Electronic Colloquium on Computational Complexity, Report No. 24 (2006)

Inverting Onto Functions Might Not Be Hard

Harry Buhrman*
CWI, Amsterdam

Lance Fortnow†
University of Chicago

Michal Koucký‡
Mathematical Institute of Czech Academy of Sciences

John D. Rogers§
DePaul University, Chicago

Nikolai Vereshchagin¶
Moscow State University

February 17, 2006

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?

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.

To create the oracle, we introduce Kolmogorov-generic oracles where the strings placed in the oracle are derived from an exponentially long Kolmogorov-random string. We also show that relative to this same oracle, $\mathbf{P} \neq \mathbf{UP}$ and $\mathbf{TFNP}^{\mathbf{SAT}}$ functions are not computable in polynomial-time with a **SAT** oracle.

1 Introduction

Megiddo and Papadimtriou [MP91] defined the class **TFNP**, the class of multivalued functions with values that are polynomially verifiable and guaranteed to exist. This class of functions includes Factoring, Nash Equilibrium, finding solutions of Sperner's Lemma, and finding collisions of hash functions.

Fenner, Fortnow, Naik and Rogers [FFNR03] 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. Fenner et. al. showed that 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 $M(x)$.

*Email: buhrman@cwi.nl
†Email: fortnow@cs.uchicago.edu
‡Email: mkoucky@cs.mcgill.ca
§Email: jrogers@cs.depaul.edu
¶Email: ver@mech.math.msu.su

ISSN 1433-8092

- Given an honest onto polynomial-time computable function g there is a polynomial-time computable function f such that $g(f(x)) = x$.
- 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 $L(M) =$ **SAT**, there is a polynomial-time computable function f such that for every ϕ in **SAT** and accepting path p of $M(\phi)$, $f(\phi, p)$ is a satisfying assignment of ϕ.

Fenner et. al. ask whether we can draw any complexity collapses from Q, in particular if 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.

Our proof uses a new "Kolmogorov Generic" which uses finite extensions of subsets of independently random strings.

We also show that for any Kolmogorov Generic G,

1. $\mathbf{P}^G \neq \mathbf{UP}^G$.
2. $\mathrm{Q}^{\mathbf{NP}^G}$ does not hold, i.e., that is relative to Kolmogorov generics there is an onto function computable with a **SAT** oracle which we cannot invert even with the help of a **SAT** oracle.

2 Definitions and preliminaries

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**, **PSPACE**, Σ_k^p, and Π_k^p for $k \geq 0$. The class Δ_k^p is defined as $\mathbf{P}^{\Sigma_{k-1}^p}$.

A k-alternating machine TM is one whose computation alternates $k-1$ times between existential steps and universal steps.

We call *Proposition Q* the statement "For every nondeterministic polynomial-time machine M that accepts Σ^*, there is a polynomial-time function f such that $f(x)$ is an accepting path of the computation $M(x)$."

We will label Σ_k^pQ the statement "For every nondeterministic polynomial-time Turing machine M with oracle from Σ_{k-1} that accepts Σ^*, there is a function f computable by a Δ_k^p machine such that, for all x, $f(x)$ is an accepting computation of $M(x)$."

It is easy to see the following:

Proposition 1 *If $\Sigma_k^p Q$ is true then $\Delta_k^p = \Sigma_k^p \cap \Pi_k^p$.*

2.2 Kolmogorov complexity and randomness

An excellent introduction to Kolmogorov complexity can be found in the textbook by Li and Vitányi [LV97]. We will state here the definitions and results relevant to our work. Roughly speaking, the Kolmogorov complexity of a binary string x is defined as 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.

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 C such that

$$C_\Psi(x|y) \leq C_\Phi(x|y) + C$$

for all x, y. The Solomonoff–Kolmogorov theorem 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:

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

Comparing the universal function Ψ with the function $\Phi(p, y) = p$ we see that the Kolmogorov complexity does not exceed the length:

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

for some C and all x. For most stings 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:

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

where C depends on f and on the reference universal machine but not on x, y. The factor of 2 is needed, as we have 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:

$$C(f(x_1, x_2, \ldots, x_n)) \ \leq \ |x_1| + 2|x_2| + \cdots + 2|x_n| + O(1). \tag{3}$$

2.3 Generic oracles

The oracles we use are *generic* oracles. What does this mean? To explain this we need to recall some definitions from category theory.

A *condition* on an oracle A is a finite set of *requirements* having the form $x \in A$ and $y \notin A$, where $x, y \in \Sigma^*$. We say that an oracle A *satisfies* a condition α if all the requirements in α are satisfied by A. Let U be a family of oracles and let U_α denote the set of all $A \in U$ satisfying α. An *interval* in U is a non-empty subset of U having the form U_α. A subset S of U is called *dense in* U if every non-empty interval I in U has a sub-interval J in U included in S. Countable intersections of dense sets are called *large* subsets of U.

Let $P(A)$ be a property of an oracle A. We say that P *holds for a generic oracle in* U if the set $\{A \in U \mid P(A)\}$ is large in U. Assume that U has the following property: the intersection of every infinite descending chain

$$I_1 \supset I_2 \supset I_3 \supset \ldots$$

of intervals in U is non-empty. Then by the usual diagonalization we can show that every large subset of U is non-empty. Using a metaphor, we can say that "generic oracles exist." Our usage of the term "generic" oracle is similar to the usage of the term "random" oracle. Indeed, we say that a property P holds for a random oracle if the set of oracles satisfying P is a measure 1 set.

Note that if a property P holds for a generic oracle in U and Q holds for a generic oracle in U then so does $P \wedge Q$. Therefore if we want to prove that Proposition Q holds but that the PH is infinite relative to a generic oracle in U we can prove these things separately. The same applies to countable families of properties. If each P_i holds for a generic oracle in U then the property $\forall i \ P_i$ also holds for a generic oracle in U. For example, if we want to prove that $\mathbf{P} \neq \mathbf{NP}$ relative to a generic oracle in U we can define a relativized language L that is in $\mathbf{NP}$ for generic oracle in U and then define a set of requirements R_i, where R_i is the statement "DTM M_i does not accept L." Then it is enough to prove, for every i, that R_i holds relative to a generic oracle in U. To this end it suffices to prove that the set of $A \in U$ such that R_i holds is dense in U: every interval I in U has a subinterval J in U such that R_i holds for all $A \in J$.

Good introductions and several applications of the approach we are using here may be found in the papers by Fortnow, et al [FFKL03] and Fortnow and Rogers [FR03].

2.4 Kolmogorov-generic oracles

Let us now define the specific set U of oracles. A generic oracle in U will be called a *Kolmogorov generic oracle*. For each n fix a binary string Z_n of length $n2^n$ that is incompressible, that is, $C(Z_n) \geq |Z_n|$. Divide Z_n into substrings $z_1, \ldots, z_{2^n}$, each of length n. Let Y_n be the set $\{\langle i, z_i \rangle | 1 \leq i \leq 2^n\}$. The set U is the set of all subsets of $\bigcup_n Y_n$ where the union is over all "tower" $n = 1, 2, 2^2, 2^{2^2}, \ldots$. The next tower number from n is 2^n. 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.

2.5 Categorical properties

We say that a property P of an oracle Turing machine M is *categorical in an interval* I (with respect to Kolmogorov-generic oracles) if every sub-interval of I has an oracle A such that M^A has this property. For example, if every sub-interval of I has an oracle A such that a machine accepts Σ^*, we say that it accepts Σ^* categorically in I.

3 Results

Throughout the following discussion, we will assume that $\mathbf{P} = \mathbf{PSPACE}$. We will see later that we can remove this assumption.

Theorem 1 *Relative to a Kolmogorov-generic oracle, Proposition Q is true.*

Proof. As explained above it suffices for every polynomial-time oracle NTM M, to prove that relative to a Kolmogorov-generic oracle,

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

Fix M. 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 $G \in J$. If M does not accept Σ^* categorically in I then there is a sub-interval of I such that M does not accept Σ^* relative to all oracles in this sub-interval. Then we can let J be equal to this sub-interval of I. Thus we can assume that M accepts Σ^* categorically in I.

Consider the following polynomial-time deterministic algorithm A that, given an input x, finds an accepting path of the computation $M^G(x)$. Let n be the closest tower number to $|x|$. Then $|x| \leq (2^n + n)/2$ and hence $M^G(x)$ cannot query strings in Y_{2^n} provided $|x|$ large enough (for short strings x we make an exhaustive search).

The algorithm A first asks the value of G on all the strings in Y_i for $i \leq \log n$. As $|x| \geq n/2$ this can be done in time polynomial in $|x|$. Recall that all of strings in Y_n are derived from Z_n. Using the assumption that $\mathbf{P} = \mathbf{PSPACE}$, the algorithm tries to find an accepting path of $M(x)$ along which all of the oracle queries to strings in Y_n expect not to find that string in the oracle. If no such path exists, this means that every path expects to find at least one string in the oracle. In that case consider the condition β that is obtained from α by adding requirements $y \notin G$ for all $y \in Y_n$ (we can assume that the length of strings in Y_n greater than the length of all strings in α). Then M^G does not accept x for every $G \in U$ satisfying β. But this contradicts the fact that M categorically in I accepts Σ^*.

Let p be the lexicographically least such accepting path. The algorithm now queries every string queried along p. If none of the strings are in the oracle then p is a valid accepting path relative to the oracle and so A outputs it. Otherwise, A places each string it finds into a set Q and repeats

its search, this time looking for paths in $M(x)$ that only expect to find the strings in Q. We then repeat the process until A outputs a path.

We can describe each of the strings we add to Q by the index of the query A makes to the oracle, $k \log |x|$ bits if A runs in time $|x|^k$. We can describe Z_n by the strings in Y_n not queried, the indices used to describe each string in Q, the input x and the values of G on strings of length at most $\log n$. So by (3) we have

$$\begin{aligned} n2^n \leq C(Z_n) &\leq (2^n - |Q|)n + 2k|Q| \log |x| + 2|x| + 2n + O(1) \\ &\leq (2^n - |Q|)n + 2k|Q| \log |x| + 7|x|. \end{aligned}$$

We have $|Q| \leq 7|x|/(n - 2k \log |x|)$. We have two cases

1. $2k \log |x| \geq n$: Then $|x| \geq 2^{n/2k}$ and so even if we queried every string $\langle i, u \rangle$, $i \leq 2^n$, $|u| = n$, we are still running in time polynomial in $|x|$.

2. $2k \log |x| < n$: Then $|Q| \leq 7|x|$ and since we add a string to $|Q|$ in every step, we are running in time polynomial in $|x|$. ∎

Noting that the above proof is still valid under the hypothesis that $\mathbf{P} = \mathbf{PSPACE}$, we can remove the hypothesis that $\mathbf{P} = \mathbf{PSPACE}$ by first relativizing to an oracle making $\mathbf{P} = \mathbf{PSPACE}$. It is known that relative to every **PSPACE**-complete set H we have $\mathbf{P} = \mathbf{PSPACE}$. Thus, relative to a Kolmogorov generic oracle G, Q-property holds relative to H. In other words, for a Kolmogorov generic G, Q-property holds relative to the oracle

$$G \oplus H = \{0x \mid x \in G\} \cup \{1x \mid x \in H\}.$$

Theorem 2 *Relative to a Kolmogorov generic oracle G, $\Sigma_k^p \neq \Sigma_{k+1}^p$.*

Proof.

Stockmeyer and Meyer [MS72] show that if $\Sigma_k^p = \Sigma_{k+1}^p$ then $\Sigma_k^p = \Sigma_j^p$ for all $j \geq k$. So it is sufficient for us to show that $\Sigma_{k-2}^p \neq \Sigma_{k+1}^p$ for all $k \geq 3$ relative to G.

We use the Sipser [Sip83] functions as defined by Hastad [Has89]. The function f_k^m is represented by a depth k alternating circuit tree with a top OR gate on top with fan-in $\sqrt{m/\log m}$, bottom fan-in $\sqrt{km \log m/2}$ and all other fanouts are m. Each variable occurs just once at each leaf.

Theorem 3 (Håstad) *Depth $k-1$ circuits computing f_k^m are of size at least $2^{\Omega(\sqrt{m/(k \log m)})}$.*

Pick a tower n. Set $m = 2^{n/k}$. The number of variables of f_k^m is $m^{k-1}\sqrt{k/2} < 2^n$ 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 $L_k(G)$ such that input 1^n is in $L(G)$ if f_k^m is true if we set the variables corresponding to i to one if $\langle i, z_i \rangle$ is in G and 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 a 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 $\Sigma_k^{\mathbf{NP},G} = \Sigma_{k+1}^{p,G}$ machine accepting $L_k(G)$.

Let M be a 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 subinterval J of I such that M does not accept $L(G)$ for all $G \in J$. Along the lines of Sipser [Sip83] we can convert the computation to a circuit whose variables correspond to queries to G of depth $k-1$ and size $2^{O(n^j)}$. Hardwire the queries not of the form (i, z_i) to zero and we have a circuit whose variables are the same as those in f_k^m in the definition of $L_k(G)$ on 1^n. By Theorem 3 for sufficiently large n this circuit cannot compute f_k^m so there must be some setting of the variables where the circuit and f_k^m 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 $L(G)$. ∎

Just by the same proof we can show, that for a Kolmogorov generic G, $\Sigma_k^p \neq \Sigma_{k+1}^p$ relative to $G \oplus H$ where H is any **PSPACE**-complete set.

We can also show that one-way functions exist relative to G.

Theorem 4 *Relative to a Kolmogorov generic oracle G, $\mathbf{P} \neq \mathbf{UP}$.*

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 $\mathbf{UP}^G$. A simple diagonalization argument demonstrates that L^G is not in $\mathbf{P}^G$. ∎

Proposition Q at other levels

Generalizing Q to other levels in the polynomial time hierarchy gives us the proposition Σ_k^pQ: "For every k-alternating polynomial-time Turing machine M that accepts Σ^*, there is a function f computable by a Δ_k^p machine such that, for all x, $f(x)$ is an accepting computation of $M(x)$." The implication that $\mathbf{P} = \mathbf{NP} \cap \mathbf{coNP}$ holds when Q holds becomes: If Σ_k^pQ is true then $\Delta_k^p = \Sigma_k^p \cap \Pi_k^p$.

Can the proof that Q holds relative to a Kolmogorov-generic be lifted to show that Σ_k^pQ holds and we get the collapse of the 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, fix a length n. Let f be a function from Σ^n to Σ^{n-1} where

$$f(x) = y_1 \ldots y_{n-1}$$

and

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

No matter what strings are in G, the pigeonhole principle tells us there will always be a *collision*, that is, two different strings x_a and x_b of length m such that $f(x_a) = f(x_b)$. Also note that placing strings in the oracle to set the value of f on one string will not change the value of f on any other string. For $x \in \Sigma^m$ we call the set $\{\langle xju, z\rangle \in Y_{2n+\log n}\}$ the *bag* corresponding to x. Note, to set a particular bit of $f(x)$ to 1 we can choose one of 2^n strings from the bag of x to be put in G.

Let M be a Σ_2^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. It is clear from the definition of f that M can find these collisions and that it accepts Σ^*. A $\mathbf{P}^{\mathbf{NP}^G}$ machine that finds an accepting path of M could be modified to output the two colliding strings on that path so, without loss of generality, we will assume it does just that.

Theorem 5 *Relative to a Kolmogorov generic oracle G, no $\mathbf{P}^{\mathbf{NP}}$ machine can find an accepting path of the computation $M(x)$ for every x. The same is true relative to $G \oplus H$, where H is any* **PSPACE***-complete set.*

Proof. Let P be a DTM oracle machine running in time p_P and making queries to L^G, an $\mathbf{NP}^G$ language. Let N be an oracle NTM machine accepting L that runs in time p_N. Let I_α be an interval in U. It suffices to show that for all large enough n, $P^{L^G}(0^n)$ does not find an accepting computation of $M^G(0^n)$, that is, it does not find two strings that collide on f, for some $G \in I$. Indeed, we then can include in α the answers to queries to G along all the computations of N on all queries to N and the answers to queries to G along the computation of P on 0^n.

Fix large n. We will decide about membership in G of strings from Y_n in at most $p_P(n)+1$ iterations. We run P on 0^n and one by one we decide whether N^G answers the i-th query q_i of P by YES or NO. During these iterations we will maintain a set $D \subseteq \Sigma^n$ for which we have already decided the values of f as well as about membership in G of all the strings in bags corresponding to D.

Iteratively for each q_i we do the following. Consider the computation of N^G on query q_i. Its outcome is given by an OR of exponentially many ANDs, where each AND is of size at most $p_N(p_P(n))$, and it is an AND of membership and non-membership queries to G. Answers to these queries to G are already fixed except for strings in bags corresponding to $\Sigma^n \setminus D$. (All queries to strings w outside Y_n are answered negatively unless the requirement $w \in G$ belongs to the condition α.) If there is a way to make one of the ANDs to evaluate to true by setting membership in G for strings in at most $p_N(p_P(n))$ bags for some $D_i \subseteq \Sigma^n \setminus D$ without creating any collision of f on $D \cup D_i$, we set the membership of all these strings in bags corresponding to D_i in that way and we extend D by D_i. In this case we force $N^G(q_i)$ to evaluate to YES. If there is no way to make any of the ANDs to evaluate to true given our conditions, we declare that $N^G(q_i)$ evaluates to NO. We proceed to the next query of P.

After at most $p_P(n)$ queries, P outputs two strings x_1 and x_2, where f presumably collides. At this point we set all strings in bags for $x \in \Sigma^n \setminus D$ so that $f(x)$ contains exactly $n/2$ zeros and so that $f(x_1) \neq f(x_2)$. We claim that after this P^{N^G} on 0^n computes the way how we determined.

Indeed, if P computes differently then there must be a query that P asks to N^G which evaluates to YES although we declared that it will evaluate to NO. Let q_i be the first such a query. Hence, in the representation of the computation $N^G(q_i)$ one of the ANDs evaluates to true now. Call this AND $\mathcal{G}$. Consider all $x \in \Sigma^n \setminus D$ such that $\mathcal{G}$ queries a string in the bag of x. $f(x)$ contains $n/2$ zeros for every such x. By adding certain $\langle xju, z\rangle$ from the bag of x to the oracle G we can change any zero in $f(x)$ into one without changing the value of $\mathcal{G}$. There are exponentially many values $f(x)$ that can be obtained in this way. This is true for all bags that are touched by $\mathcal{G}$ and that do not belong to D. Hence, we could have fixed $\mathcal{G}$ to true without introducing collisions in f already in the i-th iteration. Hence, all queries that we declared to be NO are indeed NO. ∎

4 Conclusion and open problems

Is there an oracle relative to which the polynomial-time hierarchy is proper and $\Sigma_k^p \mathrm{Q}$ 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.

Acknowledgments

We thank Steve Fenner and Marcus Schaefer for helpful discussions.

References

[FFKL03] Lance Fortnow, Stephen Fenner, Stuart Kurtz, and Lide Li. An oracle builder's toolkit. *Information and Computation*, 182:95–136, 2003.

[FFNR03] Lance Fortnow, Stephen Fenner, Ashish Naik, and John Rogers. Inverting onto functions. *Information and Computation*, 186:90–103, 2003.

[FR03] Lance Fortnow and John Rogers. Separability and one-way functions. *Computational Complexity*, 11:137–157, 2003.

[Has89] J. Hastad. Almost optimal lower bounds for small depth circuits. *Advances in Computing Research*, 5:143–170, 1989.

[LV97] Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Graduate Texts in Computer Science. Springer, New York, second edition, 1997.

[MP91] N. Megiddo and C. Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, 81(2):317–324, 1991.

[MS72] A. Meyer and L. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Proceedings of the 13th IEEE Symposium on Switching and Automata Theory*, pages 125–129. IEEE, New York, 1972.

[Sip83] M. Sipser. Borel sets and circuit complexity. In *Proceedings of the 15th ACM Symposium on the Theory of Computing*, pages 61–69. ACM, New York, 1983.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'