Using Autoreducibility to Separate Complexity Classes

Harry Buhrman*
CWI
PO Box 94079
1090 GB Amsterdam
The Netherlands

Lance Fortnow†
University of Chicago
Department of Computer Science
1100 E. 58th St.
Chicago, IL 60637

Leen Torenvliet‡
University of Amsterdam
Plantage Muidergracht 24
1024 TV, Amsterdam
The Netherlands

Abstract

A language is autoreducible if it can be reduced to itself by a Turing machine that does not ask its own input to the oracle. We use autoreducibility to separate exponential space from doubly exponential space by showing that all Turing-complete sets for exponential space are autoreducible but there exists some Turing-complete set for doubly exponential space that is not. We immediately also get a separation of logarithmic space from polynomial space.

Although we already know how to separate these classes using diagonalization, our proofs separate classes solely by showing they have different structural properties, thus applying Post's Program to complexity theory. We feel such techniques may prove unknown separations in the future. In particular if we could settle the question as to whether all complete sets for doubly exponential time were autoreducible we would separate polynomial time from either logarithmic space or polynomial space.

We also show several other theorems about autoreducibility.

1 Introduction

While complexity theorists have made great strides in understanding the structure of complexity classes, they have not yet found the proper tools to do nontrivial separation of complexity classes such as **P** and **NP**. They have developed sophisticated diagonalization, combinatorial and algebraic techniques but none of these ideas have yet proven useful in the separation task.

Back in the early days of recursion theory, Post [Pos44] wanted to show that the set of nonrecursive recursively enumerable sets strictly contained the Turing-complete recursively enumerable sets. In what we now call "Post's Program" (see [Odi89, Soa87]), Post tried to show these classes differed by finding a property that holds for all sets in one class but not for some set in the second.

We would like to resurrect Post's Program for separating classes in complexity theory. In particular we will show how some classes differ by showing that their complete sets have different structure. While we do not separate any classes not already separable by known diagonalization techniques, we feel that refinements to our techniques may yield some new separation results.

In this paper we will concentrate on the property known as "autoreducibility." A set L is autoreducible if there is a polynomial-time oracle Turing machine M that accepts L using L as an oracle with the caveat that $M(x)$ may not query whether $x \in L$.

Trakhtenbrot [Tra70a] first looked at autoreducibility in both the recursion theory and resource-bounded models. Ladner [Lad73] showed that there existed Turing-complete recursively enumerable sets that are not autoreducible. Ambos-Spies [AS84] first transferred the notion of autoreducibility to the polynomial-time settings. More recently, Yao [Yao90] and Beigel and Feigenbaum [BF92] have studied a probabilistic variant of autoreducibility known as "coherence."

In this paper, we ask for what complexity classes do all the complete sets have the autoreducibility property. In particular we show:

- All Turing-complete sets for **EXPSPACE** are

*E-mail: buhrman@cwi.nl. Part of this research was done while visiting the Univ. Politècnica de Catalunya in Barcelona. Partially supported by the Dutch foundation for scientific research (NWO) through NFI Project ALADDIN, under contract number NF 62-376 and a TALENT stipend.

†Email: fortnow@cs.uchicago.edu. Supported in part by NSF grant CCR 92-53582.

‡E-mail: leen@fwi.uva.nl. Partially supported by HC&M grant nr. ERB4050PL93-0516.

autoreducible.

- There exists a (2-)Turing-complete set for **EEXPSPACE** that is not autoreducible.

As an immediate corollary we have **EXPSPACE** $\neq$ **EEXPSPACE** and thus that **L** $\neq$ **PSPACE**. Although we have known these separations via the usual space hierarchy theorems [HS65] our proof does not rely on diagonalization, rather separates the classes by showing that the classes have different structural properties.

To prove the first result we first give a new characterization of **EXPSPACE** based on work of Chandra, Kozen and Stockmeyer [CKS81], Simon [Sim75] and Orponen [Orp83]. We characterize **EXPSPACE** as a game between two exponential players playing for exponential moves with a polynomial-time judge.

Issues of relativization do not apply to this work because of oracle access (see [For94]), a polynomial-time autoreduction can not view as much as the oracle as an exponential or double exponential computation. To illustrate this point we show

- There exists an oracle relative to which there exist **EXPSPACE**-complete sets that are not autoreducible.

Note that if we can settle whether the Turing-complete sets for **EEXP** are all autoreducible one way or the other, we will have a major separation result. If all of the Turing-complete sets for **EEXP** are autoreducible then we get that **EEXP** $\neq$ **EEXPSPACE** and thus **P** $\neq$ **PSPACE**. If there exists a Turing-complete set for **EEXP** that is not autoreducible then we get that **EXPSPACE** $\neq$ **EEXP** and thus **L** $\neq$ **P**. Thus this autoreducibility question about **EEXP** becomes an exciting line of research.

In contrast to the above results we show the limitations of our approach:

- All $\leq^p_{2-tt}$-complete sets for **EXP**, **EEXP**, **EXPSPACE**, and **EEXPSPACE** are $\leq^p_{2-tt}$ autoreducible.
- All **#P**-robust sets are nonuniformly autoreducible.
- All complete sets for **EEXP** and **EEXPSPACE** are nonuniformly autoreducible.

2 Preliminaries

Let $\Sigma = \{0,1\}$. Strings are elements of Σ^*, and are denoted by small letters $x, y, u, v, \ldots$. For any string x, the length of x is denoted by $|x|$. Languages are subsets of Σ^*, and are denoted by capital letters $A, B, C, S, \ldots$. We assume that the reader is familiar with the standard Turing machine model and other standard notions of complexity theory, as can be found in [BDG88]. Nevertheless, some of the definitions that we feel may not be common knowledge, are cited below.

An *oracle* machine is a multitape Turing machine with an input tape, an output tape, work tapes, and a *query* tape. Oracle machines have three distinguished states QUERY, YES and NO, which are explained as follows: at some stage(s) in the computation the machine may enter the state QUERY, and then goes to the state YES, or goes to the state NO, depending on the membership of the string currently written on the query tape in a fixed *oracle* set. We will also say the machine received the answer 1 (0) when the machine goes to the YES(NO) state.

Oracle machines appear in the paper in two flavors: adaptive and non-adaptive. For a non-adaptive machine, queries may not be interdependent, whereas an adaptive machine may compute the next query depending on the answer to previous queries. If a Turing machine accepts (rejects) a string x, we will sometimes write $M(x) = 1$ ($M(x) = 0$). We use the same notation for oracle machines: $M^A(x) = 0/1$ or $M(x, A) = 0/1$. The set of strings recognized by a Turing (oracle) machine (with oracle A), is denoted by $L(M)$, $L(M, A)$.

We use polynomial time bounded adaptive oracle machines, to model Turing reductions ($\leq^p_T$) and non-adaptive machines to model *truth-table* reductions ($\leq^p_{tt}$). For polynomial time bounded machines, this yields definitions equivalent to the standard definitions of reducibilities in [LLS75].

The set of queries generated on input x by oracle machine M is denoted $Q_M(x)$. For adaptive machines, this set may be oracle dependent, and is therefore denoted $Q^A_M(x)$, where A is the oracle set. The (possibly exponential size) set of all *possible* queries generated by adaptive machine M on input x—also called the *query tree* of M on input x—is denoted $QT_M(x)$.

We will also use a structural property of sets. This property can be defined as a reductions of the set to itself. The property we will use is *autoreducibility.*

Definition 2.1 *A set A is* autoreducible *if and only if there exist a polynomial time machine M such that:*

1. $L(M, A) = A$.
2. $x \notin Q^A_M(x)$.

3 Positive Results

By characterizing **EXPSPACE** by an exponential game we show

Theorem 3.1 *Every polynomial-time Turing-complete set for* **EXPSPACE** *is autoreducible.*

To prove Theorem 3.1 we first give a new characterization of **EXPSPACE** that extends the alternating characterization of **PSPACE** due to Chandra, Kozen and Stockmeyer [CKS81], the oracle characterization of **NEXP** by Simon [Sim75] and the alternating oracle characterization of the exponential-time hierarchy due to Orponen [Orp83].

Let p be a polynomial and M an oracle Turing machine running in time $p(n)$. Let us also have two arbitrary players A and B that take turns deciding bits of an oracle D on strings of length $p(n)$. Once they have decided these $2^{p(n)}$ bits, M can then have random access to D as an oracle.

Theorem 3.2 *A language L is in* **EXPSPACE** *if and only if there exists a p and M as described above such that for all x,*

$$\begin{aligned} x \in L &\Rightarrow \exists A\, \forall B\ M^D(x) \text{ accepts} \\ x \notin L &\Rightarrow \exists B\, \forall A\ M^D(x) \text{ rejects} \end{aligned}$$

Sketch of Proof: Chandra, Kozen and Stockmeyer [CKS81] show that every language L in **EXPSPACE** is accepted by an alternating exponential time Turing machine N. Consider two players, A and B who play the existential and universal roles as follows: On input x, Player A writes down an initial configuration of $N(x)$. If the current configuration is in an existential state then player A writes down a valid next configuration of N. If the current configuration of $M(x)$ is a universal state then player B writes down a valid next configuration of N. Player A wins if the final configuration is in an accepting state.

A polynomial-time Turing machine cannot verify that this exponential-size "tableau" of configurations represents a valid accepting computation. Cook [Coo71] in his proof that Satisifiability is NP-complete, showed that a tableau has a "locally-checkable" property: Given a pointer to a mistake in a tableau, one can verify the mistake by checking only a small number of bits of the tableau.

If x is not in L then there must be some mistake in the tableau. After each player has played all the configurations, we then have player B point to the *first* mistake made by player A. Finally, we give player A one more chance by pointing out a mistake made by player B before the mistake of player A that B pointed to.

It is now easy to verify that $x \in L$ if and only if A has a winning strategy in this game. In order to meet the requirement that A and B only play one bit at a time we just have the other player play dummy bits when A and B play longer strings. □

One can think of A and B as strategies where A tries to make M accept and B tries to make M reject. One can then implement these strategies as functions that map x and $y_1, \ldots, y_k$ to $\{0,1\}$ where the y_i's are all the bits played so far. If $x \in L$ there exists a strategy for A such that for any strategy for B, M will accept. We call such a strategy a *winning* strategy for showing $x \in L$. Likewise if $x \notin L$ then there is a winning strategy for B.

The following corollary about the complexity of winning strategies falls out of the proof of Theorem 3.2.

Corollary 3.3 *There exists a polynomial q such that a winning strategy for A or B (depending on whether or not $x \in L$) is computable by a Turing machine using $2^{q(|x|)}$ space.*

Proof of Theorem 3.1: Suppose L is the Turing-complete set for **EXPSPACE** that we want to show autoreducible. Fix a set K many-one complete for **EXPSPACE**. Let M_K be the polynomial-time oracle Turing machine such that K is the language accepted by M_K^L.

Fix an input x. We will now describe the autoreduction to determine whether $x \in L$. Remember we are allowed to query L except for x. Let $L^+ = L \cup \{x\}$ and $L^- = L - \{x\}$

We define functions $A'(i)$ and $B'(i)$ that describe a game where $A'(i)$ is the ith move for player A and $B'(i)$ is the ith move for player B.

For A'(i):

- Look at but do not execute the following
  - Compute whether $x \in L$. If not play zero.
  - Otherwise compute recursively the first $i-1$ moves of A' and B'.
  - Play the winning strategy for A on this history.
- This computation runs in **EXPSPACE**. Reduce this computation question to whether a certain string $y \in K$.
- Return $M_K^{L^+}(y)$.

For B'(i):

- Look at but do not execute the following
  - Compute whether $x \in L$. If so play zero.
  - Otherwise compute recursively the first i moves of A' and $i-1$ moves of B'.
  - Play the winning strategy for B on this history.
- This computation runs in **EXPSPACE**. Reduce this computation question to whether a certain string $y \in K$.
- Return $M_K^{L^-}(y)$.

Note that A' and B' can both be computed in polynomial-time with access to an oracle L without querying x. The autoreduction then simulates $M(x)$ using A' and B' for the oracles.

If $x \in L$ then A' will play according to a winning strategy for A and will cause $M(x)$ to accept. If $x \notin L$ then B' will play according to a winning strategy for B and will cause $M(x)$ to reject. $\Box$

Similar though simpler proofs yield the following corollary:

Corollary 3.4 *All Turing-complete sets for* **PSPACE** *and* **EXP** *are autoreducible.*

Beigel and Feigenbaum [BF92] had previously shown that Turing complete sets for **PSPACE** as well as all the levels of the polynomial-time hierarchy are autoreducible.

We can get more stronger autoreducibilities of complete sets if we allow nonuniformity, i.e., a polynomial amount of *advice* (see [KL82]) that depends only on the input size.

Feigenbaum and Fortnow [FF93] define the following concept of #**P**-robustness: A set L is #**P**-*robust* if

$$\mathbf{P}^{\#\mathbf{P}^L} = \mathbf{P}^L$$

Theorem 3.5 *Every* #**P**-*robust language is nonuniformly autoreducible.*

Proof: Feigenbaum and Fortnow [FF93] show that every #**P**-robust language is random-self-reducible. Beigel and Feigenbaum [BF92] show that every random-self-reducible set is "weakly coherent" where *weakly coherent* means nonuniformly probabilistically autoreducible. One can then amplify the probability of correctness so that one random string works for all inputs and then add that random string to the advice. $\Box$

Since **NEXP**, **EEXP** and **EEXPSPACE** complete sets are #**P**-robust we get

Corollary 3.6
Every Turing-complete set for **NEXP**, **EEXP** *and* **EEXPSPACE** *is nonuniformly autoreducible.*

4 Negative Results

In this section we will construct a set that is complete for **EEXPSPACE** but not autoreducible. This together with Theorem 3.1 shows a structural difference between sets complete for **EXPSPACE** and **EEXPSPACE**.

Theorem 4.1 *There exists a 2-Turing complete set* A *for* **EEXPSPACE** *that is not autoreducible.*

Proof: The construction will need to satisfy two requirements. We will have to diagonalize against all autoreductions ensuring at the same time that the constructed set remains complete. We will diagonalize in intervals. We will need the following function in the construction:

$$b(n) = \begin{cases} 1 & \text{if } n = 0 \\ b(n-1)^{(n-1)} + 1 & \text{otherwise} \end{cases}$$

In order to make A complete we will make sure that K – a standard $\leq_m^p$-complete set for **EEXPSPACE** – reduces to A with a 2-Turing reduction. The reduction will behave as follows:

input x
Let i be such that $b(i) \leq |x| < b(i+1)$
if $0^{b(i)} \in A$ **then** accept x iff $<1,x> \in A$
else accept x iff $<0,x> \in A$

We will call the Turing machine that performs this $\leq_{2-T}^p$ reduction M_r.

We assume that the i^{th} polynomial time autoreduction runs in time n^i, on inputs of length n. We assume the following ordering on strings: 0 is the first, 1 is the second, 00 is the third etc. In this ordering x_i denotes the i^{th} string. We will construct A in stages, but first we will need the following general lemma concerning autoreductions. The lemma says, that there is a way of coding K correctly and still keeping an autoreduction accepting or rejecting.

Lemma 4.2 *For all* i *and* k, *let* $A_0 \subseteq \Sigma^{\leq b(i)-1}$. *Let* $b_j \in \{0,1\}$. *Assume that* $M(0^{b(i)})$ *is an autoreduction that queries at most* $2k$ *strings in its entire query tree*[1]. *It will be the case that:*

[1] We can assume that every autoreduction only queries strings of the form $<1,x>$ and $<0,x>$. Moreover we also assume that whenever $<i,y>$ is queried also $<(1-i),y>$ is queried $(i = 0, 1)$.

$\forall b_{j_1} \exists S_1 \subseteq \{<0, x_{j_1}>, <1, x_{j_1}>\} \ldots \forall b_{j_m} \exists S_m \subseteq \{<0, x_{j_m}>, <1, x_{j_m}>\}$, $(j_l \geq 2^{b(i)}, 1 \leq l \leq m, m \leq 2^{b(i)^i})$, $A = S_1 \cup \ldots \cup S_m \cup A_0$ *such that either:*

(a)
- $M(0^{b(i)}, A) = 0$, *and*
- $M_r(x_{j_l}, A) = b_{j_l}$, $(1 \leq l \leq m)$, *and*
- $0^{b(i)} \in A$

or

(b)
- $M(0^{b(i)}, A) = 1$, *and*
- $M_r(x_{j_l}, A) = b_{j_l}$, $(1 \leq l \leq m)$, *and*
- $0^{b(i)} \notin A$

We will call autoreductions that satisfy (b) of type 0 and the ones that satisfy (a) type 1.

Proof Sketch of Lemma 4.2: We will use induction on k. The statement is true for $k = 0$ since the autoreduction that queries no strings at all is either of type 0 or of type 1 (depending on whether it accepts $0^{b(i)}$). Assume the lemma is true for k. Assume $M(0^{b(i)})$ queries $2k+2$ strings. Assume $<1, x_a>$ and $<0, x_a>$ are the first queries. Let M_{ij} (with $i, j \in \{0, 1\}$) correspond to the computation of $M(0^{b(i)})$ with $<0, x_a>$ answered i and $<1, x_a>$ answered j (without querying $<0, x_a>$ and $<1, x_a>$). Using the induction hypothesis it follows that M_{00}, M_{01}, M_{10} and M_{11} have either type 0 or type 1. It is not hard to see that by doing a case analysis one can always code x_a according to b_a into A keeping $M(0^{b(i)})$ of type 1 or 0 at the same time. □

The actual construction of A is as follows:

stage i:

TYPE := type of $M_i(0^{b(i)})$ * either 0 or 1 *
put $0^{b(i)}$ in A iff TYPE = 1
for all x such that $b(i) < |x| \leq b(i)^i$ do:
      Put $<\mathrm{TYPE}, x>$ in A iff $x \in K$
(*)   Keep $M_i(0^{b(i)})$ of type TYPE by
      putting $<(1 - TYPE), x>$ in or out of A

end of stage i

Line (*) in the above construction can be done because of Lemma 4.2. The computation whether $M_i(0^{b(i)})$ is of type 0 or 1 can be done within 2^{n^i} alternating time and hence in double exponential space. Since **EEXPSPACE** is closed under complementation, the computation whether $x \in K$ can also be carried out in double exponential space and it thus follows that $A \in$ **EEXPSPACE**. Moreover $K \leq^p_{2-T} A$. □

Note that Corollary 3.6 contrasts this theorem. It shows that it is impossible to diagonalize against nonuniform autoreductions, keeping the construction complete at the same time.

We would like to refine the above construction such that A is computable in **EEXP**, with K playing the role of a standard complete set for **EEXP**. The main problem is that it seems to be hard to figure out whether an autoreduction is of type 0 or 1. (See Section 6.) However tweaking the function $b(n)$, one can easily adapt the above construction so that it can be carried out in double exponential time with a non-constant number of alternations.

Corollary 4.3 *There exists a $\leq^p_{2-T}$-complete set A for double exponential time with a non-constant number alternations, that is not autoreducible.*

Corollary 4.4 L *is different from polynomial time with a non-constant number of alternations.*

Another nice feature of our construction is that Theorem 3.1 and Corollary 3.4 for **EXP** do not relativize.

Theorem 4.5 *There exists an oracle A such that not all sets complete for* $\mathbf{EXP}^A$ *and* $\mathbf{EXPSPACE}^A$ *are autoreducible relative to A.*

Proof: The construction of the oracle A will parallel the construction of Theorem 4.1. The oracle will contain information on when autoreduction $M_i(0^{b(i)})$ is of type 0 and of type 1. This information will be coded high enough in A such that $M_i(0^{b(i)})$ can not reach it, because it can not ask long queries. On the other hand an exponential time or space machine can now compute the type of autoreductions and hence the construction of Theorem 4.1 can be carried out in exponential time or exponential space. □

5 Nonadaptive Autoreductions

The results in the previous sections go through with respect to truth-table (nonadaptive) reductions. Using a logtime oracle characterization analogous to the one for **EXPSPACE** (Theorem 3.2), we can show the following.

Theorem 5.1 *All sets $\leq^p_{tt}$-complete for* **PSPACE** *are $\leq^p_{tt}$ autoreducible.*

Proof: The proof is analogous to the proof of Theorem 3.1. Details will be in the full version of this paper. □

Considering truth-table reductions also brings down Theorem 4.1 an exponent.

Theorem 5.2 *There exists a $\leq_{tt}^p$-complete set for* **EXPSPACE** *that is not $\leq_{tt}^p$ autoreducible.*

Proof: The proof is analogous to the proof of Theorem 4.1 using the fact that a truth-table autoreduction only queries polynomial many queries in its entire query tree, hence it only costs super polynomial alternating time to compute the type of a $\leq_{tt}^p$ autoreduction. □

The above two theorems show that the question whether all $\leq_{tt}^p$-complete sets for **EXP** are $\leq_{tt}^p$ autoreducible is very interesting. An answer would prove either **PSPACE** $\neq$ **EXP** or **P** $\neq$ **PSPACE**.

Corollary 3.6 also finds it counterpart for non-adaptive autoreductions.

Theorem 5.3 *All $\leq_{tt}^p$-complete sets for* **NP** *are $\leq_{tt}^p$ autoreducible with respect to nonuniform reductions.*

Proof Sketch: We use techniques from [BvHT93]. Let L be a $\leq_{tt}^p$-complete set for **NP**. On input x one can compute relative to L a witness y_x that witnesses that $x \in L$. Whenever x is queried in this computation answer 1 ($x \in L$). If a witness y_x is computed accept x otherwise reject. This procedure is an adaptive autoreduction. We now use techniques from [VV86] in order to find a witness using non-adaptive queries to L and polynomially many random bits. Since this computation can be amplified, the construction can be derandomized and becomes nonuniform. □

The following theorem shows another technique that enables one to show that $\leq_{2-tt}^p$-complete sets are $\leq_{2-tt}^p$ autoreducible.

Theorem 5.4 *If C is a $\leq_{2-tt}^p$-complete set for* **EXP** *then C is 2-truth-table autoreducible.*

Proof: We assume an enumeration of $\leq_{2-tt}^p$ reductions $M_1, M_2, \ldots$ We construct a set D in **EXP**. Since C is $\leq_{2-tt}^p$-complete for **EXP**, $D \leq_{2-tt}^p C$. Let M_j witness this fact. Now we will construct, simulating M_j on input $<j,x>$, an autoreduction on x for C.

The following algorithm will define D. Simulate $M_i(<i,x>)$. There are several cases:

1. $M_i(<i,x>)$ accepts: reject $<i,x>$
2. $M_i(<i,x>)$ rejects: accept $<i,x>$
3. $x \notin Q_{M_i}(<i,x>)$: accept iff $x \in C$
4. $\|Q_{M_i}(<i,x>)\| = 1$ and $x \in Q_{M_i}(<i,x>)$: accept iff $x \notin C$
5. $\|Q_{M_i}(<i,x>)\| = 2$ and $x \in Q_{M_i}(<i,x>)$. Let y be the other query. Compute whether $x \in C$. Substitute this answer in the oracle computation of M_i. So $M_i(<i,x>)$ can be seen as a 1 truth-table reduction, depending only on y. There are again several cases:
   (a) $M_i(<i,x>)$ accepts: reject $<i,x>$
   (b) $M_i(<i,x>)$ rejects: accept $<i,x>$
   (c) $M_i(<i,x>)$ accepts iff y is in the oracle set: accept iff $x \in C$
   (d) $M_i(<i,x>)$ rejects iff y is in the oracle set: accept iff $x \notin C$

It is not hard to see that $D \in$ **EXP**, so $D \leq_{2-tt}^p C$. Let M_j witness this reduction. The autoreduction is the following algorithm. On input x simulate $M_j(<j,x>)$. If it is the case that $x \notin Q_{M_j}(<j,x>)$ then use C as an oracle to compute the reduction. We are in case 3 in the definition of D and $<j,x> \in D$ iff $x \in C$ iff $M_j^C(<j,x>)$ accepts. If this is not the case (i.e. $x \in Q_{M_j}(<j,x>)$) let y be the other query we arranged D so that the only remaining cases are 5c and 5d. In both cases we arranged it so that $x \in C$ iff $y \in C$. Thus in this case accept iff $y \in C$. Furthermore observe that this autoreduction is in fact a 2-truth-table reduction. □

As a corollary to the proof we get

Corollary 5.5
All $\leq_{2-tt}^p$ -complete sets for **EXPSPACE**, **EEXP**, *and* **EEXPSPACE** *are $\leq_{2-tt}^p$ autoreducible.*

This corollary shows that the coding in Theorem 4.1 can not be done via a $\leq_{2-tt}^p$ reduction. It also shows a structural difference between $\leq_{2-T}^p$-complete sets and $\leq_{2-tt}^p$-complete sets.

Corollary 5.6 *[BST93]*
There exists a $\leq_{2-T}^p$-complete set for **EXPSPACE** *that is not $\leq_{2-tt}^p$ -complete.*

Proof: Use A in Theorem 5.2 together with Corollary 5.5. □

6 Conclusions

We believe that this research may lead to a separation of classes not separable by known diagonalization techniques. We would like to mention a few words about some thoughts in this direction.

One does not have to look at just complete sets. Suppose we could construct a set that is Turing hard for **EXPSPACE** but lies inside the double exponential-time hierarchy. This would separate not only the classes **EXPSPACE** from the double exponential-time hierarchy but also **L** from **NP**.

One would hope on a variation of Lemma 4.2 where the construction of A could occur in the double exponential-time hierarchy. Unfortunately such coding tricks seem to be $\mathbf{DSPACE}(m)$-complete for coding $m = 2^{n^i}$ potential queries.

Given m bits $w_1, ..., w_m$ and a polynomial-time boolean function $f(z)$ on $2m$ bits where z represents $(x_1, y_1, x_2, y_2, ..., x_m, y_m)$. We want to pick a z such that

(a) For all i, $w_i = x_i$ and $f(z) = 0$, or

(b) For all i, $w_i = y_i$ and $f(z) = 1$.

Say f is type 0 if (a) occurs and type 1 if (b) occurs.

As an added problem, we want to do this online, i.e. given each bit w_i we need to decide on x_i and y_i before we get w_{i+1}. In addition we need to decide on the type of f before we see any of the w's.

In order to choose f to be type 0 the following must hold:

$$\forall x_1 \exists y_1 \forall x_2 \exists y_2 ... \forall x_m \exists y_m f(z) = 0 \qquad (1)$$

For f to be type 1 we get:

$$\forall y_1 \exists x_1 \forall y_2 \exists x_2 ... \forall y_m \exists x_m f(z) = 1 \qquad (2)$$

Note that by De Morgan's laws and the fact that

$$\exists x_i \forall y_i \Rightarrow \forall y_i \exists x_i$$

we get that the negation of Equation (1) implies Equation (2). This is essentially what we have proven by induction in the proof of Lemma 4.2. Note that it is possible that for some f both Equations (1) and (2) could hold.

Now suppose we have a quantified boolean formula of the following form (assume m even):

$$\psi = \exists x_1 \forall y_2 \exists x_3 ... \forall y_m \phi(x_1, y_2, ..., y_m).$$

Note that if we use ϕ as our f above, we get that

- ψ is true implies Equation (2) holds and Equation (1) fails, and
- ψ is false implies Equation (1) holds and Equation (2) fails.

Since $\mathbf{QBF}_m$, the set of quantified boolean formulas with m quantifiers, is $\mathbf{DSPACE}(m)$-complete under polynomial-time reductions, it is $\mathbf{DSPACE}(m)$-complete to determine whether f is type 0 or type 1 and it thus could be $\mathbf{DSPACE}(m)$-complete to code.

Generalizations of autoreducibility may give us the coding power we need. For example, one could look at $k(n)$-autoreducibility where $k(n)$ bits of the language remain unknown to the querying machine. With some tricks, Theorem 3.1 would go through for $k(n) = O(\log n)$. Perhaps one could use this generalized model to get an appropriate non-autoreducible set to separate classes.

Finally, perhaps one could use a property other than autoreducibility to separate classes. One possibility is mitoticity, a property closely related to autoreducibility [Lad73, AS84, BHT94]. A set is mitotic if it is the disjoint union of two Turing-equivalent sets. Perhaps mitoticity or some other natural or artificial property can be used to separate classes.

Acknowledgments

We would like to thank Manindra Agrawal and Ashish Naik for very helpful discussions.

References

[AS84] K. Ambos-Spies. p-mitotic sets. In E. Börger, G. Hasenjäger, and D. Roding, editors, *Logic and Machines*, volume 177 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 1984.

[BDG88] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I.* Springer, 1988.

[BF92] R. Beigel and J. Feigenbaum. On being incoherent without being very hard. *Computational Complexity*, 2:1–17, 1992.

[BHT94] H. Buhrman, A. Hoene, and L. Torenvliet. Splittings, robustness and structure of complete sets. In *Proceedings of the 10th Symposium on Theoretical Aspects of Computer Science*, volume 665 of *Lecture Notes in Computer Science*, pages 175–184. Springer, 1994.

[BST93] H. Buhrman, E. Spaan, and L. Torenvliet. Bounded reductions. In K. Ambos-Spies, S. Homer, and U. Schöning, editors, *Complexity Theory*, pages 83–99. Cambridge University Press, December 1993.

[BvHT93] H. Buhrman, P. van Helden, and L. Torenvliet. P-selective self-reducible sets: A new characterization of P. In *Proceedings of the 8th IEEE Structure in Complexity Theory Conference*, pages 44–51. IEEE, New York, 1993.

[CKS81] A. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

[Coo71] S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd ACM Symposium on the Theory of Computing*, pages 151–158. ACM, New York, 1971.

[FF93] J. Feigenbaum and L. Fortnow. On the random-self-reducibility of complete sets. *SIAM Journal on Computing*, 22:994–1005, 1993.

[For94] L. Fortnow. The role of relativization in complexity theory. *Bulletin of the European Association for Theoretical Computer Science*, 52:229–244, February 1994.

[HS65] J. Hartmanis and R. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[KL82] R. Karp and R. Lipton. Turing machines that take advice. *L'Enseignement Mathematique*, 28:191–209, 1982.

[Lad73] R. Ladner. Mitotic recursively enumerable sets. *Journal of Symbolic Logic*, 38(2):199–211, 1973.

[LLS75] R. Ladner, N. Lynch, and A. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1:103–123, 1975.

[Odi89] P. Odifreddi. *Classical Recursion Theory*, volume 125 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 1989.

[Orp83] P. Orponen. Complexity classes of alternating machines with oracles. In *Proceedings of the 10th International Colloquium on Automata, Languages and Programming*, volume 154 of *Lecture Notes in Computer Science*, pages 573–584. Springer, 1983.

[Pos44] E. Post. Recursively enumerable sets of positive integers and their decision problems. *Bulletin of the American Mathematical Society*, 50:284–316, 1944.

[Sim75] J. Simon. *On Some Central Problems in Computational Complexity.* PhD thesis, Cornell University, 1975. Technical Report TR 75-224.

[Soa87] R. Soare. *Recursively Enumerable Sets and Degrees.* Springer, Berlin, 1987.

[Tra70a] B. Trakhtenbrot. On autoreducibility. *Dokl. Akad. Nauk SSSR*, 192:1224–1227, 1970. In Russian. English Translation in [Tra70b].

[Tra70b] B. Trakhtenbrot. On autoreducibility. *Soviet Math. dokl.*, 11:814–817, 1970.

[VV86] L. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.

[Yao90] A. Yao. Coherent functions and program checkers. In *Proceedings of the 22nd ACM Symposium on the Theory of Computing*, pages 84–94. ACM, New York, 1990.