Does the polynomial hierarchy collapse if onto functions are invertible?

Buhrman, H.; Fortnow, L.; Koucký, M.; Rogers, J.D.; Vereshchagin, N.

DOI
10.1007/s00224-008-9160-8
Publication date
2010
Document Version
Final published version
Published in
Theory of Computing Systems

Link to publication

Citation for published version (APA):
Buhrman, H., Fortnow, L., Koucký, M., Rogers, J. D., & Vereshchagin, N. (2010). Does the polynomial hierarchy collapse if onto functions are invertible? *Theory of Computing Systems*, *46*(1), 143-156. https://doi.org/10.1007/s00224-008-9160-8

Theory Comput Syst (2010) 46: 143–156
DOI 10.1007/s00224-008-9160-8

Does the Polynomial Hierarchy Collapse if Onto Functions are Invertible?

Harry Buhrman · Lance Fortnow · Michal Koucký · John D. Rogers · Nikolay Vereshchagin

Published online: 17 December 2008

Abstract The class TFNP, defined by Megiddo and Papadimitriou, consists of multivalued functions with values that are polynomially verifiable and guaranteed to exist. Do we have evidence that such functions are hard, for example, if TFNP is computable in polynomial-time does this imply the polynomial-time hierarchy collapses? By computing a multivalued function in deterministic polynomial-time we mean on every input producing one of the possible values of the function on that input.

We give a relativized negative answer to this question by exhibiting an oracle under which TFNP functions are easy to compute but the polynomial-time hierarchy is infinite. We also show that relative to this same oracle, $\mathrm{P} \neq \mathrm{UP}$ and $\mathrm{TFNP}^{\mathrm{NP}}$ functions are not computable in polynomial-time with an NP oracle.

Keywords Computational complexity · Polynomial-time hierarchy · Multi-valued functions · Kolmogorov complexity

H. Buhrman
CWI and University of Amsterdam, Amsterdam, The Netherlands
e-mail: buhrman@cwi.nl

L. Fortnow
University of Chicago, Chicago, USA
e-mail: fortnow@cs.uchicago.edu

M. Koucký
Institute of Mathematics, Academy of Sciences of the Czech Republic, Prague, Czech Republic
e-mail: koucky@math.cas.cz

J.D. Rogers (✉)
DePaul University, Chicago, USA
e-mail: jrogers@cs.depaul.edu

N. Vereshchagin
Lomonosov Moscow State University, Moscow, Russia
e-mail: ver@mech.math.msu.su

1 Introduction

Many problems studied in complexity theory are NP decision problems: for a polynomial time computable binary relation $R(x, y)$ and a polynomial p, given any string x find out whether there is a string y of length at most $p(|x|)$ such that $R(x, y)$ holds. Usually, we are interested not only in finding out whether there is such y but also in finding such y in the case it exists. Problems of this kind are called NP *search problems*.

If an algorithm solves an NP search problem then it certainly solves the corresponding decision problem, but not the other way around. However, for every NP search problem S it is easy to construct an NP decision problem D which S reduces to, using a binary search. Thus every NP complete decision problem is equivalent to the corresponding NP search problem. Under the assumption that for example double exponential time does not equal Nondeterministic double exponential time it is easy to show that there are search problems in NP that are harder than the corresponding search problem [2].

The situation differs however if we are given the promise that for every x there is such a y. The class of such search problems was defined by Megiddo and Papadimitriou [10] and called TFNP, the abbreviation reads "Total Functions in NP". A (multi-valued) function from TFNP is specified by a polynomial time computable binary relation $R(x, y)$ and a polynomial p such that for every string x there is a string y of length at most $p(|x|)$ such that $R(x, y)$ holds. It maps x to the set of y's of length at most $p(|x|)$ such that $R(x, y)$ holds, where in the sequel by a value of the function on x we mean any of the y's. To compute a TFNP function will mean to solve the corresponding search problem: given a x find a y of length at most $p(|x|)$ such that $R(x, y)$ holds. This class of problems includes Factoring, finding a Nash Equilibrium, finding solutions of Sperner's Lemma, finding solutions to Ramsey theorem, and finding collisions of hash functions.

Note that NP decision problems corresponding to TFNP search problems are always trivial. On the other hand, TFNP search problems might be hard, as the above examples show.

Fenner, Fortnow, Naik and Rogers [3] consider the hypothesis, which they called "Q", that for every function in TFNP there is a polynomial-time procedure that will output a value of that function. That is, Proposition Q states that for every R and p defining a TFNP-function there is a polynomial time computable function f such that $R(x, f(x))$ holds for all x.

If some of the above listed problems cannot be solved in polynomial time then Proposition Q is false. As all those problems seem to be hard, it is plausible that Proposition Q is false. In this paper we address the following questions: is there any evidence that Proposition Q is false, and how does Proposition Q relate to other similar hypotheses in Complexity Theory?

Fenner et al. showed that Proposition Q is equivalent to a number of different hypotheses including:

- Given an NP machine M with $L(M) = \Sigma^*$, there is a polynomial-time computable function f such that $f(x)$ is an accepting computation of $M(x)$.

- Given an honest onto polynomial-time computable function g there is a polynomial-time computable function f such that $g(f(x)) = x$. (A function $g(x)$ is called honest if there is a polynomial $p(n)$ such that $|x| \leq p(|g(x)|)$ for all x.)
- For all polynomial-time computable subsets S of SAT there is a polynomial-time computable function f such that for all ϕ in S, $f(\phi)$ is a satisfying assignment to ϕ.
- For all NP machines M such that $L(M) = \text{SAT}$, there is a polynomial-time computable function f such that for every ϕ in SAT and accepting path c of $M(\phi)$, $f(\phi, c)$ is a satisfying assignment of ϕ.

Here are the known relations between Proposition Q and similar hypotheses. (1) As TFNP is a sub-class of the class of all NP search problems, P = NP implies Proposition Q. (2) Proposition Q implies that any pair of disjoint coNP-sets is P-separable (which implies that NP $\cap$ coNP = P). A P-separator for two disjoint sets A and B is a set $S \in \text{P}$ such that $A \subseteq S$ and $B \subseteq \overline{S}$. Indeed, for a pair A, B of disjoint coNP-sets, consider the following TFNP search problem: given an x find a witness that $x \notin A$ or a witness that $x \notin B$. Since A and B are disjoint there will be a witness for every x. If Proposition Q holds for every x a witness can be found in polynomial time. Define S as follows: $x \in S$ if the witness found on input x indicates that $x \notin B$ and $x \in \overline{S}$ otherwise. It is not hard to see that S is a polynomial time separator for A and B.

Fenner et. al. ask whether we can draw any stronger complexity collapses from Q, in particular whether Q implies that the polynomial-time hierarchy collapses. We give a relativized negative answer to this question by exhibiting an oracle relative to which Q holds and the polynomial-time hierarchy is infinite. In particular, there is no relativizable proof that Proposition Q implies P = NP.

Proposition Q naturally generalizes to other levels of the polynomial hierarchy. Namely, define the class $\text{TF}\Sigma_k^p$ as follows. A $\text{TF}\Sigma_k^p$-function is specified by a binary relation $R(x, y)$ computable in polynomial time with an oracle from Σ_{k-1}^p and a polynomial p such that for every string x there is a string y of length at most $p(|x|)$ such that $R(x, y)$ holds. For $k \geq 1$ we will label Σ_k^pQ the statement.

Proposition Σ_k^pQ *For every R and p defining a $\text{TF}\Sigma_k^p$-function there is a function f computable in polynomial time with an oracle from Σ_{k-1}^p such that $R(x, f(x))$ holds for all x.*

For $k = 1$ we obtain the class TFNP and Proposition Q.

Proposition Σ_k^pQ implies that $\text{P}^{\Sigma_{k-1}^p} = \Sigma_k^p \cap \Pi_k^p$ and is implied by $\Sigma_{k-1}^p = \Sigma_k^p$. A natural question is whether, similar to the implication

$$\Sigma_{k-1}^p = \Sigma_k^p \quad \Longrightarrow \quad \Sigma_k^p = \Sigma_{k+1}^p,$$

Proposition Σ_k^pQ implies Proposition Σ_{k+1}^pQ. We give a relativized negative answer to this question in the case $k = 1$: there is an oracle under which Proposition Σ_2^pQ does not hold and Proposition Q holds.

Our proof uses a new "Kolmogorov generic" oracle: we show that for any Kolmogorov generic G, relative to $G \oplus$ (PSPACE-complete set), Proposition Q holds, the polynomial hierarchy is infinite and Proposition Σ_2^pQ does not hold. In addition we show that $\mathrm{P} \neq \mathrm{UP}$ relative to that oracle. The following theorem summarizes our results. It is a consequence of Theorems 2, 6, 3 and 5.

Theorem 1 *Let H denote any* PSPACE*-complete set and G any Kolmogorov-generic oracle. Let $G \oplus H$ stand for the join of G and H, that is, $G \oplus H = \{0x \mid x \in G\} \cup \{1x \mid x \in H\}$.*

1. *Relative to $G \oplus H$, Proposition* Q *is true.*
2. *Relative to G, as well as relative to $G \oplus H$, Proposition Σ_2^p*Q *is false.*
3. *Relative to G, as well as relative to $G \oplus H$, for all $k \geq 0$ we have $\Sigma_k^p \neq \Sigma_{k+1}^p$.*
4. *Relative to G, as well as relative to $G \oplus H$, $\mathrm{P} \neq \mathrm{UP}$.*

Note that other notions of genericity studied before, do not resolve the question. Indeed, if Proposition Q is true and PH is infinite under an oracle then $\mathrm{P} \neq \mathrm{NP}$ and $\mathrm{P} = \mathrm{NP} \cap \mathrm{coNP}$ under that oracle. The oracles having the latter property were constructed in [1, 4, 7, 12]. It is not hard to see that PH is infinite if and only if Proposition Q is false for all oracles constructed in those papers.

2 Definitions and Preliminaries

Let Σ denote the alphabet $\{0, 1\}$. The set of all finite-length binary strings is denoted Σ^*.

2.1 Complexity Classes

Our model of computation is the oracle Turing machine, both deterministic (DTM) and nondeterministic (NTM). Unless otherwise noted, all machines in this paper run in polynomial time. We assume that the reader is familiar with the complexity classes P, NP, UP, PSPACE, Σ_k^p, and Π_k^p for $k \geq 0$ as defined in e.g., [13, 15]. The class Δ_k^p is defined as $\mathrm{P}^{\Sigma_{k-1}^p}$, and $\mathrm{PH} = \bigcup_k \Sigma_k^p$ stands for the polynomial hierarchy. The class $\mathrm{F}\Delta_k^p$ is defined as the class of all functions from Σ^* to Σ^* that are computable in polynomial time with an oracle from Σ_{k-1}^p.

We say that disjoint sets B and C are P-separable if there is a set $D \in \mathrm{P}$ such that $B \subseteq D$ and $C \subseteq \Sigma^* - D$.

Proposition Q and its generalizations Σ_k^pQ are defined in the Introduction. The standard relationship between different definitions of NP [15, Theorem 7.17] gives that Σ_k^pQ is equivalent to the following statement:

Statement For every nondeterministic polynomial-time Turing machine M with oracle from Σ_{k-1}^p that accepts Σ^*, there is a function f in $\mathrm{F}\Delta_k^p$ such that, for all x, $f(x)$ is an accepting computation of $M(x)$.

It is easy to see the following:

Proposition 1 *If $\Sigma_{k-1}^p = \Sigma_k^p$ then $\Sigma_k^p Q$ is true. If $\Sigma_k^p Q$ is true then $\Delta_k^p = \Sigma_k^p \cap \Pi_k^p$.*

The first part of the proposition follows from our ability to binary search for a value of a $\mathrm{TF}\Sigma_k^p$-function using a Σ_k^p oracle which by the assumption falls into Σ_{k-1}^p, and the second part follows from the fact that for any language L in $\Sigma_k^p \cap \Pi_k^p$ we can design a $\mathrm{TF}\Sigma_k^p$-function that on input x outputs a string starting with one iff $x \in L$ and otherwise it outputs a string starting with zero. Proposition $\Sigma_k^p\mathrm{Q}$ implies that the value of the function can be found in $\mathrm{P}^{\Sigma_{k-1}^p}$.

2.2 Kolmogorov Complexity and Randomness

An excellent introduction to Kolmogorov complexity can be found in the textbook by Li and Vitányi [9]. We will state here the definitions and results relevant to our work. Roughly speaking, the Kolmogorov complexity of a binary string x is the minimal length of a program that generates x; the conditional complexity $C(x|y)$ of x conditional to y is the minimal length of a program that produces x with y as input. We provide a precise definition.

A *conditional description method* is a partial computable function Φ (that is, a Turing machine) mapping pairs of binary strings to binary strings. A string p is called a *description of x conditional to y* with respect to Φ if $\Phi(p, y) = x$. The complexity of x conditional to y with respect to Φ is defined as the minimal length of a description of x conditional to y with respect to Φ:

$$C_\Phi(x|y) = \min\{|p| \mid \Phi(p, y) = x\}.$$

A conditional description method Ψ is called *universal* if for all other conditional description methods Φ there is a constant k such that

$$C_\Psi(x|y) \le C_\Phi(x|y) + k$$

for all x, y. The Solomonoff–Kolmogorov theorem [8, 16] states that universal methods exist. We fix a universal Ψ and define *conditional Kolmogorov complexity* $C(x|y)$ as $C_\Psi(x|y)$. We call this Ψ the reference universal Turing machine. The (unconditional) Kolmogorov complexity $C(x)$ is defined as the Kolmogorov complexity of x conditional to the empty string. Comparing the universal function Ψ with the function $\Phi(p, y) = \Psi(p, \text{empty string})$ we see that the conditional Kolmogorov complexity does not exceed the unconditional one:

$$C(x|y) \;\le\; C(x) + O(1).$$

Comparing the universal function Ψ with the function $\Phi(p, y) = p$ we see that the Kolmogorov complexity does not exceed the length by no more than a constant:

$$C(x) \le |x| + k \tag{1}$$

for some k and all x. For most strings this inequality is close to an equality: the number of strings x of length n with

$$C(x) < n - m$$

is less than 2^{n-m}. Indeed, the total number of descriptions of length less than $n-m$ is equal to

$$1+2+\cdots+2^{n-m-1}=2^{n-m}-1.$$

In particular, for every n there is a string x of length n and complexity at least n. Such strings are called *incompressible, or random.*

Let $f(x,y)$ be a computable function mapping strings to strings. To describe the string $f(x,y)$ it is enough to concatenate x and y. Thus we obtain:

$$C(f(x,y)) \le 2|x|+|y|+k. \tag{2}$$

where k depends on f and on the reference universal machine but not on x,y. We have the extra factor of 2, as we need to separate x from y. To this end we write the former in a self-delimiting form. As a self-delimiting encoding of a string u we take the string $\bar{u}$ obtained from u by doubling all its bits and appending the pattern 01. For instance, $\overline{001}=00001101$. A similar inequality holds for computable functions of more than 2 strings:

$$C(f(x_1,x_2,\dots,x_n)) \le 2|x_1|+2|x_2|+\cdots+2|x_{n-1}|+|x_n|+O(1). \tag{3}$$

2.3 Kolmogorov-Generic Oracles

In order to create a relativized world where Proposition Q holds but the polynomial-time hierarchy is infinite we develop a new type of oracle, which we call a *Kolmogorov generic oracle.*

We create a set of allowable strings Y of indexed mutually independent Kolmogorov-random strings as follows:

For each n fix a binary string Z_n of length $n2^n$ that is incompressible, that is, $C(Z_n)\ge|Z_n|$. Divide Z_n into substrings $z_1,\dots,z_{2^n}$, each of length n. Let Y_n be the set $\{\langle i,z_i\rangle | i\in\{0,1\}^n\}$. (Here we identify i with the integer binary represented by i. A pair $\langle u,v\rangle$ is encoded be the string $\bar{u}v$, where $\bar{u}$ stands for the self-delimiting encoding of u defined in Sect. 2.2.) Let U be the set of all subsets of $Y=\bigcup Y_n$, where the union is over all tower n, i.e., n can be expressed as a tower of twos.

A *condition* $\alpha: Y\to\{0,1,*\}$ indicates which strings we have forced in or out of our generic oracle G where $\alpha(x)=0$ if we guarantee that x is not in G and $\alpha(x)=1$ if we guarantee that x is in G. In this paper, we only consider conditions that force a finite number of strings of G, i.e., the set $\alpha^{-1}(\{0,1\})$ is finite.

An *interval* U_α is a subset of U consistent with some condition α. If β is a condition satisfying for each $x\in Y$, $\alpha(x)\neq *$ implies $\beta(x)=\alpha(x)$, then U_β is a *sub-interval* of U_α. Suppose we have a property $P(A)$ on sets A. We say $P(A)$ is *dense* within U if for every interval U_α there is a sub-interval U_β such that for each $G\in U_\beta$, $P(G)$ is true.

For example, let $P_k(A)$ be the property that A has at least k strings. This is a dense property by choosing k strings in $\alpha^{-1}(*)$ and setting $\beta(x)=1$ for these strings.

Let α_0 map every x in Y to $*$. Define α_k so that $U_{\alpha_k}\subset U_{\alpha_{k-1}}$ and every G in U_{α_k} has property P_k, i.e., has at least k ones. Note that $\bigcap_k U_{\alpha_k}$ is non-empty and any G in that set must be infinite.

The same argument holds for any countable collection of dense properties. Intuitively, by Kolmogorov-generic oracle we mean an oracle satisfying some dense property. This is similar to the meaning of the term "random" oracle. Indeed, we say that a property P holds for a random oracle if P is a measure 1 set. In order to give the term Kolmogorov-generic oracle a fixed meaning within the paper we fix a logical system Γ strong enough to define every property described in this paper and we let $\mathcal{P}$ be the set of dense properties defined in Γ. By the argument above, there exists a set G such that $P(G)$ holds for every $P \in \mathcal{P}$. We call such G *Kolmogorov-generic*.

When proving that a certain property holds for a Kolmogorov-generic oracle G we use the fact that every two different lengths of strings in G are exponentially far apart. When discussing a particular polynomial-time computation, we only have to worry about strings at exactly one length in the oracle. Longer strings cannot be queried by the computation and so cannot affect it. Shorter strings can all be queried and found by the computation.

3 Results

Let H denote any PSPACE-complete set and G any Kolmogorov-generic oracle. Let $G \oplus H$ stand for the join of G and H, that is, $G \oplus H = \{0x \mid x \in G\} \cup \{1x \mid x \in H\}$.

Theorem 2 *Relative to $G \oplus H$, Proposition* Q *is true.*

We provide a sketch of the argument first. First, we relativize to a PSPACE-complete set H so that we are able to answer queries in P^H about PSPACE^H. Although later we relativize further to a Kolmogorov generic oracle $G \in U$, thus considering $(P^H)^G = P^{G\oplus H}$ computation, we will only need to be able to answer queries computable in PSPACE^H.

So we want to show that relative to a Kolmogorov generic oracle G and simultaneously relative to H, Proposition Q holds. We look on each nondeterministic polynomial time machine M independently. For a given interval of U if we can find a sub-interval of U that makes M not to accept all the inputs then we pick this sub-interval of U for our Kolmogorov generic oracle G; Proposition Q will be trivially satisfied for M then. If that is not the case so we cannot dispose of M so easily we will show how to find accepting paths of M efficiently for all the oracles in the given interval of U. We have the power of PSPACE at our disposal so we could search for accepting paths of M if M were not relativized to G or if it were relativized to a simple enough G so that we could pass the description of the oracle or its relevant part to our PSPACE search procedure. Since strings of different lengths in G are exponentially far apart the relevant part of G is quite restricted although, still possibly large in size. However, iteratively we can find a small (polynomial size) portion of G that is truly relevant for our search. We use the Kolmogorov properties of G for that. Once we know the relevant part of G we find the accepting path of M in PSPACE. The actual proof is next.

Proof of Theorem 2 We first assume that P = PSPACE and prove that Proposition Q is true under a Kolmogorov-generic oracle $G \in U$.

As explained in the section on generic oracles it suffices to show that for every polynomial-time oracle NTM M and relative to a Kolmogorov-generic oracle,

$$\begin{aligned}&\text{If } M \text{ accepts } \Sigma^* \text{ then there is a polynomial time machine}\\&\text{finding for each input an accepting computation of } M.\end{aligned} \tag{4}$$

Fix M. Without loss of generality, M on an input x runs in time $|x|^k + k$, for some constant k independent of its oracle. Indeed, for each oracle nondeterministic Turing machine M (not necessarily polynomial time) and natural k we can construct an NTM that acts as M supplied with a clock that prevents it from running more than in $|x|^k + k$ steps. If M^A runs in polynomial time then for some k the machine M^A supplied with the clock $|x|^k + k$ is equivalent to M^A.

We will show that the set of oracles satisfying (4) is dense. Let $I = U_\alpha$ be an interval in U. We need to construct a sub-interval J of I such that (4) is true for all $G \in J$. Consider two cases.

Case 1. There is a sub-interval of I such that for all A in that sub-interval, M^A does not accept Σ^*. Then let J be equal to that sub-interval of I.

Case 2. There is no such sub-interval. Consider the following polynomial-time deterministic algorithm A that, given an input x of length at least two, finds an accepting path of the computation $M^G(x)$. Let n be the largest tower number smaller or equal to $4|x|^{2k}$. The algorithm A will try to collect enough information about the oracle G so that it can find an accepting path of $M^G(x)$. The algorithm A starts by asking the value of G on all the strings in Y_i for $i \le \log n$. This can be done in time polynomial in $|x|$.

After that it iteratively builds a set Q of strings from $G \cap Y_n$ starting from an empty set Q. Using the assumption that P = PSPACE and the information about G collected so far, the algorithm finds the lexicographically first accepting path of M^G on x under the assumption that $G \cap \{\langle i, u\rangle | i, u \in \{0,1\}^n\} = Q$. (Note, M on x cannot query any string in Y_m, for $m > n$ so in PSPACE we can find such an accepting path given x, $Y_{\le \log n}$ and Q.) Such path does exist, as otherwise, the sub-interval J of I, consisting of all G' with $G' \cap \{\langle i, u\rangle | i, u \in \{0,1\}^n\} = Q$ and $G' \cap Y_i = G \cap Y_i$ for all $i \le \log n$ would qualify for case (1).

If this path is indeed an accepting path of the computation $M^G(x)$, A is done. If not then there is a string $w \in (G \cap \{\langle i, u\rangle | i, u \in \{0,1\}^n\}) \setminus Q$ that is queried along this path. Clearly such w is from Y_n. The algorithm picks the first such w, adds it to the set Q and iterates the process. Clearly, A eventually finds a correct accepting path of $M^G(x)$. We claim that A will find it within polynomially many iterations.

Observe, given M, x, $G \cap Y_{\le \log n}$ and the first $i-1$ strings of Q, the ith string added to Q can be described by $k \log |x|$ bits by its order number among the queries of M on x on the accepting path found under the assumption that $G \cap Y_n = \{$the first $i-1$ strings of $Q\}$. The set $G \cap Y_{\le \log n}$ has at most $n + \log n + \log\log n + \cdots$ strings, each of length at most $\log n$. Thus $G \cap Y_{\le \log n}$ can be described in at most $O(n \log n)$ bits. Hence if Q reaches size ℓ, we can describe Q by $\ell k \log |x| + O(n \log n) + 2|x| + O(1)$ bits (by (3) and the fact that we need to specify the length of $k \log |x|$-bit strings only once).

Recall that all of the strings in Y_n are derived from Z_n. Because of the way Y_n is defined any set A of ℓ strings from Y_n has Kolmogorov complexity at least $\ell n/2 - O(1)$.

Indeed, each element of Y_n is a pair $\langle i, y\rangle$. Let p denote the concatenation of all y's from all pairs $\langle i, y\rangle$ outside A arranged according to the order on i's. The length of p is $n(2^n - \ell)$. The initial string Z_n can be obtained from p by inserting the second components of pairs from A, their first components specifying the places where to insert. Thus given p and the shortest description q of A we can find Z_n, and (2) implies

$$n2^n \le C(Z_n) \le |p| + 2|q| + O(1) = n(2^n - \ell) + 2C(A) + O(1).$$

Since $2 + 2k \log|x| < n \le 4|x|^{2k}$, the Kolmogorov complexity of ℓ strings from Y_n is at least $\ell k \log|x| + \ell - O(1)$. Thus Q cannot grow bigger than $O(n \log n) + 2|x| = O(|x|^{2k} \log|x|)$.

We can remove the hypothesis that P = PSPACE by first relativizing to an oracle making P = PSPACE. It is known that relative to every PSPACE-complete set H we have P = PSPACE. Thus, relative to H, Proposition Q holds relative to a generic oracle in U. So we first relativize to H and then to G which is no different than relativizing to $G \oplus H$. □

The following theorem implies that relative to a Kolmogorov-generic oracle the polynomial hierarchy is infinite.

Theorem 3 *Relative to G, as well as relative to $G \oplus H$, for all $k \ge 0$ we have $\Sigma_k^p \ne \Sigma_{k+1}^p$.*

To establish the theorem we use the technique of Sipser [14] together with the result of Håstad [6] that there are functions computable by polynomial size depth-k circuits consisting of unbounded fan-in AND and OR gates that are not computable by depth-$(k-1)$ circuits of polynomial size. Sipser observes that the output of a $\Sigma_{k-1}^{p,G}$ computation on a fixed input can be computed by an appropriate size depth-$(k-1)$ circuit consisting of unbounded fan-in AND and OR gates that takes as its input the characteristic sequence of G (or its beginning segment). The proof follows.

Proof of Theorem 3 Meyer and Stockmeyer [11] show that if $\Sigma_k^p = \Sigma_{k+1}^p$ then $\Sigma_k^p = \Sigma_j^p$ for all $j \ge k$ and the proof of this relativizes. So it is sufficient for us to show that $\Sigma_{k-2}^p \ne \Sigma_{k+1}^p$ for all $k \ge 3$ relative to a Kolmogorov generic oracle G.

We use the Sipser [14] functions. The function f_k^m is represented by a depth k alternating AND/OR circuit tree with an OR gate at the top with fan-in m, and all fan-ins are m. Each variable occurs just once at each leaf.

Theorem 4 (Håstad [6]) *Depth $k-1$ circuits computing f_k^m are of size at least $2^{\Omega(m/\log m)}$.*

Pick a tower n. Set $m_n = 2^{\lfloor n/k \rfloor}$. The number of variables of $f_k^{m_n}$ is $m_n^k \leq 2^n$ for large n. For each of the variables of this formula assign a unique $i \in \{0,1\}^n$ so we can in polynomial-time find i from the variable and vice-versa.

Now consider the language $L_k(G) \subseteq \{1\}^*$:

> 1^n *is in* $L_k(G)$ *iff* $f_k^{m_n}$ *is true if we set the variables corresponding to* i *to one when* $\langle i, z_i \rangle$ *is in* G *and to zero otherwise.*

We will show relative to a Kolmogorov generic oracle G, $L_k(G) \in \Sigma_{k+1}^{p,G} - \Sigma_{k-2}^{p,G}$.

First notice that $L_k(G) \in \Sigma_{k+1}^{p,G}$ for all $G \in U$: Consider an alternating Turing machine that uses k alternations to simulate the circuit. To determine whether a variable corresponding to i is true the machine makes the NP query "is there a z such that $\langle i, z \rangle$ is in G." This gives us a $\Sigma_k^{\mathrm{NP},G} = \Sigma_{k+1}^{p,G}$ machine accepting $L_k(G)$.

Let M be an alternating Σ_{k-2}^p oracle Turing machine that runs in time n^j. Let $I = U_\alpha$ be an interval in U. We need to construct a sub-interval J of I such that M^G does not accept $L(G)$ for all $G \in J$. Along the lines of Sipser [14] we can convert the computation to a circuit of depth $k-1$ and size $2^{O(n^j)}$ whose input variables correspond to queries to G. This way we obtain a circuit whose variables are the same as those in $f_k^{m_n}$ in the definition of $L_k(G)$ on 1^n. By Theorem 4 for sufficiently large n this circuit cannot compute $f_k^{m_n}$ so there must be some setting of the variables where the circuit and $f_k^{m_n}$ have different outputs. Add to the condition α the requirement $\langle i, z_i \rangle \in G$ if variable i is assigned 1 in this setting and the requirement $\langle i, z_i \rangle \notin G$ otherwise. For all $G \in U$ satisfying the resulting condition, $M^G(1^n)$ accepts iff 1^n is not in $L(G)$.

The case of Kolmogorov-generic oracle is done. For the oracle $G \oplus H$, the proof is entirely similar: again, one uses precisely the same language $L_k(G)$, and in the circuit obtained from the computation of machine M one hardwires the queries not of the form $\langle i, z_i \rangle$ to one if they belong to H and to zero otherwise. □

Grollman and Selman [5] established a connection between the existence of *worst-case* one-way functions and P $\neq$ UP, see also [13]. Using this connection we can show that one-way functions exist relative to G.

Theorem 5 *Relative to a Kolmogorov-generic oracle G, as well as relative to $G \oplus H$,* P $\neq$ UP.

Proof Define the relativized language L^X as $\{\langle i, 0^n \rangle : (\exists z)|z| = n \;\&\; \langle i, z \rangle \in X\}$. For a string z of length n, there is at most one string of the form $\langle i, z \rangle$ in G so the language is in UP^G. A simple argument demonstrates that L^G is not in P^G. □

Can the proof that Proposition Q holds relative to a Kolmogorov-generic be lifted to show that Σ_k^pQ holds and thus we get the collapse of Δ_k^p and $\Sigma_k^p \cap \Pi_k^p$? The answer is no for $k = 2$ and the proof of this shows that this is true for a broad class of finite extension oracles.

To show that Σ_2^pQ fails relative to a Kolmogorov-generic oracle G, let f^G be a function from Σ^* to Σ^* where for every x of length n

$$f^G(x) = y_1 \dots y_{n-1}$$

and

$$y_j = 1 \quad \Longleftrightarrow \quad (\exists u, z), \quad |u| = n, \quad |z| = 2n + \lceil \log n \rceil, \quad \langle xju, z \rangle \in G.$$

No matter what strings are in G, the pigeonhole principle tells us that, for all n, there will always be a *collision*, that is, two different strings x_1 and x_2 of length n such that $f^G(x_1) = f^G(x_2)$.

Let M be a $\Sigma_2^{p,G}$ machine that on any input of length n guesses two different strings of length n in its existential step and then accepts iff those strings collide on f^G. It is clear from the definition of f^G that M can find these collisions and that it accepts Σ^*. A $\mathrm{P}^{\mathrm{NP}^G}$ machine that finds an accepting path of M could be modified to output the two colliding strings found by M on that path so, without loss of generality, we will assume it does just that. (Because of the inner working of M which we designed by ourselves, the two colliding strings are directly determined by certain well defined bits in the nondeterministic choices of M.)

Theorem 6 *Relative to a Kolmogorov generic oracle G, as well as relative to $G \oplus H$, no* P^{NP} *machine can find an accepting path of the computation $M(x)$ for every x.*

Again in the proof we will look on each P^{NP} machine independently and show that relative to a Kolmogorov generic oracle G the machine fails to find an accepting path of M, i.e., it fails to output a collision in f^G. We will do it by imposing certain conditions on G that will fix the computation of the P^{NP} machine while still allowing us to define f^G on most of its domain in a suitable way. The conditions will be imposed iteratively for each additional query of the P^{NP} machine to its NP oracle. Indeed our goal will be to fix the computation of the NP oracle machine on the queries it receives. We will do it by imposing additional constraints on G in a way that does not produce *observable* collisions in f^G. Once the P^{NP} machine outputs the pair of strings forming a presumed collision we constrain G as not give a collision of f^G on that pair of strings.

Proof of Theorem 6 Let $\langle R, N \rangle$ be an arbitrary pair consisting of an oracle polynomial time DTM R and an oracle polynomial time NTM N. We will show that the set of all oracles G such that R with oracle N^G does not find any collision of f^G is dense in U.

Without loss of generality we can assume that there are polynomial upper bounds of the running time of R and N that do not depend on their oracles. Let p_R and p_N stand for those polynomials, respectively.

Let I_α be an interval in U. We will show that for some n there is an interval $I_\beta \subset I_\alpha$ such that for all $G \in I_\beta$, $R^{N^G}(0^n)$ does not find two strings that collide on f^G.

We pick a large enough n so that the rest of the argument would go through. In particular, n should be such that $2n + \lceil \log n \rceil$ is a tower number and it should be bigger than the maximal length of strings in the domain of α. (We call the set of all y such that α contains a condition $y \in G$ or $y \notin G$ the domain of α and use the notation $\operatorname{dom} \alpha$ for the domain of α.)

Note that the outcome of R^{N^G} on input 0^n depends only on membership in G of strings of length at most $p_N(p_R(n))$. First we add to α the requirements $y \notin G$

for all strings $y \notin Y_{2n+\lceil\log n\rceil} \cup \operatorname{dom}\alpha$ of length at most $p_N(p_R(n))$ and denote by β_0 the resulting condition. The condition β is obtained from β_0 in at most $p_R(n)$ iterations. In ith iteration we define a condition β_i obtained from β_{i-1} by adding some requirements of the form $y \in G$ and $y \notin G$ for $y \in Y_{2n+\lceil\log n\rceil}$.

Let us explain this in more detail. For $x \in \Sigma^n$ and $j = 1, \dots, n-1$ let

$$B_{xj} = \{\langle xju, z_{xju}\rangle \mid u \in \Sigma^n\}.$$

We call the set $B_x = \bigcup_{j=1}^{n-1} B_{xj}$ the *bag* corresponding to x. The value $f^G(x)$ depends only on $B_x \cap G$. More specifically, jth bit of $f^G(x)$ is 0 if the set $B_{xj} \cap G$ is empty.

In each iteration we choose a set $D \subset \Sigma^n$ of cardinality at most $p_N(p_R(n))$ and *set* oracle's value on the set $\bigcup_{x\in D} B_x$. This means that for every y in this set we include in β_i either the condition $y \notin G$, or the condition $y \in G$. The notation D_i will refer to the set of all strings x such that oracle's value is set on B_x during iterations $s = 1, \dots, i$. We will keep the following statement invariant:

$$f^G \textit{ is injective on } D_i \textit{ for all } G \in I_{\beta_i}.$$

Additionally, in the ith iteration we choose the *desired* answer a_i of N^G to the ith query to N^G in the run of R on input 0^n.

In ith iteration we run R on input 0^n assuming the answers $a_1, \dots, a_{i-1}$ to oracle queries until R makes ith query q_i to the oracle or outputs a result. If the first option happens, we choose the desired answer of N^G on q_i as follows.

Assume that $G \in I_{\beta_{i-1}}$ and C is an accepting computation of N^G on input q_i. We say that $\langle G, C\rangle$ is a *good pair* if the following holds. Let D be the set of all $x \in \Sigma^n$ such that computation C queries a string in the bag of x. The pair $\langle G, C\rangle$ is good if f^G is injective on the set $D \cup D_{i-1}$.

Assume first that there is a good pair $\langle G, C\rangle$. In this case we pick a good pair $\langle \tilde{G}, \tilde{C}\rangle$, define D as explained above and choose YES as the desired answer to ith query. The condition β_i is obtained from β_{i-1} by adding the requirements $y \in G$ for all $y \in \bigcup_{x\in D} B_x \cap \tilde{G}$ and the requirements $y \notin G$ for all $y \in \bigcup_{x\in D} B_x \setminus \tilde{G}$. Note that $N^G(q_i) = 1$ for all $G \in I_{\beta_i}$.

If there is no good pair $\langle G, C\rangle$ then we choose NO as the desired answer to ith query and set $\beta_i = \beta_{i-1}$, $D_i = D_{i-1}$.

On some iteration $k \le p_R(n)$, R makes no new queries and outputs two strings x_1 and x_2, where f^G presumably collides. At this point we set oracle's value on all remaining strings in $Y_{2n+\lceil\log n\rceil}$ as follows. Pick any oracle $\tilde{G} \in I_{\beta_{k-1}}$ such that $f^{\tilde{G}}$ is injective on the set $D_k = D_{k-1} \cup \{x_1, x_2\}$ and such that for all $x \in \Sigma^n \setminus D_k$, $f^{\tilde{G}}(x) = 00\dots0$. As n is large enough there is such $\tilde{G}$. Indeed, the length of q_i is at most $p_R(n)$ and thus every computation of $N^{\tilde{G}}$ on input q_i runs in time $p_N(p_R(n))$. Hence $|D_k|$ is bounded by the polynomial $p_R(n)p_N(p_R(n)) + 2$. If 2^{n-1} is bigger than this bound then there are enough strings in the range of f to avoid collision in D_k.

We let β be the condition containing the requirements $y \in G$ for all $y \in \tilde{G}$ of length at most $p_N(p_R(n))$ and the requirements $y \notin G$ for all $y \notin \tilde{G}$ of length at most $p_N(p_R(n))$.

We claim that for all $G \in I_\beta$, R^{N^G} on 0^n computes the way how we determined. Indeed, if R computes differently for some $G \in I_\beta$ then there must be a query answered in the opposite way than we desire. Let q_i be the first such query. Note that q_i coincides with the ith query in our construction, as all the previous queries are answered by N^G as we desire. If we have chosen YES as the desired answer to ith query then by construction $N^G(q_i) = 1$ and thus the desired answer is correct. Therefore this may happen only if we have chosen NO as the ith answer and $N^G(q_i) = 1$.

By way of contradiction, assume that this is the case. Pick then an accepting computation C of N^G on q_i. We will show that there is $G' \in I_{\beta_{i-1}}$ such that $\langle G', C\rangle$ is a good pair. Let D be the set of all $x \in \Sigma^n$ such that computation C queries a string in the bag of x. Note that by construction f^G is injective on D_k. (However, f^G may be not injective on $D_{i-1} \cup D$ thus $\langle G, C\rangle$ may be not a good pair.)

We will add to G some strings from $\bigcup_{x \in D \setminus D_k} B_x$ so that for the resulting oracle G' the pair $\langle G', C\rangle$ is good. We may assume that 2^n, the cardinality of every set B_{xj}, is greater than the number of queries along C. For every $x \in D \setminus D_k$ and every j we can change jth bit of $f^G(x)$ to 1 by adding to G a non-queried string from B_{xj}. All of the 2^{n-1} values in the range of f can be obtained in this way as we did not set any string in B_x for any $x \in D \setminus D_k$. Thus we can change $f^G(x)$ for all $x \in D \setminus D_k$ one by one so that for the resulting oracle G', C is an accepting computation and $f^{G'}(x)$ is injective on $D \cup D_k$ and hence on $D \cup D_{i-1}$.

The case of Kolmogorov-generic oracle is done. For the oracle $G \oplus H$, the proof is entirely similar: again one uses the same function f^G. As the oracle H is fixed the same reasoning as above gives conditions on G that ascertain that no P^{NP} machine can find collisions in f^G even if it has access to H in addition to G. □

4 Conclusion and Open Problems

Is there an oracle relative to which the polynomial-time hierarchy is proper and Σ_k^pQ is true for all k? As a corollary we would get a relativized world where the hierarchy is proper and $\Delta_k^p = \Sigma_k^p \cap \Pi_k^p$. The second statement remains open even relative to Kolmogorov generics and, if true, would give a relativized version of the polynomial-time hierarchy that acts like the arithmetic hierarchy.

Acknowledgements We thank Steve Fenner and Marcus Schaefer for helpful discussions. We thank the anonymous referees for useful comments. The work of N. Vereshchagin was in part done at the CWI, Amsterdam and in part supported by RFBR grant 06-01-0012. M. Koucký was in part done at the CWI, Amsterdam and supported in part by grant GA ČR 201/07/P276, project No. 1M0021620808 of MŠMT ČR and Institutional Research Plan No. AV0Z10190503. H. Buhrman and M. Koucký were supported by the Netherlands Organisation for Scientific Research (NWO) in the framework of the Innovational Research Incentives Scheme (in Dutch: Vernieuwingsimpuls), Vici-project 639.023.302.

References

1. Baker, T., Gill, J., Solovay, R.: Relativization of P = NP question. SIAM J. Comput. **4**(4), 431–442 (1975)
2. Bellare, M., Goldwasser, S.: The complexity of decision versus search. SIAM J. Comput. **23**(1), 97–119 (1994)
3. Fenner, S., Fortnow, L., Naik, A., Rogers, J.: Inverting onto functions. Inf. Comput. **186**, 90–103 (2003)
4. Fortnow, L., Rogers, J.: Separability and one-way functions. Comput. Complex. **11**, 137–157 (2003)
5. Grollman, J., Selman, A.: Complexity measures for public-key cryptosystems. SIAM J. Comput. **17**(2), 309–335 (1988)
6. Håstad, J.: Almost optimal lower bounds for small depth circuits. Adv. Comput. Res. **5**, 143–170 (1989)
7. Impagliazzo, R., Naor, M.: Decision trees and downward closures. In: Proceedings of the 3rd IEEE Structure in Complexity Theory Conference, pp. 29–38. IEEE, New York (1988)
8. Kolmogorov, A.N.: Three approaches to the quantitative definition of information. Probl. Inf. Trans. **1**(1), 1–7 (1965)
9. Li, M., Vitányi, P.: An Introduction to Kolmogorov Complexity and Its Applications, 2nd edn. Graduate Texts in Computer Science. Springer, New York (1997)
10. Megiddo, N., Papadimitriou, C.: On total functions, existence theorems and computational complexity. Theor. Comput. Sci. **81**(2), 317–324 (1991)
11. Meyer, A., Stockmeyer, L.: The equivalence problem for regular expressions with squaring requires exponential space. In: Proceedings of the 13th IEEE Symposium on Switching and Automata Theory, pp. 125–129. IEEE, New York (1972)
12. Muchnik, A., Vereshchagin, N.: A general method to construct oracles realizing given relationships between complexity classes. Theor. Comput. Sci. **157**, 227–258 (1996)
13. Papadimitriou, C.: Computational Complexity. Addison-Wesley, New York (1994)
14. Sipser, M.: Borel sets and circuit complexity. In: Proceedings of the 15th ACM Symposium on the Theory of Computing, pp. 61–69. ACM, New York (1983)
15. Sipser, M.: Introduction to the Theory of Computation. PWS, Boston (1997)
16. Solomonoff, R.J.: A formal theory of inductive inference, parts 1 and 2. Inf. Control **7**, 1–22, 224–254 (1964)