

Total Functions in the Polynomial Hierarchy

Robert Kleinberg

Cornell University, Ithaca, NY, USA

Oliver Korten

Columbia University, New York, NY, USA

Daniel Mitropolsky

Columbia University, New York, NY, USA

Christos Papadimitriou

Columbia University, New York, NY, USA

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 α -**PEPP**, which are inside $\mathbf{FP}^{\mathbf{NP}}|_{\text{poly}}$. Higher in the hierarchy, we identify the constructive version of the Sauer-Shelah lemma and the appropriate generalization of **PPP** that contains it, as well as the problem of finding a king in a tournament (a vertex k such that all other vertices are defeated by k , or by somebody k defeated).

2012 ACM Subject Classification Theory of computation \rightarrow Complexity classes

Keywords and phrases total complexity, polynomial hierarchy, pigeonhole principle

Digital Object Identifier 10.4230/LIPIcs.ITCS.2021.44

Funding *Daniel Mitropolsky*: This research was supported in part by a grant from the Columbia-IBM center for Blockchain and Data Transparency, and by JPMorgan Chase & Co. This research was supported in part by NSF Awards CCF1763970 and CCF1910700, and by a research contract with Softbank

Christos Papadimitriou: This research was supported in part by NSF Awards CCF1763970 and CCF1910700, and by a research contract with Softbank.

Acknowledgements Many thanks to Noga Alon for a very enlightening conversation about the union bound, and to Omri Weinstein for an interesting discussion. The authors also thank Ofer Grossman and Eylon Yogev for useful discussions after the pre-print.

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. [9, 10].

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*



© Robert Kleinberg, Oliver Korten, Daniel Mitropolsky, and Christos Papadimitriou; licensed under Creative Commons License CC-BY

12th Innovations in Theoretical Computer Science Conference (ITCS 2021).

Editor: James R. Lee; Article No. 44; pp. 44:1–44:18



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

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 [5, 22, 14].

More recently a logic-inspired class **PTFNP** (for “provable **TFNP**”) was identified containing all of the above classes [11], its definition motivated by the existential proof point of view described above. It was also pointed out in [11] 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 [12].

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 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 **TFΣ₂**. Actually, we shall soon see that there are natural and interesting search problems occupying the third level of the hierarchy. (For the formal definition of **TFΣ_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 **GF₂**; this is a much studied problem [2].

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

Other natural problems in **PEPP** 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 [24] 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 (see the related work subsection for a further discussion of this). 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 $2^{\frac{n}{2}}$.

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 also 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 sparse function exponentially dense.

Is this problem **PEPP**-complete, or otherwise hard in a demonstrable sense? This is an 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 α be a positive quantity, possibly a function of n , and define the class α -**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, **PEPP** = 0-**PEPP**, while many of the problems in **PEPP** discussed are known to belong to α -**PEPP** for some constant α . In particular, we denote 1-**PEPP** by **APEPP**, circuits with twice as many outputs than inputs.

We prove two theorems on α -**PEPP**. First, we establish that the precise value of α is in some sense irrelevant, in that any class α -**PEPP** with α between $\frac{1}{\text{poly}}$ and poly can be reduced to any other such class through **FP^{NP}** reductions (Theorem 7; it is not known whether polynomial time reductions are possible here). Second, it turns out that for $\alpha \geq \frac{1}{\text{poly}}$, for any problem in α -**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 α -**PEPP** is contained in **FZPP^{NP}** – *functional ZPP* (Monte Carlo algorithms) with a satisfiability oracle – and, analogously to Adleman's theorem [1], that α -**PEPP** is contained in **FP^{NP}|poly**, **FP^{NP}** with polynomial advice; we see no reason why **PEPP** = 0-**PEPP** should be so confined.

So far we have been discussing problems and classes in **TFΣ₂**, the next level after **TFNP** of what can be called the polynomial total function hierarchy. It turns out that there are also interesting problems 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^k 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 [19, 21, 25], 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}^{Σ_2} , the pigeonhole principle class when the function mapping pigeons to pigeonholes can use a Σ_2 oracle in its computations.

We present another $\mathbf{TF}\Sigma_3$ problem, which we denote as KING: given a tournament (succinctly described by a circuit), find a vertex v such that every other vertex is reachable from v by a directed path of length one or two. The proof that such a vertex must exist is a local-search argument dating back to the 1950's [15], but the potential function used in the proof is $\#\mathbf{P}$ -hard to compute, hence the KING problem does not evidently belong to any natural subclass of $\mathbf{TF}\Sigma_3$ such as \mathbf{PLS}^{Σ_2} .

Related Work

The difficulty of making existential arguments based on the union bound *constructive* has in fact been a fundamental problem in Complexity Theory and combinatorics for over seven decades. Already in 1947, after Erdős published his paper proving Ramsey lower bounds via the probabilistic method, he recognized the difficulty of matching this with a constructive proof, and offered a \$100 prize to anyone who could do so [8]. Two years after that, Shannon used the union bound to give a nonconstructive proof that some functions require exponential size circuits, and also noted the difficulty in finding constructive proofs of size lower bounds for explicit functions, comparing it to the difficulty of proving that particular numbers are transcendental [20]. At this time, “constructive” was a rather informal concept, but a few decades later Complexity Theory offered us a plausible definition: a constructive proof is an algorithm that constructs an object with the desired properties from scratch, in time polynomial in the size of the object. Over time, an important research tradition in Complexity Theory has developed around such explicit construction problems, pertaining mostly to the construction of computational devices (pseudorandom generators, randomness extractors, exponentially hard Boolean functions in the worst or average case, etc.) whose existence is guaranteed by the union bound. Many celebrated results in this domain compare the difficulty of such explicit construction problems through, essentially, reductions [24, 17, 13, 6, 23]. In particular, already in 1977 Les Valiant [24] showed that an explicit construction of a rigid matrix would imply an explicit Boolean function requiring shallow circuits of superlinear size – a reduction between two explicit construction problems whose corresponding existence proofs rely on the union bound. Next, Nisan and Wigderson established in 1995 that an explicit pseudorandom generator can be constructed in polynomial time, given an explicit construction of a truth table which is hard to approximate for exponentially large circuits, and Impagliazzo and Wigderson [13] showed a decade later that such hard-to-approximate truth tables can in fact be constructed in polynomial time from truth tables which are very hard to compute in the worst case. More recently, an equivalence has been shown between the explicit construction of randomness dispersers and the construction of Ramsey graphs, and a significant body of work has been devoted to deriving more efficient constructions of such objects [8, 6].

Many results in this realm can be reformulated as reductions between total function problems in a particular subclass of **APEPP**, which could be called **SAPEPP** (for “sparse **APEPP**”). Every problem in **SAPEPP** is defined by a polynomial-time Turing machine-computable function $M : \{0, 1\}^* \mapsto \{0, 1\}^*$ such that for all $x \in \{0, 1\}^*$, $|M(x)| = f(|x|)$, where $f(n) > n$. The total search problem associated with M asks: given n in unary, find a bit string y of length $f(n)$ such that for all x with $|x| = n$, $M(x) \neq y$. That **SAPEPP** is a subset of **APEPP** follows easily from the basic fact that any fixed polynomial-time Turing machine with a given input length can be rendered as a Boolean circuit in time polynomial in the input length. For a concrete example, for the problem of explicitly constructing a truth table for a function $[N] \rightarrow \{0, 1\}$ which requires circuits of size greater than $\frac{N}{3 \log N}$, M is a machine which transforms concisely encoded circuits of size $\frac{N}{3 \log N}$ into truth tables, and $f(n) = n + 1$. The associated function problem in **SAPPEP**, which could be called **SPARSE COMPLEXITY**, seeks the explicit construction of a hard Boolean function; a polynomial-time solution for this problem would imply, among other tectonic consequences, that $\mathbf{P} = \mathbf{BPP}$ [13].

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.** $\mathbf{FNP} \subseteq \mathbf{PEPP}$.

Proof. We prove that **SAT** can be reduced to **EMPTY**. Let ϕ 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 ϕ . 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 ϕ : If $s \neq 1^n$ then ϕ is satisfiable and s satisfies it; otherwise, ϕ is unsatisfiable. ◀

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 $\mathbf{FP}^{\mathbf{NP}}$ 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 α -**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. The following problem motivated by Shannon’s construction has been recently identified: 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. This is known as the **REMOTE POINT** problem, studied extensively in Complexity and Cryptography [2, 3, 4].

► **Proposition 2.** **REMOTE POINT** is in **PEPP**.

Proof. Its proof of totality is an application of the union bound. ◀

In fact, **REMOTE POINT** is strictly speaking **PEPP**-complete, because any instance of **EMPTY** is also an instance of **REMOTE POINT** with $d = 1$.

Another natural problem lying in **PEPP** comes from the fact that graphs of bounded degree have logarithmic diameter. One way to capture this is through the problem **REMOTE VERTEX**: given a directed graph on $[N]$ with vertices of outdegree at most 2, specified by circuits $C_L, C_R : [N] \rightarrow [N]$ which output the “left” and “right” successors of a given node respectively, find a vertex whose distance from the all-zero vertex is at least $\log N$.

► **Proposition 3.** **REMOTE VERTEX** is in **PEPP**.

Proof. Consider the circuit that takes as input a string $s \in \{L, R\}^*$ of length $0 \leq |s| \leq \log N - 1$, and outputs the vertex we arrive at by starting with the all-zero vertex and repeatedly applying C_L or C_R to the current input based on the next character in s . This circuit maps each path of length at most $\log N - 1$ starting at the all-zero vertex to its endpoint, a vertex in $[N]$. As there are at most $N - 1$ such paths, this is a valid instance of **EMPTY**, whose solution is indeed a remote vertex. ◀

One can define several variants of **REMOTE VERTEX** using other implicit representations of graphs, for example the representations for undirected and directed graphs used to define the canonical problems of bounded degree for **PPA** and **PPAD** [18]. Both of these variants reduce to the version of **REMOTE VERTEX** defined above.

Next we introduce two problems in **PEPP** capturing two other classical applications of the union bound. In δ -**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 δ -rigid: it cannot be turned into a matrix of rank $\leq \delta n$ by changing n^δ 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. [16]; alternatively, if n is given in binary, then the problem is even more ill-posed since an exponentially long output is required².

To see that δ -**RIGID MATRIX COMPLETION** reduces to **EMPTY**, let $N = 2^{n^2 - n \log n}$ denote the total number of possible completions of the matrix, and let M denote the number of pairs (L, S) where L is a $n \times n$ matrix of rank at most δn and S is a matrix that has at

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.

most n^δ ones per row. There are at most $2^{2\delta n^2}$ choices for L and at most $\binom{n}{n^\delta}^n < 2^{n^{1+\delta} \log n}$ choices for S , hence $M < 2^{2\delta n^2 + n^{1+\delta} \log n}$. Now consider the circuit $C : [M] \mapsto [N]$ that takes an input in $[M]$, interprets it as the encoding of a pair (L, S) , computes the sum $L + S$, and outputs the element of $[N]$ encoding this sum if it is a completion of the given matrix, or else it outputs an arbitrary element of $[N]$, for example the element representing the trivial matrix completion that sets all remaining entries of the given matrix to zero. Note that $M < N$ as long as n is large enough that $(1 - 2\delta)n^2 > (n^{1+\delta} + n) \log n$. Any element of $[N]$ not in the range of C must be the encoding of a matrix completion that cannot be expressed in the form $L + S$, hence is rigid.

RAMSEY-ERDŐS COMPLETION is the problem of finding an n -node graph with no independent set of size $k = 4\lceil \log n \rceil$ and no clique of this size, given the connectivity of $\ell = \lceil \log n \rceil$ nodes in the graph.

There are $N = 2^{\binom{n-\ell}{2}}$ completions of the given graph. The completions containing a clique or independent set of size k are parameterized by tuples (A, b, x) where A is a vertex set of size k , b is a bit indicating whether A forms a clique or independent set, and x is a bitstring indicating which edges belong to the completion, excluding those having both endpoints in A and those having one endpoint among the ℓ vertices whose connectivity is given in the problem input. There are $\binom{n}{k}$ possible values for A , 2 possible values for b , and at most $2^{\binom{n-\ell}{2} - \binom{k-\ell}{2}}$ possible values for x , hence at most $M = \binom{n}{k} 2^{1 + \binom{n-\ell}{2} - \binom{k-\ell}{2}}$ possible values for the triple (A, b, x) . When $k = 4\ell$ and $\ell = \lceil \log n \rceil$, it follows from a standard calculation that $M < N$.

We have established this result:

► **Proposition 4.** δ -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 Circuit Complexity 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 [20] 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 $v = \lceil \log n \rceil + 1$ 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 $\lceil \log n \rceil$, which when its inputs are the bits $b_1, \dots, b_{\lceil \log n \rceil}$, 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 ℓ 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.

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 5.** *COMPLEXITY is in PEPP.*

2.2 Wasteful counting and α -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 α -**PEPP**, parameterized by a function $\alpha : \mathbb{N} \rightarrow \mathbb{R}_+$.

The complexity class α -**PEPP** is defined to consist of all total functions that are polynomial-time reducible to the following α -**EMPTY** problem. An instance of α -**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 6.** *The problems δ -RIGID MATRIX COMPLETION, RAMSEY-ERDŐS COMPLETION, and COMPLEXITY all belong to APEPP.*

Proof. Above, we presented reductions from δ -RIGID MATRIX COMPLETION, RAMSEY-ERDŐS COMPLETION, and COMPLEXITY to **EMPTY** with the following parameters.

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

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

The ratio N/M exceeds 2 when $\delta < \frac{1}{3}$, $n > 125$.

- For a RAMSEY-ERDŐS COMPLETION instance with n vertices, the reduction yields a circuit of size $\text{poly}(n)$ mapping $[M]$ to $[N]$, where $M = \binom{n}{4^{\lceil \log n \rceil}} 2^{1 + \binom{n - \lceil \log n \rceil}{2} - \binom{3 \lceil \log n \rceil}{2}}$, $N = 2^{\binom{n - \lceil \log n \rceil}{2}}$,

$$\frac{N}{M} = \frac{2^{\binom{3 \lceil \log n \rceil}{2} - 1}}{\binom{n}{4^{\lceil \log n \rceil}}}.$$

The ratio N/M exceeds 2 when $n > 8$.

- For a COMPLEXITY instance of length n , the reduction yields a circuit of size $\text{poly}(n)$ mapping $[M]$ to $[N]$, where $M = 2^n$, $N = 2^{2^{\lceil \log n \rceil + 1}}$,

$$N/M = 2^{2^{\lceil \log n \rceil + 1} - n}.$$

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 7.** *For any function $\alpha(n) > \frac{1}{\text{poly}(n)}$, we have α -**PEPP** \subseteq **FZPP**^{NP} \subseteq **FP**^{NP} | poly where **FZPP** denotes functional **ZPP**.*

Proof. First we show α -EMPTY \in **FZPP**^{NP}. 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 $(1 + \alpha)^{-k} < e^{-n}$. Consider an algorithm that randomly samples the y_i , and queries an **NP** oracle whether there exists $x \in [M]$ such that $C(x) = y_i$, and outputs the first y_i for which the oracle confirms no such x exists: this shows α -EMPTY \in **FZPP**^{NP}.

To see the second inclusion of the theorem, suppose an algorithm that uses randomness r , $|r| \leq \text{poly}(n)$ fails with probability $< e^{-n}$ for any length n input. By the union bound, a random sample of r has positive probability of containing a valid solution to *every* length- n input instance. In fact, this probability is greater than $1 - 2^n \cdot e^{-n}$. Hence, there exists an advice string such that for every length- n instance, the algorithm finds a correct output. ◀

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

³ Actually, the ratio is greater than or equal to 2 for all n , but it is equal to 2 when $n + 1$ is a power of 2. Since the definition of α -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 .

44:10 Total Functions in the Polynomial Hierarchy

► **Theorem 8.** *If $\frac{1}{\text{poly}(n)} \leq \alpha(n) \leq 2^{\text{poly}(n)}$, then α -PEPP and APEPP are equivalent under $\mathbf{FP}^{\mathbf{NP}}$ reductions.*

Proof. For any positive integers N, k , let $T : [N^k] \rightarrow [N]^k$ 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 $\beta(n), \gamma(n) : \mathbb{N} \rightarrow \mathbb{R}_+$ are any two functions such that

$$k(n) \stackrel{\Delta}{=} \lceil \log_{1+\beta(n)}(1 + \gamma(n)) \rceil \leq \text{poly}(n).$$

We can reduce β -EMPTY to γ -EMPTY as follows. Given a length- n bitstring describing a circuit that computes a function $C : [M] \rightarrow [N]$, let $k = k(n)$ and construct the description of a circuit that computes the function $C' : [M^k] \rightarrow [N^k]$ 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 γ -EMPTY and applying the function T , we obtain a k -tuple (y_1, \dots, y_k) that is not in the range of the function $C^k : [M]^k \rightarrow [N]^k$. Now, given an \mathbf{NP} oracle, we can proceed as in the proof of Theorem 7 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 \mathbf{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 $C^k(x_1, \dots, x_k)$ would equal (y_1, \dots, y_k) contradicting our assumption that (y_1, \dots, 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 β -EMPTY instance.

We have shown a $\mathbf{FP}^{\mathbf{NP}}$ reduction from β -PEPP to γ -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 α -PEPP to APEPP is obtained by taking $\beta = \alpha$ and $\gamma \equiv 1$, and a reduction from APEPP to α -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 9.** *A family of sets over some finite universe, $F = \{s_1, s_2, \dots\}$, 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 10** (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 11** (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 12.** Let $\text{BinomSum}(n, d)$ denote $\sum_{i=0}^{d-1} \binom{n}{i}$.

► **Definition 13.** In the SHATTERING problem, we are given as inputs parameters n, d , and $k > \log(\text{BinomSum}(n, d))$, and a circuit computing a function $C : \{0, 1\}^k \rightarrow \{0, 1\}^n$, 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 14.** SHATTERING is in $\mathbf{TF}\Sigma_3$.

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

1. checks that $k > \log(\text{BinomSum}(n, d))$;
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 \mathbf{PPP} that lies within $\mathbf{TF}\Sigma_3$:

► **Theorem 15.** SHATTERING is in \mathbf{PPP}^{Σ_2}

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

► **Lemma 16.** Using a Σ_2 oracle, one can compute a polynomial time function M mapping distinct sets in F to distinct sets shattered by F .

Proof (Lemma 16, 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 $1, \dots, |F| - 1$). 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 $M_0 : F_0 \rightarrow \{0, 1\}^n$ and $M_1 : F_1 \rightarrow \{0, 1\}^n$ mapping each subcollection to sets shattered by that subcollection. Define $M : F \rightarrow \{0, 1\}^n$ as follows:

44:12 Total Functions in the Polynomial Hierarchy

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 $\{0, 1, \dots, n-1\}$ 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. ◀

Proof (Lemma 16). 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 Σ_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 $y^i \neq y'^i$. ◀

Proof (Lemma 16 \implies Theorem). Given an instance C of SHATTERING, we shall describe an instance H of PIGEONHOLE $^{\Sigma_2}$ – that is, a hashing circuit with k input gates and $2^k - 1$ possible outputs, whose computation makes calls to a Σ_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 $M(C(x))$ is 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. \blacktriangleleft

It turns out that $\text{TF}\Sigma_3$ contains another natural problem aside from SHATTERING, based on a simple fact in graph theory dating back to the 1950's [15].

► **Definition 17.** A vertex v in a digraph G is called a king if every vertex can be reached from v by a path of length at most 2.

► **Definition 18.** A digraph G is called a tournament if for every pair of distinct vertices $u, v \in G$, exactly one of the directed edges (u, v) or (v, u) is present in G .

► **Lemma 19.** Every tournament has a king [15].

Proof. Given a vertex v , will say that the *court* of v is the set of vertices reachable from v in exactly one step, and the *domain* of v is the set of vertices reachable from v in exactly 1 or 2 steps. By definition, a king is a vertex whose domain contains all other vertices. Starting with an arbitrary vertex v , we can now locate a king as follows: if the domain of v contains all other vertices then v is a king and we are done. Otherwise there exists a $u \neq v$ outside the domain of v , and we continue our search from u . To see that this iterative process terminates, note that at each step the size of the court of our current vertex strictly increases: if u is outside the domain of v , u 's court must contain at least every element of v 's court, and must also contain v . \blacktriangleleft

This gives rise to the following total search problem KING: given a tournament, represented as a circuit $C : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$ determining if the directed edge (u, v) is present, find either a king or a pair of inputs $x \neq y$ such that $C(x, y) = C(y, x)$ (proving that C does not define a tournament). Since KING seeks a vertex k such that for all other vertices v there is an intermediate vertex i which is reachable from k in one or zero steps, and such that (i, v) is an edge, this alternation of quantifiers implies the following:

► **Proposition 20.** KING is in $\text{TF}\Sigma_3$.

The proof of Lemma 19 showing the totality of KING is eerily reminiscent of familiar totality arguments for the class PLS. In PLS, the input is an implicitly defined directed graph, along with a polynomial time “potential function” mapping vertices to natural numbers, such that the potential is strictly increasing along edges. The goal is to find a vertex with no outgoing edges, and the potential function gives us a syntactic guarantee that the underlying graph is acyclic and therefore that such a vertex must exist. In the case of KING, we have a similar situation. Given a circuit defining a tournament, we can generate another implicitly

defined graph which has a directed edge (v, u) if and only if u is outside the domain of v , and a potential function which assigns each vertex a potential equal to the size of its court. A vertex with no outgoing edges in this graph will be a king, and since the potential increases along edges, such a vertex must exist. If both the “edge function” (the function finding an outgoing edge of a vertex or telling us that none exists) and the potential function were computable in Σ_2 , this would place KING in \mathbf{PLS}^{Σ_2} . However, it seems that only the edge function has this property: computing the potential requires solving a rather generic counting problem. This leaves us in a curious situation, where the proof that a solution exists uses an implicit potential function, but directly computing the potential is seemingly harder than finding a solution. We do not know of another natural total function with this property.

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 the total function 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 $\mathbf{TFNP} = \mathbf{FP}$ and yet $\mathbf{TF}\Sigma_2 \neq \mathbf{FP}^{\mathbf{NP}}$ [7]. We have not explored how this result extends to higher levels. After a preprint of this article was posted online, Ofer Grossman showed us a sketch of an argument that, if $\mathbf{TFNP} = \mathbf{TF}\Sigma_2$, then the decisional polynomial hierarchy does collapse (Ofer Grossman, personal communication). This can be shown to imply that the total function hierarchy collapses.
- A very striking apparent difference between \mathbf{TFNP} and $\mathbf{TF}\Sigma_2$ is the dearth of diversity in the latter. There are half a dozen apparently distinct complexity subclasses of \mathbf{TFNP} , corresponding to natural genres of existence proofs. In contrast, in $\mathbf{TF}\Sigma_2$ we have identified \mathbf{PPEP} , but – despite some intense daydreaming – no other credible class. For example, recall that \mathbf{PLS} is the class of all problems in \mathbf{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 \mathbf{PEPP} ... For \mathbf{TFNP} , the invention of new natural subclasses is impeded by the result in [11], 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 since the 1930s 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 \mathbf{APEPP} under $\mathbf{P}^{\mathbf{NP}}$ 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 \mathbf{APEPP} under $\mathbf{P}^{\mathbf{NP}}$ reductions? That would be very interesting as well – especially if it holds true even in the special case in which the code is *linear*.

- The **SAPEPP** class, as defined in the related work subsection, encompasses some of the most important problems in Complexity Theory. Sparsity complicates proving these problems intractable, and yet we know already some fascinating reductions between them. Does **SAPEPP** have natural complete problems? Is **SPARSE COMPLEXITY** complete for it?
- The problem **KING** encompasses a novel aspect of total functions related to local optimality. Problems in the class **PLS** are presented in terms of an implicit DAG defined in terms of an edge function and a potential function. Higher in the hierarchy, **KING** is defined only in terms of an edge function, while the DAG property is established through an *extraneous proof*, that is, a proof not encoded in the instance's description in terms of an explicit potential function. Unless $\#\mathbf{P}$ is in the polynomial hierarchy, **KING** does not appear to belong in \mathbf{PLS}^{Σ_2} : it is a problem in $\mathbf{TF}\Sigma_3$ whose totality follows from a local optimality argument, and yet one that is *sui generis*, in a class in and by itself. Are there such problems in **TFNP**?

References

- 1 Leonard Adleman. Two theorems on random polynomial time. In *19th Annual Symposium on Foundations of Computer Science*, pages 75–83, 1978. doi:10.1109/SFCS.1978.37.
- 2 Noga Alon, Rina Panigrahy, and Sergey Yekhanin. Deterministic approximation algorithms for the nearest codeword problem. In Irit Dinur, Klaus Jansen, Joseph Naor, and José Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 339–351, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- 3 Vikraman Arvind and Srikanth Srinivasan. Circuit lower bounds, help functions, and the remote point problem, 2009. arXiv:0911.4337.
- 4 Vikraman Arvind and Srikanth Srinivasan. The remote point problem, small bias space, and expanding generator sets. In *27th International Symposium on Theoretical Aspects of Computer Science - STACS 2010*, pages 59–70, 2010.
- 5 Frank Ban, Kamal Jain, Christos H. Papadimitriou, Christos-Alexandros Psomas, and Aviad Rubinfeld. Reductions in PPP. *Inf. Process. Lett.*, 145:48–52, 2019. doi:10.1016/j.ipl.2018.12.009.
- 6 Boaz Barak, Anup Rao, Ronen Shaltiel, and Avi Wigderson. 2-source dispersers for sub-polynomial entropy and ramsey graphs beating the frankl-wilson construction. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '06, page 671–680, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1132516.1132611.
- 7 Harry Buhrman, Lance Fortnow, Michal Koucký, John D. Rogers, and Nikolai K. Vereshchagin. Does the polynomial hierarchy collapse if onto functions are invertible? *Theory Comput. Syst.*, 46(1):143–156, 2010. doi:10.1007/s00224-008-9160-8.
- 8 Gil Cohen. *Two-Source Dispersers for Polylogarithmic Entropy and Improved Ramsey Graphs*, page 278–284. Association for Computing Machinery, New York, NY, USA, 2016. doi:10.1145/2897518.2897530.
- 9 Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a nash equilibrium. *SIAM J. Comput.*, 39(1):195–259, 2009. doi:10.1137/070699652.
- 10 Aris Filos-Ratsikas and Paul W. Goldberg. Consensus halving is ppa-complete. 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 51–64. ACM, 2018. doi:10.1145/3188745.3188880.

- 11 Paul W. Goldberg and Christos H. Papadimitriou. Towards a unified complexity theory of total functions. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, volume 94 of *LIPIcs*, pages 37:1–37:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ITCS.2018.37.
- 12 Pavel Hubáček, Moni Naor, and Eylon Yogev. The journey from NP to TFNP hardness. In Christos H. Papadimitriou, editor, *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, volume 67 of *LIPIcs*, pages 60:1–60:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ITCS.2017.60.
- 13 Russell Impagliazzo and Avi Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, STOC '97*, page 220–229, New York, NY, USA, 1997. Association for Computing Machinery. doi:10.1145/258533.258590.
- 14 Ilan Komargodski, Moni Naor, and Eylon Yogev. White-box vs. black-box complexity of search problems: Ramsey and graph property testing. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 622–632. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.63.
- 15 H. G. Landau. On dominance relations and the structure of animal societies: III The condition for a score structure. *The bulletin of mathematical biophysics*, 15(2):143–148, June 1953. doi:10.1007/BF02476378.
- 16 Stephen 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*, pages 54–60. IEEE Computer Society, 1980. doi:10.1109/SFCS.1980.40.
- 17 Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994. doi:10.1016/S0022-0000(05)80043-1.
- 18 Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994. doi:10.1016/S0022-0000(05)80063-7.
- 19 N. Sauer. On the density of families of sets. *J. Combinatorial Theory Ser. A*, 13:145–147, 1972. doi:10.1016/0097-3165(72)90019-2.
- 20 C. E. Shannon. The synthesis of two-terminal switching circuits. *The Bell System Technical Journal*, 28(1):59–98, 1949.
- 21 Saharon Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages. *Pacific J. Math.*, 41:247–261, 1972. URL: <http://projecteuclid.org/euclid.pjm/1102968432>.
- 22 Katerina Sotiraki, Manolis Zampetakis, and Giorgos Zirdelis. Ppp-completeness with connections to cryptography. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 148–158. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00023.
- 23 Luca Trevisan. Extractors and pseudorandom generators. *J. ACM*, 48(4):860–879, July 2001. doi:10.1145/502090.502099.
- 24 Leslie G. Valiant. Graph-theoretic arguments in low-level complexity. In Jozef Gruska, editor, *Mathematical Foundations of Computer Science 1977, 6th Symposium, Tatranska Lomnica, Czechoslovakia, September 5-9, 1977, Proceedings*, volume 53 of *Lecture Notes in Computer Science*, pages 162–176. Springer, 1977. doi:10.1007/3-540-08353-7_135.
- 25 V. N. Vapnik and A. Ja. Červonenkis. The uniform convergence of frequencies of the appearance of events to their probabilities. *Teor. Verojatnost. i Primenen.*, 16:264–279, 1971.

A Total Function Polynomial Hierarchy

► **Definition 21 (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 22 (TF Σ_2).** A relation $R(x, y)$ is in **TF Σ_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 23 (TF Σ_i).** A relation $R(x, y)$ is in **TF Σ_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 $R(x, y) \iff \forall z_1 \in \{0, 1\}^{p(|x|)_1} \exists z_2 \in \{0, 1\}^{p(|x|)_2} \forall z_3 \in \{0, 1\}^{p(|x|)_3} \dots M(x, y, z_1, z_2, z_3, \dots, z_{i-1})$ accepts.

At this point one may ask, what about a **TF Π_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 **TF Σ_{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, $\Sigma_{i-1} \neq \Pi_i \neq \Sigma_i$.

► **Proposition 24.** 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 $R(x, y) \iff \exists z_1 \in \{0, 1\}^{p_1(|x|)} \forall z_2 \in \{0, 1\}^{p_2(|x|)} \exists z_3 \in \{0, 1\}^{p_3(|x|)} \dots M(x, y, z_1, z_2, z_3, \dots, z_{i-1})$ accepts. Every search problem in **TF Σ_{i-1}** can be expressed with such a relation, and the search problem for any such relation is polynomial-time reducible to **TF Σ_{i-1}** .

Proof. The fact that any search problem in **TF Σ_{i-1}** can be expressed with such a relation (one that starts with \exists) is trivial: the relation is the same, and one can reuse the **TF Σ_{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 $R(x, (y, z)) \iff \forall z_2 \in \{0, 1\}^{p(|x|)_2} \exists z_3 \in \{0, 1\}^{p(|x|)_3} \dots M(x, y, z_1, z_2, z_3, \dots)$ is total, and is clearly in **TF Σ_{i-1}** . Hence one can solve $R(x, y)$ with one call to a **TF Σ_{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; **TF $\Sigma_i \subseteq \text{TFNP}^{\Sigma_{i-1}}$** , and **TFNP $^{\Sigma_{i-1}}$** is polynomial time reducible to **TF Σ_i** .

► **Theorem 25.** **TF $\Sigma_i \subseteq \text{TFNP}^{\Sigma_{i-1}} \leq_T^P \text{TF}\Sigma_i$** , where the latter class indicates **TFNP $^{\Sigma_{i-1}}$** problems where the verifying Turing Machine has access to a Σ_{i-1} oracle.

Proof. We present the proof for **TF Σ_2** ; the proof for other levels is analogous. The trivial direction is that **TF $\Sigma_2 \subseteq \text{TFNP}^{\Sigma_1}$** . For a relation $R(x, y)$ with verifying machine $M(x, y, z)$ we define a **TFNP $^{\Sigma_1}$** machine $M'(x, y)$ which issues a single Σ_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 **TFNP $^{\Sigma_1}$** relation with verifying Turing Machine $M^{\Sigma_1}(x, y)$ which makes at most $p(|x|)$ oracle queries in its computation, each of length at most $p(|x|)$. Define $R'(x, (y, \mathbf{a}, \mathbf{z}))$ where $\mathbf{a} \in \{0, 1\}^{p(|x|)}$, $\mathbf{z} \in \{0, 1\}^{2p(|x|)}$ by whether $\mathbf{w} \in \{0, 1\}^{2p(|x|)} M'(x, (y, \mathbf{a}, \mathbf{z}, \mathbf{w}))$ accepts, where M' only accepts if:

44:18 Total Functions in the Polynomial Hierarchy

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

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