boolean function with $v = \lfloor \log n \rfloor + 1$ inputs which cannot be computed by an *$x$-oracle circuit* with $c \cdot \frac{n}{\log^2 n}$ gates, where $c > 0$ is a fixed constant. Here, by "$x$-oracle circuit" we mean a Boolean circuit which, besides the traditional And, Or, Not gates also has an Oracle gate, with fan-in $\lfloor \log n \rfloor$, which when its inputs are the bits $b_1, \ldots, b_{\lfloor \log n \rfloor}$, the value of the gate is *the $b + 1$-th bit of $x$*, where $b < n$ is the integer spelled by the bits.

We shall assume that, for each $k, \ell > 0$, we have a standard *representation* $R^{k,\ell}$ of such oracle circuits, where $k$ is the number of inputs to the circuit and $\ell$ is the fan-in of the oracle gates. $R^{k,\ell}$ is a partial function (that is, possibly undefined) from bitstrings to circuits, such that:

- Every $x$-oracle circuit $K$ has at least one bitstring $z$ such that $R^{k,\ell}(z) = K$.
- Given $z$, $K = R^{k,\ell}(z)$ can be decoded in polynomial time.
- If $K$ has $g$ gates, the length of all $z$'s such that $R^{k,\ell}(z) = K$ is at most $c \cdot g \log^2 g$, where $c$ is a constant.

[2]A related question is, how large should be the given part of the matrix in order to avoid sparsity? Giving the first row is not enough, since, there are, up to isomorphism, $n + 1$ such rows, and a similar argument precludes finitely many rows. With $\log n$ rows in the input, the problem is arguably no longer sparse.

It is easy to see that these desiderata are satisfied by several standard and natural representations (for example, encoding every bit by two bits to create delimiters, encoding gate names by binary integers $\leq g$ and similarly with gate types, and finally encoding the adjacency lists of the circuit graph). The extra $\log g$ in the last item is due to the oracle gate, whose adjacency list requires $\log^2 g$ bits.

Coming back to Complexity, it is, evidently, a computational problem that captures certain aspects of Complexity Theory. We shall show that it is a total problem, and in fact one in the class **PEPP**.

The argument is essentially Shannon's: given input $x$ of length $n$, we construct a circuit $C_x$ implementing the following polynomial-time (in $n$) algorithm: on any input $y$ also of length $n$, $C_x$ interprets $y$ as a binary representation of an $x$-oracle circuit $K_y = R^{k,\ell}(y)$ with $k = \lfloor \log n \rfloor + 1$ input gates and fan-in $\ell = \lfloor \log n \rfloor$, and goes on to construct it (if $R^{k,\ell}(y)$ is undefined, $C_x$ outputs a default string). Next, $C_x$ simulates $K_y$ consecutively on each possible input in $[2^{\lfloor \log n \rfloor + 1}]$. The output of $C_x$ is then the concatenation of these $2^{\lfloor \log n \rfloor + 1}$ bits output by the circuit $K_y$, in the order in which they were produced.

In other words, the circuit $C_x$ maps $M = [2^n]$ (all inputs $y$ of length $n$) to $N = 2^{2^{\lfloor \log n \rfloor + 1}}$ possible outputs, and it is clear that $N > M$. Therefore, if we were able to solve Empty and obtain a possible output not realized by any possible input, we would be able to find the table of a Boolean function with $\lfloor \log n \rfloor + 1$ inputs which cannot be represented by $n$ bits, and therefore requires $\Omega(\frac{n}{\log^2 n})$ gates. This completes the proof of the following result:

**Proposition 4.** Complexity *is in* **PEPP**.

2.2. **Wasteful counting and $\alpha$-PEPP.** When the union bound is used to prove the existence of objects with a certain property by showing that a random object satisfies the property with positive probability, this success probability is typically not exponentially small. The reason is that this genre of existence proof seems inherently wasteful: the union bound adds probabilities of events that typically overlap, while counting objects such as non-rigid matrices and circuits typically counts the same object many times (for example, all permutations of gate names), and there seems to be no way to be accurate enough. To capture the complexity of the search problems implied by union bound arguments with a significant "margin of error", we define a family of complexity classes **$\alpha$-PEPP**, parameterized by a function $\alpha : \mathbb{N} \to \mathbb{R}_+$.

The complexity class **$\alpha$-PEPP** is defined to consist of all total functions that are polynomial-time reducible to the following $\alpha$-Empty problem. An instance of $\alpha$-Empty is given by a bitstring of length $n$ interpreted as a description of a circuit $C$ mapping $[M]$ to $[N]$, where $N/M > 1 + \alpha$. The search problem is to find $y \in [N]$ such that $y \neq C(x)$ for all $x \in [M]$.

Note that **0-PEPP** = **PEPP**. The complexity class **1-PEPP**, which we will denote by **APEPP** in the sequel, contains most of the search problems introduced earlier in this section.

**Proposition 5.** *The problems* $\delta$-Rigid Matrix Completion, Ramsey-Erdős completion, *and* Complexity *all belong to* **APEPP**.

*Proof.* Above, we presented reductions from $\delta$-Rigid Matrix Completion, Ramsey-Erdős Completion, and Complexity to Empty with the following parameters.

- For a $\delta$-Rigid Matrix Completion instance of size $n \times n$, the reduction yields a circuit of size poly($n$) mapping $[M]$ to $[N]$, where $M = 2^{2\delta n^{1+2\delta} - n \log n}$ and $N = 2^{n^2 - n \log n}$,

$$N/M = 2^{n^2} - 2n^{1+2\delta}.$$

The ratio $N/M$ exceeds 2 when $\delta < \frac{1}{3}, n \geq 64$.

- For a Ramsey-Erdős Completion instance with $n$ vertices, the reduction yields a circuit of size poly($n$) mapping $[M]$ to $[N]$, where $M = 2^{\binom{n}{2} - \binom{2\lceil \log n \rceil}{2}}$, $N = 2^{n^2 - 2n\lceil \log n \rceil}$,
$$N/M = 2^{\binom{n+1}{2} - 2n\lceil \log n \rceil - \binom{2\lceil \log n \rceil}{2}}.$$
The ratio $N/M$ exceeds 2 when $n \geq 23$.
- For a Complexity instance of length $n$, the reduction yields a circuit of size poly($n$) mapping $[M]$ to $[N]$, where $M = 2^n$, $N = 2^{2^{\lfloor \log n \rfloor + 1}}$,
$$N/M = 2^{2^{\lfloor \log n \rfloor + 1} - n}.$$
The ratio $N/M$ is greater than 2 for all $n \in \mathbb{N}$.[3]

Thus, the reductions presented earlier verify that all three problems belong to **APEPP**. □

It seems unlikely that **APEPP** contains a **PEPP**-complete problem, due to the following upper bound on the complexity of **APEPP**.

**Theorem 6.** *For any function $\alpha(n) > \frac{1}{\text{poly}(n)}$, we have $\boldsymbol{\alpha}$-**PEPP** $\subseteq$ $\mathbf{FP^{NP}}|\text{poly}$.*

*Proof.* We need only show that $\alpha$-Empty $\in \mathbf{FP^{NP}}|\text{poly}$. Consider a circuit $C$ mapping $[M]$ to $[N]$, where $N/M > 1 + \alpha$. Let $R(C)$ denote the range of $C$, i.e. the set of all $y \in [N]$ such that there exists $x \in [M]$ with $C(x) = y$. The probability that a random $y \in [N]$ belongs to $R(C)$ is at most $M/N < 1/(1+\alpha)$. Hence, if $k = \lceil n/\alpha \rceil$ and $y_1, y_2, \dots, y_k$ are independent random elements of $[N]$, the probability that $\{y_1, \dots, y_k\} \subseteq R(C)$ is less than $(1+\alpha)^{-k} < e^{-n}$. By the union bound, a random $k$-tuple $(y_1, y_2, \dots, y_k) \in [N]^k$ has positive probability of containing a valid solution to *every* length-$n$ instance of $\alpha$-Empty. In fact, this probability is greater than $1 - 2^n \cdot e^{-n}$.

Given an **NP** oracle and an advice string $(y_1, \dots, y_k)$ such that every $\alpha$-Empty instance has a solution in the set $\{y_1, \dots, y_k\}$, it becomes easy to solve $\alpha$-Empty in $O(kn)$ time: for $1 \leq i \leq k$ one queries the **NP** oracle to find out if there exists an $x \in [M]$ such that $C(x) = y_i$, and one outputs the first $y_i$ for which the oracle confirms that no such $x$ exists. □

We conclude this section by showing a collapse of the complexity classes $\boldsymbol{\alpha}$-**PEPP** for $\frac{1}{\text{poly}(n)} \leq \alpha(n) \leq 2^{\text{poly}(n)}$ under $\mathbf{FP^{NP}}$ reductions.

**Theorem 7.** *If $\frac{1}{\text{poly}(n)} \leq \alpha(n) \leq 2^{\text{poly}(n)}$, then $\boldsymbol{\alpha}$-**PEPP** and **APEPP** are equivalent under $\mathbf{FP^{NP}}$ reductions.*

*Proof.* For any positive integers $N, k$, let $T : [N^k] \to [N]^k$ denote the function that takes the binary representation of a number $x \in [N^k]$, writes $x$ in base $N$ as a sequence of $k$ digits (each an element of $[N]$), and outputs the binary string obtained by concatenating the binary representations of each of these $k$ base-$N$ digits.

Suppose $\beta(n), \gamma(n) : \mathbb{N} \to \mathbb{R}_+$ are any two functions such that
$$k(n) \triangleq \lceil \log_{1+\beta(n)}(1 + \gamma(n)) \rceil \leq \text{poly}(n).$$

[3]Actually, the ratio is greater than or equal to 2 for all $n$, but it is equals 2 when $n+1$ is a power of 2. Since the definition of $\alpha$-Empty requires the strict inequality $N/M > 1+\alpha$, we need to correct for this technicality with a small modification in the definition of Complexity, tweaking the problem definition to use a slightly smaller constant $c$ so that all circuits of the appropriate size or less can be encoded in $n-1$ bits, instead of $n$.

We can reduce $\beta$-Empty to $\gamma$-Empty as follows. Given a length-$n$ bitstring describing a circuit that computes a function $C : [M] \to [N]$, let $k = k(n)$ and construct the description of a circuit that computes the function $C' : [M^k] \to [N^k]$ defined via the following composition:

$$[M^k] \xrightarrow{T} [M]^k \xrightarrow{C^k} [N]^k \xrightarrow{T^{-1}} [N^k].$$

Here $C^k$ denotes the function that applies $C$ to each element of a $k$-tuple.

Assuming $N/M > 1 + \beta(n)$, we have $N^k/M^k > (1 + \beta(n))^k \geq 1 + \gamma(n)$, by the definition of $k$. Hence, by solving an instance of $\gamma$-Empty and applying the function $T$, we obtain a $k$-tuple $(y_1, \ldots, y_k)$ that is not in the range of the function $C^k : [M]^k \to [N]^k$. Now, given an **NP** oracle, we can proceed as in the proof of Theorem 6 to find $y \in [N]$ such that for all $x \in [M]$, $y \neq C(x)$. Namely, for $1 \leq i \leq k$ one queries the **NP** oracle to find out if there exists some $x_i \in [M]$ such that $C(x_i) = y_i$. If such an $x_i$ existed for each $i$, then $C^k(x_1, \ldots, x_k)$ would equal $(y_1, \ldots, y_k)$ contradicting our assumption that $(y_1, \ldots, y_k)$ is not in the range of $C^k$. Therefore, for at least one value of $i$ the oracle will answer that no $x_i \in [M]$ satisfies $C(x_i) = y_i$, and we can output this $y_i$ as a solution of the given $\beta$-Empty instance.

We have shown a $\mathbf{FP^{NP}}$ reduction from $\boldsymbol{\beta}$**-PEPP** to $\boldsymbol{\gamma}$**-PEPP** whenever $\log_{1+\beta(n)}(1 + \gamma(n)) = \text{poly}(n)$. If $\frac{1}{\text{poly}(n)} \leq \alpha(n) \leq 2^{\text{poly}(n)}$, then a reduction from $\boldsymbol{\alpha}$**-PEPP** to **APEPP** is obtained by taking $\beta = \alpha$ and $\gamma \equiv 1$, and a reduction from **APEPP** to $\boldsymbol{\alpha}$**-PEPP** is obtained by taking $\beta \equiv 1$ and $\gamma = \alpha$. □

## 3. The Shattering Problem

We recall the definition of *shattering*, an important notion in finite set theory and classical learning theory:

**Definition 8.** A family of sets over some finite universe, $F = \{s_1, s_2, \ldots\}$, *shatters a set $s$ if for every subset $t \subseteq s$, there exists $s_i \in F$ such that $t = s \cap s_i$.*

The famous Sauer-Shelah lemma guarantees shattering properties if the family is large enough. Here it is stated in its "strong" form:

**Theorem 9.** (Sauer-Shelah Lemma, Strong.) *A family $F$ of finite sets shatters at least $|F|$ sets.*

The more well-known statement of the Sauer-Shelah lemma is the weak form, which follows from the above:

**Corollary 10.** (Sauer-Shelah Lemma, Weak.) *If a family of sets $F$ over a universe of $n$ elements satisfies $|F| > \sum_{i=0}^{d-1} \binom{n}{i}$, then $F$ must shatter a set of cardinality at least $d$.*

*Proof.* (Of the weak form from the strong form:) There are at most $\sum_{i=0}^{d-1} \binom{n}{i}$ sets in an $n$-element universe that have size less than $d$. □

It is natural to consider the search problem resulting from this lemma: given a family of sets over $n$ elements, which can be represented as $n$ bit strings, find a large shattered set. This search problem is interesting for two reasons: first, its standard proof uses a counting argument that is, in essence, non-constructive, and second, it involves multiple alternations: given a family find the set (*exists*) such that *for all* subsets there *exists* a corresponding set in the family. In fact, this has one more alternation than all the problems we have considered previously, which belong in $\mathbf{TF\Sigma_2}$. Instead, this belongs in $\mathbf{TF\Sigma_3}$.

**Definition 11.** Let $BinomSum(n, d)$ denote $\sum_{i=0}^{d-1} \binom{n}{i}$.

**Definition 12.** In the Shattering problem, we are given as inputs parameters $n$, $d$, and $k > \log(BinomSum(n, d))$, and a circuit computing a function $C : \{0,1\}^k \to \{0,1\}^n$, representing $2^k$ indexed sets the collection of which we will denote $F$. The search problem is to output either a pair of indices $x_1 \neq x_2$ such that $C(x_1) = C(x_2)$ (a *collision*, in which case the premise of the Sauer-Shelah lemma is not satisfied), or a subset $Y \subseteq [n]$ of size $|Y| = d$ that is shattered by the $F$, the range of C.

The following is now clear:

**Proposition 13.** Shattering *is in* $\mathbf{TF\Sigma_3}$.

*Proof.* Consider the Turing Machine $M((n, d, k, C), s, (u, i))$, which:

1. checks that $k > \log(BinomSum(n, d))$;
2. checks whether $s$ is a string representing a tuple $x_1, x_2$ of $k$-bit strings, in which case it accepts if $C(x_1) = C(x_2)$ and rejects otherwise;
3. checks whether $s$ is an $n$-bit string, in which case it accepts if $s \cap u = s \cap C(i)$ and rejects otherwise.

Clearly, $s$ solves Shattering on the input $(n, d, k, C)$ when the conditions of the Sauer-Shelah lemma are not satisfied, or, if $\forall u \exists i$ s.t. $M((n, d, k, C), s, (u, i)) = 1$. That Shattering is total is a consequence of the Sauer-Shelah lemma. □

More interestingly, we can place Shattering in a generalization of **PPP** that lies within $\mathbf{TF\Sigma_3}$:

**Theorem 14.** Shattering *is in* $\mathbf{PPP^{\Sigma_2}}$

The main technical result is the following lemma, from which the theorem follows naturally.

**Lemma 15.** *Using a* $\mathbf{\Sigma_2}$ *oracle, one can compute a polynomial time function* $M$ *mapping distinct sets in* $F$ *to distinct sets shattered by* $F$.

*Proof.* (Lemma 15, informal). $M$ is defined recursively on the size of $F$. For collections of size $|F| = 1$, the single element of $F$ is mapped to the empty set which is certainly shattered by $F$.

Assume now that $M(F')$ is defined for all collections $F'$ of size $|F'| < |F|$ (i.e. $M$ defined for collection of size $1, \ldots, |F|-1$). We show how to define $M$ on $F$, first with an informal argument.

Suppose we have identified an element $x$ that is in *at least one but not all* sets of $F$. Then we can write $F = F_0 \cup F_1$, dividing $F$ into collections $F_0$ of sets containing $x$, and $F_1$ of sets that do not contain $x$.

Since $|F_0|, |F_1| < |F|$, by induction there exists $M_0 : F_0 \to \{0,1\}^n$ and $M_1 : F_1 \to \{0,1\}^n$ mapping each subcollection to sets shattered by that subcollection. Define $M : F \to \{0,1\}^n$ as follows:

1. For $s \in F_1$, reuse the shattered set, i.e. let $F(s) = F_1(s)$.
2. For $s \in F_0$, if $\forall s' \in F_1, M_1(s') \neq M_0(s)$, we reuse the label for $s$, i.e. $M(s) = M_0(s)$. If $\exists s' \in F_1$ such that $M_1(s') = M_0(s)$, then it must be that $F$ shatters both $M_0(s)$ and $M_0(s) \cup x$. Hence we can assign $M(s) = M_0(s) \cup x$

In the informal construction above, we assumed that $x$ is in some but not all sets of $F$. To define $M$ consistently, we go through all elements in the universe $\{0, 1, \ldots, n-1\}$ in order, and we divide $F$ into those sets that contain the element and those that don't — since these sets are $n$-bit strings, we divide them into those strings with 1 in the first coordinate, and those with 0 in first coordinate). It is possible that one side is empty and the other is all of $F$; in this case, we continue splitting by containment of subsequent elements. When one side is empty, then all labels assigned to sets in the non-empty side are reused. This gives a way to build a complete binary tree of subfamilies starting with $F$ at level 0, and where the $i+1$-st level comes from splitting the previous level by containment of element $i$. $M$ then is built recursively from the leaves up. $\square$

*Proof.* (Lemma 15) We describe how to compute $M(s)$ for a given $s \in F$. Define $T_n$ to be the labeled binary tree with $2^n$ leaves representing $n$-bit strings; level $i$ contains nodes labelled by the $2^i$ binary strings of length $i$, and the children of a node labelled with $s \in \{0,1\}^i$ is $s \cdot 0$ and $s \cdot 1$.

The idea is to go up $T_n$, beginning from the leaf representing set $s$, computing $y^{(n)}, , y^{(n-1)}, \dots, y^1$. This path is unique and has length $n$; denote this path with $P$, and its nodes as $P(n), \dots, P(0)$, from leaf to root (we will interchangeably use $P(i)$ to refer to both a node in $T_n$, and its associated label, a string of length $i$).

1. When we begin at node $P(n)$ we initialize $y$ with the empty set label $y^n = 0^n$
2. Assume we have traversed $T_n$ up to level $i$, i.e. $P(i)$.
3. If the node $P(i)$ is the right child of $P(i-1)$, move up to $P(i-1)$ with $y^{(i-1)} := y^i$.
4. For $P(i)$ that is the left child of $P(i-1)$, denote the right child of $P(i-1)$ (sibling of $P(i)$) and its label as $P'(i)$, and denote by $F_{P'(i)}$ the subcollection of sets in $F$ that have the label $P'(i)$ as its prefix (i.e., those sets that agree on the inclusion/exclusion decisions of the first $i$ elements represented by node $P'(i)$). We check whether $F_{P'(i)}$ also shatters $y^i$, in which case we reuse $y^i$ but flipping bit $i$ to 1, i.e. $y^{(i-1)} := y^i; y_i^{(i-1)} := 1$. Whether $F_{P'(i)}$ (or any particular subfamily corresponding to a node in $T_n$) shatters $y$ can be established in $O(i)$ time: the algorithm checks whether $\forall z \in \{0,1\}^n \exists w \in \{0,1\}^k (C(w) \in F_{P'(i)}) \wedge (z \cap y = C(w) \cap y)$. This can be determined with one call to the $\mathbf{\Sigma_2}$ oracle. Note that $C(w) \in F_{P'(i)}$ can be represented as an $\wedge$ of $i$ equalities.

The following invariant is maintained throughout the algorithm: after completing level $i$, $F_{P(i)}$ shatters $y^i$. This is clearly true at level $n$. With level $i$ completed, if the algorithm assigns $y^{i-1} = y^i$, the invariant is maintained as $y$ does not change. The only way $y^{i-1}$ changes is if $y_1^{i-1} = 1$; this implies both $F_{P(i)}$ and $F_{P'(i)}$ shattered $y^i$.

If $|F| = 1$ (the range of $C$ is one set), assigning the empty set to the lone element is correct. For $s \neq s'$ in the range of $C$, let $P(i)$ be their lowest common ancestor in level $i < n$. Denote its children as $P(i+1)$ and $P'(i+1)$; without loss of generality, $P(i)$ is on the path for $s$ and $P'(i)$ is on the path for $s'$. Consider the algorithm at level $i+1$ when run on both inputs to compute the shattered sets $y$ and $y'$: if $y^{(i+1)} \neq y'^{(i+1)}$, they will remain different for the remainder of the algorithm since at level $j$ we can only change the $j$-th bit. If $y^{(i+1)} = y'^{(i+1)}$, then both $P(i+1)$ and $P'(i+1)$ shatter $y^{(i+1)}$ so the algorithm will set $y^i \neq y'^i$. $\square$

*Proof.* (Lemma 15 $\implies$ Theorem) Given an instance $C$ of SHATTERING, we shall describe an instance $H$ of PIGEONHOLE$^{\mathbf{\Sigma_2}}$ — that is, a hashing circuit with $k$ input gates and $2^k - 1$ possible outputs, whose computation makes calls to a $\mathbf{\Sigma_2}$ oracle — which solves this instance. First, the circuit $H$ determines through an oracle call if $C$ has a collision, and, if it does — two strings $x, y \in [2^k]$ such that $x > y$ and $C(x) = C(y)$ — it computes a perturbation of the identity permutation on $[2^k]$ which exposes the collision: $H$ maps $x$ to $y$, if $x \neq 2^k - 1$ it maps $2^k - 1$ to $x$, and $H$ is the identity on all other strings.

If $C$ has no collision, then on input $x \in [2^k]$ $H$ first computes the distinct set $C(x)$ and then implements the lemma to compute the corresponding set $M(C(x))$ shattered by the $C$ family of sets. If the set $M(C(x))$ if smaller than $d$, the computation ends here and the set is output, in a representation which encodes subsets of $[n]$ in order of increasing size; since by assumption $2^k - 1 \geq \text{BinomSum}(n, d)$, any set smaller than $d$ can be represented. If the set is of size $d$ or larger, then the first $d - 1$ elements of the set are output. This completes the reduction. □

## 4. Discussion and Open Problems

We have introduced a polynomial hierarchy of total functions, whose first couple of levels are populated with interesting computational problems and complexity subclasses with intriguing structural properties. Naturally, a host of questions remain:

- Does this hierarchy behave in similar ways as the polynomial hierarchy — for example, does it collapse upwards? As we have mentioned, the answer to this question is already known, modulo relativization, and it is negative: there are oracles with respect to which **TFNP** = **FP** and yet $\mathbf{TF\Sigma_2} \neq \mathbf{FP^{NP}}$ [3]. We have not explored how this result extends to higher levels.
- A very striking apparent difference between **TFNP** and $\mathbf{TF\Sigma_2}$ is the dearth of diversity in the latter. There are half a dozen apparently distinct complexity subclasses of **TFNP**, corresponding to natural genres of existence proofs. In contrast, in $\mathbf{TF\Sigma_2}$ we have identified **PPEP**, but — despite some intense daydreaming — no other credible class. For example, recall that **PLS** is the class of all problems in **TFNP** reducible to Sink: "Given the circuit representation of a DAG, find a sink" (details of the representation omitted). It is natural to ask — and we did: "How about the problem Source? It is in $\mathbf{TF\Sigma_2}$, of course, but does it define its own class?" It turns out that Source is in **PEPP**...

  For **TFNP**, the invention of new natural subclasses is impeded by the result in [6], establishing, through Herbrand's Theorem, that any such subclass capturing a style of existence proofs in first-order logic must correspond to a *finitary* property of first-order structures: one that is false for infinite structures. How about the logic formulae corresponding to $\mathbf{TF\Sigma_2}$? These would be the so-called Schönfinkel-Bernays formulae (first-order formulae preceded by a sequence of quantifiers of the form $\forall^*\exists^*$), a much studied class in Logic but also in Complexity (it had been known for eight decades that this is a decidable class). Is there a result restricting the usefulness of such formulae in characterizing total search problems, analogous to — but perhaps stricter than — Herbrand's theorem for existentially quantified (Herbrand) formulas?
- Is Complexity complete for **APEPP** under $\mathbf{P^{NP}}$ reductions? This would be a tremendously interesting result. Naturally, any finer completeness result for Complexity would be even more exciting.
- Is Remote Point with large $d$ complete for **APEPP** under $\mathbf{P^{NP}}$ reductions? That would be very interesting as well — especially if it holds true even in the special case in which the code is *linear*.

## Appendix A. $\mathbf{TF\Sigma_i}$ and the Total Function Polynomial Hierarchy

**Definition 16.** (**TFNP**) A relation $R(x, y)$ is in **TFNP** if it is polynomial and **total** (for every $x$ there exists $y$ such that $(x, y)$ is in the relation) and there exists a polynomial time Turing machine $M$ such that $M(x, y)$ accepts iff $R(x, y)$ holds.

**Definition 17.** ($\mathbf{TF\Sigma_2}$) A relation $R(x,y)$ is in $\mathbf{TF\Sigma_2}$ if it polynomial, total, and there exists a polynomial time Turing machine $M$ and polynomial $p(n)$ such that $R(x,y) \iff \forall z \in \{0,1\}^{p(|x|)} M(x,y,z)$ accepts.

**Definition 18.** ($\mathbf{TF\Sigma_{\boldsymbol{i}}}$) A relation $R(x,y)$ is in $\mathbf{TF\Sigma_{\boldsymbol{i}}}$ if it polynomial, total, and there exists a polynomial time Turing machine $M$ and polynomials $p(n)_1, \dots, p(n)_{i-1}$ such that $R(x,y) \iff \forall z_1 \in \{0,1\}^{p(|x|)_1} \exists z_2 \in \{0,1\}^{p(|x|)_2} \forall z_3 \in \{0,1\}^{p(|x|)_3} \cdots M(x,y,z_1,z_2,z_3,\cdots,z_{i-1})$ accepts.

At this point one may ask, what about a $\mathbf{TF\Pi_{\boldsymbol{i}}}$? Could we define total function complexity classes where the first quantifier is an *exists*? It turns out that such a definition results in a complexity class that is polynomial-time reducible to $\mathbf{TF\Sigma_{\boldsymbol{i-1}}}$ and vice versa, and hence, does not capture anything new. In this way, the total function hierarchy is different from its decision-problem analogue, where, by the way of oracles, $\mathbf{\Sigma_{\boldsymbol{i-1}}} \neq \mathbf{\Pi_{\boldsymbol{i}}} \neq \mathbf{\Sigma_{\boldsymbol{i}}}$.

**Proposition 19.** *Let $R(x,y)$ be a polynomial, total relation such that there exists a polynomial time Turing machine $M$ and polynomials $p_1(n), \dots, p_{i-1}(n)$ such that $R(x,y) \iff \exists z_1 \in \{0,1\}^{p_1(|x|)} \forall z_2 \in \{0,1\}^{p_2(|x|)} \exists z_3 \in \{0,1\}^{p_3(|x|)} \cdots M(x,y,z_1,z_2,z_3,\cdots,z_{i-1})$ accepts. Every search problem in $\mathbf{TF\Sigma_{\boldsymbol{i-1}}}$ can expressed with such a relation, and the search problem for any such relation is polynomial-time reducible to $\mathbf{TF\Sigma_{\boldsymbol{i-1}}}$.*

*Proof.* The fact that any search problem in $\mathbf{TF\Sigma_{\boldsymbol{i-1}}}$ can expressed with such a relation (one that starts with $\exists$) is trivial: the relation is the same, and one can reuse the $\mathbf{TF\Sigma_{\boldsymbol{i-1}}}$ Turing Machine $M$, simply by ignoring $z_1$. On the other hand, given a $R(x,y)$ as above, by totality for every $x$ there is a $y$ such that there exists $z$ satisfying the rest of the condition; hence, the relation $R(x,(y,z))$ defined by $R(x,(y,z)) \iff \forall z_2 \in \{0,1\}^{p(|x|)_2} \exists z_3 \in \{0,1\}^{p(|x|)_3} \cdots M(x,y,z_1,z_2,z_3,\cdots)$ is total, and is clearly in $\mathbf{TF\Sigma_{\boldsymbol{i-1}}}$. Hence one can solve $R(x,y)$ with one call to a $\mathbf{TF\Sigma_{\boldsymbol{i-1}}}$oracle, obtaining a pair $(y,z)$ and discarding $z$. ☐

In other words, the total function polynomial hierarchy does not have "two symmetric sides" like the classical one, but is a single tower of classes.

Finally, analogously to the decision problem polynomial hierarchy, the total function polynomial hierarchy can be understood through oracles; $\mathbf{TF\Sigma_{\boldsymbol{i}}} \subseteq \mathbf{TFNP^{\Sigma_{\boldsymbol{i-1}}}}$, and $\mathbf{TFNP^{\Sigma_{\boldsymbol{i-1}}}}$ is polynomial time reducible to $\mathbf{TF\Sigma_{\boldsymbol{i}}}$.

**Theorem 20.** $\mathbf{TF\Sigma_{\boldsymbol{i}}} \subseteq \mathbf{TFNP^{\Sigma_{\boldsymbol{i-1}}}} \leq_T^P \mathbf{TF\Sigma_{\boldsymbol{i}}}$, *where the latter class indicates* $\mathbf{TFNP^{\Sigma_{\boldsymbol{i-1}}}}$*problems where the verifying Turing Machine has access to a* $\mathbf{\Sigma_{\boldsymbol{i-1}}}$ *oracle.*

*Proof.* We present the proof for $\mathbf{TF\Sigma_2}$; the proof for other levels is analogous. The trivial direction is that $\mathbf{TF\Sigma_2} \subseteq \mathbf{TFNP^{\Sigma_1}}$. For a relation $R(x,y)$ with verifying machine $M(x,y,z)$ we define a $\mathbf{TFNP^{\Sigma_1}}$ machine $M'(x,y)$ which issues a single $\mathbf{\Sigma_1}$ query for whether $\forall z M(x,y,z)$ accepts, and outputs the answer. For the other direction, let $R(x,y)$ be a $\mathbf{TFNP^{\Sigma_1}}$ relation with verifying Turing Machine $M^{\mathbf{\Sigma_1}}(x,y)$ which makes at most $p(|x|)$ oracle queries in its computation, each of length at most $p(|x|)$. Define $R'(x,(y,\boldsymbol{a},\boldsymbol{z}))$ where $\boldsymbol{a} \in \{0,1\}^{p(|x|)}, \boldsymbol{z} \in \{0,1\}^{2p(|x|)}$ by whether $\boldsymbol{w} \in \{0,1\}^{2p(|x|)} M'(x,(y,\boldsymbol{a},\boldsymbol{z},\boldsymbol{w}))$ accepts, where $M'$ only accepts if:

(1) $M(x,y)$ is an accepting computation given oracle answers $\boldsymbol{a}$;
(2) if the $i$-th oracle answer in $\boldsymbol{a}$ is a *yes* answer, then the $i$-th string in $\boldsymbol{z}$ is a satisfying assignment to the $i$-th query in the computation $M(x,y)$ (possibly using only a prefix of $\boldsymbol{z}$);
(3) if the $i$-th oracle answer in $\boldsymbol{a}$ is a *no* answer, then the $i$-th string in $\boldsymbol{w}$ does not satisfy the $i$-th query in the computation $M(x,y)$.

Indeed, $R(x, y) \iff \exists \boldsymbol{a}, \boldsymbol{z} R'(x, (y, \boldsymbol{a}, \boldsymbol{z}))$, and the latter is a $\mathbf{TF\Sigma_2}$ relation; to reduce $R$ to $R'$, compute $R'$ and discard $\boldsymbol{a}, \boldsymbol{z}$.

□

## References

[1] N. Alon, R. Panigrahy, and S. Yekhanin, *Deterministic approximation algorithms for the nearest codeword problem*, in Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, I. Dinur, K. Jansen, J. Naor, and J. Rolim, eds., Berlin, Heidelberg, 2009, Springer Berlin Heidelberg, pp. 339–351.

[2] F. Ban, K. Jain, C. H. Papadimitriou, C. Psomas, and A. Rubinstein, *Reductions in PPP*, Inf. Process. Lett., 145 (2019), pp. 48–52.

[3] H. Buhrman, L. Fortnow, M. Koucký, J. D. Rogers, and N. K. Vereshchagin, *Does the polynomial hierarchy collapse if onto functions are invertible?*, Theory Comput. Syst., 46 (2010), pp. 143–156.

[4] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou, *The complexity of computing a nash equilibrium*, SIAM J. Comput., 39 (2009), pp. 195–259.

[5] A. Filos-Ratsikas and P. W. Goldberg, *Consensus halving is ppa-complete*, in Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018, I. Diakonikolas, D. Kempe, and M. Henzinger, eds., ACM, 2018, pp. 51–64.

[6] P. W. Goldberg and C. H. Papadimitriou, *Towards a unified complexity theory of total functions*, in 9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA, A. R. Karlin, ed., vol. 94 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, pp. 37:1–37:20.

[7] P. Hubácek, M. Naor, and E. Yogev, *The journey from NP to TFNP hardness*, in 8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA, C. H. Papadimitriou, ed., vol. 67 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, pp. 60:1–60:21.

[8] I. Komargodski, M. Naor, and E. Yogev, *White-box vs. black-box complexity of search problems: Ramsey and graph property testing*, in 58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017, C. Umans, ed., IEEE Computer Society, 2017, pp. 622–632.

[9] S. R. Mahaney, *Sparse complete sets for NP: solution of a conjecture of berman and hartmanis*, in 21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980, IEEE Computer Society, 1980, pp. 54–60.

[10] N. Sauer, *On the density of families of sets*, J. Combinatorial Theory Ser. A, 13 (1972), pp. 145–147.

[11] C. E. Shannon, *The synthesis of two-terminal switching circuits*, The Bell System Technical Journal, 28 (1949), pp. 59–98.

[12] S. Shelah, *A combinatorial problem; stability and order for models and theories in infinitary languages*, Pacific J. Math., 41 (1972), pp. 247–261.

[13] K. Sotiraki, M. Zampetakis, and G. Zirdelis, *Ppp-completeness with connections to cryptography*, in 59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018, M. Thorup, ed., IEEE Computer Society, 2018, pp. 148–158.

[14] L. G. Valiant, *Graph-theoretic arguments in low-level complexity*, in Mathematical Foundations of Computer Science 1977, 6th Symposium, Tatranska Lomnica, Czechoslovakia, September 5-9, 1977, Proceedings, J. Gruska, ed., vol. 53 of Lecture Notes in Computer Science, Springer, 1977, pp. 162–176.

[15] V. N. Vapnik and A. J. Červonenkis, *The uniform convergence of frequencies of the appearance of events to their probabilities*, Teor. Verojatnost. i Primenen., 16 (1971), pp. 264–279.