Separating Complexity Classes using Autoreducibility

Harry Buhrman* — CWI
Lance Fortnow† — University of Chicago
Dieter van Melkebeek‡ — University of Chicago
Leen Torenvliet§ — University of Amsterdam

Abstract

A set 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 the polynomial-time hierarchy from exponential space by showing that all Turing-complete sets for certain levels of the exponential-time hierarchy are autoreducible but there exists some Turing-complete set for doubly exponential space that is not.

Although we already knew 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 Turing-complete sets for doubly exponential time are autoreducible, we would separate either polynomial time from polynomial space, and nondeterministic logarithmic space from nondeterministic polynomial time, or else the polynomial-time hierarchy from exponential time.

We also look at the autoreducibility of complete sets under nonadaptive, bounded query, probabilistic and nonuniform reductions. We show how settling some of these autoreducibility questions will also lead to new complexity class separations.

Key words: complexity classes, completeness, autoreducibility, coherence.

AMS classification: 68Q15, 68Q05, 03D15.

*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 NF 62-376 and a TALENT stipend. E-mail: buhrman@cwi.nl. URL: http://www.cwi.nl/~buhrman.

†Supported in part by NSF grant CCR 92-53582. Research partly done while visiting CWI. Email: fortnow@cs.uchicago.edu. URL: http://www.cs.uchicago.edu/~fortnow.

‡Supported in part by the NSF through grant CCR 92-53582, by the European Union through TMR grant ERB-4001-GT-96-0783 while visiting CWI and the University of Amsterdam, and by the Fields Institute while visiting the Fields Institute at the University of Toronto. Email: dieter@cs.uchicago.edu. URL: http://www.cs.uchicago.edu/~dieter.

§Partially supported by HC&M grant ERB4050PL93-0516. Email: leen@wins.uva.nl. URL: http://turing.wins.uva.nl/~leen.

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 very useful in the separation task.

Back in the early days of computability theory, Post [13] wanted to show that the set of noncomputable computably enumerable sets strictly contains the Turing-complete computably enumerable sets. In what we now call "Post's Program" (see [11, 15]), Post tried to show these classes differ by finding a property that holds for all sets in the first 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 A is autoreducible if we can decide whether an input x belongs to A in polynomial-time by making queries about membership of strings different from x to A.

Trakhtenbrot [16] first looked at autoreducibility in both the unbounded and space-bounded models. Ladner [10] showed that there exist Turing-complete computably enumerable sets that are not autoreducible. Ambos-Spies [1] first transferred the notion of autoreducibility to the polynomial-time setting. More recently, Yao [19] and Beigel and Feigenbaum [5] 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 Δ_k^{EXP} are autoreducible for any constant k, where $\Delta_{k+1}^{\mathrm{EXP}}$ denotes the sets that are exponential-time Turing-reducible to Σ_k^{P}.
- There exists a Turing-complete set for doubly exponential space that is not autoreducible.

Since the union of all sets Δ_k^{EXP} coincides with the exponential-time hierarchy, we obtain a separation of the exponential-time hierarchy from doubly exponential space and thus of the polynomial-time hierarchy from exponential space. Although these results also follow from the space hierarchy theorems [9] which we have known for a long time, our proof does not directly use diagonalization, rather separates the classes by showing that they have different structural properties.

Issues of relativization do not apply to this work because of oracle access (see [8]): A polynomial-time autoreduction cannot view as much of the oracle as an exponential or doubly exponential computation. To illustrate this point we show that there exists an oracle relative to which some complete set for exponential time is not autoreducible.

Note that if we can settle whether the Turing-complete sets for doubly exponential time are all autoreducible one way or the other, we will have a major separation result. If there exists a Turing-complete set for doubly exponential time that is not autoreducible, then we get that the exponential-time hierarchy is strictly contained in doubly exponential time thus that the polynomial-time hierarchy is strictly contained in exponential time. If all of the Turing-complete sets for doubly exponential time are autoreducible, we get that doubly exponential time is strictly contained in doubly exponential space, and thus polynomial time strictly in polynomial space. We will also show that this assumption implies a separation of nondeterministic logarithmic space from nondeterministic polynomial time. Similar implications hold for space bounded classes (see Section

5). Autoreducibility questions about doubly exponential time and exponential space thus remain an exciting line of research.

We also study the nonadaptive variant of the problem. Our main results scale down one exponential as follows:

- All truth-table-complete sets Δ_k^{P} are truth-table-autoreducible for any constant k, where $\Delta_{k+1}^{\mathrm{P}}$ denotes that sets polynomial-time Turing-reducible to Σ_k^{P}.
- There exists a truth-table-complete set for exponential space that is not truth-table-autoreducible.

Again, finding out whether all truth-table-complete sets for intermediate classes, namely polynomial space and exponential time, are truth-table-autoreducible, would have major implications.

In contrast to the above results we exhibit the limitations of our approach: For the restricted reducibility where we are only allowed to ask two nonadaptive queries, all complete sets for EXP, EXPSPACE, EEXP, EEXPSPACE, etc., are autoreducible.

We also argue that uniformity is crucial for our technique of separating complexity classes, because our nonautoreducibility results fail in the nonuniform setting. Razborov and Rudich [14] show that if strong pseudo-random generators exist, "natural proofs" cannot separate certain nonuniform complexity classes. Since this paper relies on uniformity in an essential way, their result does not apply.

Regarding the probabilistic variant of autoreducibility mentioned above, we can strengthen our results and construct a Turing-complete set for doubly exponential space that is not even probabilistically autoreducible. We leave the analogue of this theorem in the nonadaptive setting open: Does there exist a truth-table-complete set for exponential space that is not probabilistically truth-table autoreducible? We do show that every truth-table-complete set for exponential time is probabilistically truth-table autoreducible. So, a positive answer to the open question would establish that exponential time is strictly contained in exponential space. A negative answer, on the other hand, would imply a separation of nondeterministic logarithmic space from nondeterministic polynomial time.

Here is the outline of the paper: First, we introduce our notation and state some preliminaries in Section 2. Next, in Section 3 we establish our negative autoreducibility results, for the adaptive as well as the nonadaptive case. Then we prove the positive results in Section 4, where we also briefly look at the randomized and nonuniform settings. Section 5 discusses the separations that follow from our results and would follow from improvements on them. Finally, we conclude in Section 6 and mention some possible directions for further research.

1.1 Errata to conference version

A previous version of this paper [6] erroneously claimed proofs showing all Turing-complete sets for EXPSPACE are autoreducible and all truth-table-complete sets for PSPACE are nonadaptively autoreducible. Combined with the additional results in this version, we would have a separation of NL and NP (see Section 5).

However the proofs in the earlier version failed to account for the growth of the running time when recursively computing previous players' moves. We use the proof technique in Section 3 though unfortunately we get weaker theorems. The original results claimed in the previous version remain important open questions as resolving them either way will yield new separation results.

2 Notation and Preliminaries

Most of our complexity theoretic notation is standard. We refer the reader to the textbooks by Balcázar, Díaz and Gabarró [4, 3], and by Papadimitriou [12].

We use the binary alphabet $\Sigma = \{0, 1\}$. We denote the difference of a set A with a set B, i.e., the subset of elements of A that do not belong to B, by $A \setminus B$.

For any integer $k \geqslant 0$, a Σ_k-formula is a Boolean expression of the form

$$\exists\, y_1 \in \Sigma^{n_1}, \forall\, y_2 \in \Sigma^{n_2}, \ldots, Q_k\, y_k \in \Sigma^{n_k} : \phi(y_1, y_2, \ldots, y_k, z), \tag{1}$$

where ϕ is a Boolean formula, Q_i denotes $\exists$ if i is odd, and $\forall$ otherwise, and the n_i's are positive integers. We say that (1) has $k-1$ alternations. A Π_k-formula is just like (1) except starts with a $\forall$-quantifier. It also has $k-1$ alternations. A QBF_k-formula is a Σ_k-formula (1) or a Π_k-formula without free variables z.

For any integer $k \geqslant 0$, Σ_k^{P} denotes the k-th Σ-level of the polynomial-time hierarchy. We define these levels recursively by $\Sigma_0^{\mathrm{P}} = \mathrm{P}$, and $\Sigma_{k+1}^{\mathrm{P}} = \mathrm{NP}^{\Sigma_k^{\mathrm{P}}}$. The Δ-levels of the polynomial-time and exponential-time hierarchy are defined as $\Delta_{k+1}^{\mathrm{P}} = \mathrm{P}^{\Sigma_k^{\mathrm{P}}}$ respectively $\Delta_{k+1}^{\mathrm{EXP}} = \mathrm{EXP}^{\Sigma_k^{\mathrm{P}}}$. The polynomial-time hierarchy PH equals the union of all sets Δ_k^{P}, and the exponential-time hierarchy EXPH similarly the union of all sets Δ_k^{EXP}.

A *reduction* of a set A to a set B is a polynomial-time oracle Turing machine M such that $M^B = A$. We say that A reduces to B and write $A \leqslant_{\mathrm{T}}^{\mathrm{P}} B$ ("T" for Turing). The reduction M is *nonadaptive* if the oracle queries M makes on any input are independent of the oracle, i.e., the queries do not depend upon the answers to previous queries. In that case we write $A \leqslant_{\mathrm{tt}}^{\mathrm{P}} B$ ("tt" for truth-table). Reductions of functions to sets are defined similarly. If the number of queries on an input of length n is bounded by $q(n)$, we write $A \leqslant_{q(n)-\mathrm{T}}^{\mathrm{P}} B$ respectively $A \leqslant_{q(n)-\mathrm{tt}}^{\mathrm{P}} B$; if it is bounded by some constant, we write $A \leqslant_{\mathrm{btt}}^{\mathrm{P}} B$ ("b" for bounded). We denote the set of queries of M on input x with oracle B by $Q_{M^B}(x)$; in case of nonadaptive reductions, we omit the oracle B in the notation. If the reduction asks only one query and answers the answer to that query, we write $A \leqslant_{\mathrm{m}}^{\mathrm{P}} B$ ("m" for many-one).

For any reducibility $\leqslant_r^{\mathrm{P}}$ and any complexity class $\mathcal{C}$, a set C is $\leqslant_r^{\mathrm{P}}$*-hard* for $\mathcal{C}$ if we can $\leqslant_r^{\mathrm{P}}$-reduce every set $A \in \mathcal{C}$ to C. If in addition $C \in \mathcal{C}$, we call C $\leqslant_r^{\mathrm{P}}$*-complete* for $\mathcal{C}$. For any integer $k \geqslant 0$, the set TQBF_k of all true QBF_k-formulae is $\leqslant_{\mathrm{m}}^{\mathrm{P}}$-complete for Σ_k^{P}. For $k = 1$, this reduces to the fact that the set SAT of satisfiable Boolean formulae is $\leqslant_{\mathrm{m}}^{\mathrm{P}}$-complete for NP.

Now we get to the key concept of this paper:

Definition 2.1 *A set A is* autoreducible *if there is a reduction M of A to itself that never queries its own input, i.e., for any input x and any oracle B, $x \notin Q_{M^B}(x)$. We call such M an* autoreduction *of A.*

We will also discuss randomized and nonuniform variants. A set is *probabilistically autoreducible* if it has a probabilistic autoreduction with bounded two-sided error. Yao [19] first studied this concept under the name "coherence". A set is *nonuniformly autoreducible* if it has an autoreduction that uses polynomial advice. For all these notions, we can consider both the adaptive and the nonadaptive case. For randomized autoreducibility, nonadaptiveness means that the queries only depend on the input and the random seed.

3 Nonautoreducibility Results

In this section, we show that large complexity classes have complete sets that are not autoreducible.

Theorem 3.1 *There is a* $\leqslant^{\mathrm{P}}_{2-\mathrm{T}}$*-complete set for* EEXPSPACE *that is not autoreducible.*

Most natural classes containing EEXPSPACE, e.g., EEEXPTIME and EEEXPSPACE, also have this property.

We can even construct the complete set in Theorem 3.1 to defeat every probabilistic autoreduction:

Theorem 3.2 *There is a* $\leqslant^{\mathrm{P}}_{2-\mathrm{T}}$*-complete set for* EEXPSPACE *that is not probabilistically autoreducible.*

In the nonadaptive setting, we obtain:

Theorem 3.3 *There is a* $\leqslant^{\mathrm{P}}_{3-\mathrm{tt}}$*-complete set for* EXPSPACE *that is not nonadaptively autoreducible.*

Unlike the case of Theorem 3.1, our construction does not seem to yield a truth-table-complete set that is not probabilistically nonadaptively autoreducible. In fact, as we shall show in Section 4.3, such a result would separate EXP from EXPSPACE. See also Section 5.

We will detail in Section 4.3 that our nonautoreducibility results do not hold in the nonuniform setting.

3.1 Adaptive Autoreductions

Suppose we want to construct a nonautoreducible Turing-complete set for a complexity class $\mathcal{C}$, i.e., a set A such that:

1. A is not autoreducible.
2. A is Turing-hard for $\mathcal{C}$.
3. A belongs to $\mathcal{C}$.

If $\mathcal{C}$ has a $\leqslant^{\mathrm{P}}_{\mathrm{m}}$-complete set K, realizing goals 1 and 2 is not too hard: We can encode K in A, and at the same time diagonalize against all autoreductions. A straightforward implementation would be to encode $K(y)$ as $A(\langle 0, y\rangle)$, and stage-wise diagonalize against all $\leqslant^{\mathrm{P}}_{\mathrm{T}}$-reductions M by picking for each M an input x not of the form $\langle 0, y\rangle$ that is not queried during previous stages, and setting $A(x) = 1 - M^A(x)$. However, this construction does not seem to achieve the third goal. In particular, deciding the membership of a diagonalization string x to A might require computing $A(\langle 0, y\rangle) = K(y)$ on inputs y of length $|x|^c$, assuming M runs in time n^c. Since we have to do this for all potential autoreductions M, we can only bound the resources (time, space) needed to decide A by a function in $t(n^{\omega(1)})$, where $t(n)$ denotes the amount of resources some deterministic Turing machine accepting K uses. That does not suffice to keep A inside $\mathcal{C}$.

To remedy this problem, we will avoid the need to compute $K(y)$ on large inputs y, say of length at least $|x|$. Instead, we will make sure we can encode the membership of such strings to *any* set, not just K, and at the same time diagonalize against M on input x. We will argue that we can do this by considering two possible coding regions at every stage as opposed to a fixed one: the left region L, containing strings of the form $\langle 0, y\rangle$, and the right region R, similarly containing strings of the form $\langle 1, y\rangle$. The following states that we can use one of the regions to encode an arbitrary sequence, and set the other region such that the output of M on input x is fixed and indicates the region used for encoding.

Statement 3.1 *Either it is the case that for any setting of L there is a setting of R such that $M^A(x)$ accepts, or for any setting of R there is a setting of L such that $M^A(x)$ rejects.*

This allows us to achieve goals 1 and 2 from above as follows. In the former case, we will set $A(x) = 0$ and encode K in L (at that stage); otherwise we will set $A(x) = 1$ and encode K in R. Since the value of $A(x)$ does not affect the behavior of M^A on input x, we diagonalize against M in both cases. Also, in any case

$$K(y) = A(\langle A(x), y\rangle),$$

so deciding K is still easy when given A. Moreover — and crucially — in order to compute $A(x)$, we no longer have to decide $K(y)$ on large inputs y, of length $|x|$ or more. Instead, we have to check whether the former case in Statement 3.1 holds or not. Although quite complex a task, it only depends on M and on the part of A constructed so far, not on the value of $K(y)$ for any input of length $|x|$ or more: We verify whether we can encode *any* sequence, not just the characteristic sequence of K for lengths at least $|x|$, and at the same time diagonalize against M on input x. Provided the complexity class $\mathcal{C}$ is sufficiently powerful, we can perform this task in $\mathcal{C}$.

There is still a catch, though. Suppose we have found out that the former case in Statement 3.1 holds. Then we will use the left region L to encode K (at that stage), and we know we can diagonalize against M on input x by setting the bits of the right region R appropriately. However, deciding exactly how to set these bits of the noncoding region requires, in addition to determining which region we should use for coding, the knowledge of $K(y)$ for all y such that $|x| \leqslant |y| \leqslant |x|^c$. In order to also circumvent the need to decide K for too large inputs here, we will use a slightly stronger version of Statement 3.1 obtained by grouping quantifiers into blocks and rearranging them. We will partition the coding and noncoding regions into intervals. We will make sure that for any given interval, the length of a string in that interval (or any of the previous intervals) is no more than the square of the length of any string in that interval. Then we will block-wise alternately set the bits in the coding region according to K, and the corresponding ones in the noncoding region so as to maintain the diagonalization against M on input x as in Statement 3.1. This way, in order to compute the bit $A(\langle 1, z\rangle)$ of the noncoding region, we will only have to query K on inputs y with $|y| \leqslant |z|^2$, as opposed to $|y| \leqslant |z|^c$ for an arbitrarily large c depending on M as was the case before.

This is what happens in the next lemma, which we prove in a more general form, because we will need the generalization later on in Section 5.

Lemma 3.4 *Fix a set K, and suppose we can decide it simultaneously in time $t(n)$ and space $s(n)$. Let $\alpha : \mathbb{N} \to (0, \infty)$ be a constructible monotone unbounded function, and suppose there is a deterministic Turing machine accepting TQBF that takes time $t'(n)$ and space $s'(n)$ on QBF-formulae of size $2^{n^{\alpha(n)}}$ with at most $\log \alpha(n)$ alternations. Then there is a set A such that:*

1. *A is not autoreducible.*
2. *$K \leqslant^{\mathrm{P}}_{2-\mathrm{T}} A$.*
3. *We can decide A simultaneously in time $O(2^{n^2} \cdot t(n^2) + 2^n \cdot t'(n))$ and space $O(2^{n^2} + s(n^2) + s'(n))$.*

Proof (of Lemma 3.4)
Fix a function α satisfying the hypotheses of the lemma, and let $\beta = \sqrt{\alpha}$. Let $M_1, M_2, \ldots$ be a standard enumeration of autoreductions clocked such that M_i runs in time $n^{\beta(i)}$ on inputs of

length n. Our construction starts out with A being the empty set, and then adds strings to A in subsequent stages $i = 1, 2, 3, \ldots$ defined by the following sequence:

$$\begin{cases} n_0 &= 0 \\ n_{i+1} &= n_i^{\beta(n_i)} + 1. \end{cases}$$

Note that since M_i runs in time $n^{\beta(i)}$, M_i cannot query strings of length n_{i+1} or more on input 0^{n_i}.

Fix an integer $i \geqslant 1$ and let $m = n_i$. For any integer j such that $0 \leqslant j \leqslant \log \beta(m)$, let I_j denote the set of all strings with lengths in the interval $[m^{2^j}, \min(m^{2^{j+1}}, m^{\beta(m)} + 1))$. Note that $\{I_j\}_{j=0}^{\log \beta(m)}$ forms a partition of the set I of strings with lengths in $[m, m^{\beta(m)} + 1) = [n_i, n_{i+1})$ with the property that for any $0 \leqslant k \leqslant \log \beta(m)$, the length of any string in $\cup_{j=0}^{k} I_j$ is no more than the square of the length of any string in I_k.

During the i-th stage of the construction, we will encode the restriction $K|_I$ of K to I into $\{\langle b, y\rangle \mid b \in \{0,1\} \text{ and } y \in I\}$, and use the string 0^m for diagonalizing against M_i, applying the next strengthening of Statement 3.1 to do so:

Claim 3.1 *For any set A, at least one of the following holds:*

$$\begin{aligned} &(\forall\, \ell_y)_{y \in I_0}, (\exists\, r_y)_{y \in I_0}, (\forall\, \ell_y)_{y \in I_1}, (\exists\, r_y)_{y \in I_1}, \\ &\quad \ldots, (\forall\, \ell_y)_{y \in I_{\log \beta(m)}}, (\exists\, r_y)_{y \in I_{\log \beta(m)}} : M_i^{A'}(0^m) \text{ accepts} \end{aligned} \tag{2}$$

or

$$\begin{aligned} &(\forall\, r_y)_{y \in I_0}, (\exists\, \ell_y)_{y \in I_0}, (\forall\, r_y)_{y \in I_1}, (\exists\, \ell_y)_{y \in I_1}, \\ &\quad \ldots, (\forall\, r_y)_{y \in I_{\log \beta(m)}}, (\exists\, \ell_y)_{y \in I_{\log \beta(m)}} : M_i^{A'}(0^m) \text{ rejects}, \end{aligned} \tag{3}$$

where A' denotes $A \cup \{\langle 0, y\rangle \mid y \in I \text{ and } \ell_y = 1\} \cup \{\langle 1, y\rangle \mid y \in I \text{ and } r_y = 1\}$.

Here we use $(Q\, z_y)_{y \in Y}$ as a shorthand for $Q\, z_{y_1}, Q\, z_{y_2}, \ldots, Q\, z_{y_{|Y|}}$, where $Y = \{y_1, y_2, \ldots, y_{|Y|}\}$ and all variables are quantified over $\{0,1\}$. Without loss of generality we assume that the range of the pairing function $\langle \cdot, \cdot \rangle$ is disjoint from 0^*.

Proof (of Claim 3.1)
Fix A. If (2) does not hold, then its negation holds, i.e,

$$\begin{aligned} &(\exists\, \ell_y)_{y \in I_0}, (\forall\, r_y)_{y \in I_0}, (\exists\, \ell_y)_{y \in I_1}, (\forall\, r_y)_{y \in I_1}, \\ &\quad \ldots, (\exists\, \ell_y)_{y \in I_{\log \beta(m)}}, (\forall\, r_y)_{y \in I_{\log \beta(m)}} : M_i^{A'}(0^m) \text{ rejects}. \end{aligned} \tag{4}$$

Switching the quantifiers $(\exists\, \ell_y)_{y \in I_j}$ and $(\forall\, r_y)_{y \in I_j}$ pairwise for every $0 \leqslant j \leqslant \log \beta(m)$ in (4) yields the weaker statement (3).

(Claim 3.1) □

Figure 1 describes the i-th stage in the construction of the set A. Note that the lexicographically first values in this algorithm always exist, so the construction works fine. We now argue that the resulting set A satisfies the properties of Lemma 3.4.

1. The construction guarantees that by the end of stage i, $A(0^m) = 1 - M_i^{A \setminus \{0^m\}}(0^m)$ holds. Since M_i on input 0^m cannot query 0^m (because M_i is an autoreduction) nor any of the strings added during subsequent stages (because M_i does not even have the time to write down any of these strings), $A(0^m) = 1 - M_i^{A}(0^m)$ holds for the final set A. So, M_i is not an autoreduction of A. Since this is true of any autoreduction M_i, the set A is not autoreducible.

if formula (2) holds

  then for $j = 0, \ldots, \log \beta(m)$

    $(\ell_y)_{y \in I_j} \leftarrow (K(y))_{y \in I_j}$

    $(r_y)_{y \in I_j} \leftarrow$ the lexicographically first value satisfying

      $(\forall\, \ell_y)_{y \in I_j+1}, (\exists\, r_y)_{y \in I_{j+1}}, (\forall\, \ell_y)_{y \in I_{j+2}}, (\exists\, r_y)_{y \in I_{j+2}},$

      $\ldots, (\forall\, \ell_y)_{y \in I_{\log \beta(m)}}, (\exists\, r_y)_{y \in I_{\log \beta(m)}} : M_i^{A'}(0^m)$ accepts,

    where $A' = A \cup \{\langle 0, y\rangle \mid y \in I \text{ and } \ell_y = 1\} \cup \{\langle 1, y\rangle \mid y \in I \text{ and } r_y = 1\}$

    endfor

   $A \leftarrow A \cup \{\langle 0, y\rangle \mid y \in I \text{ and } \ell_y = 1\} \cup \{\langle 1, y\rangle \mid y \in I \text{ and } r_y = 1\}$

  else { formula (3) holds }

   for $j = 0, \ldots, \log \beta(m)$

    $(r_y)_{y \in I_j} \leftarrow (K(y))_{y \in I_j}$

    $(\ell_y)_{y \in I_j} \leftarrow$ the lexicographically first value satisfying

      $(\forall\, r_y)_{y \in I_{j+1}}, (\exists\, \ell_y)_{y \in I_{j+1}}, (\forall\, r_y)_{y \in I_{j+2}}, (\exists\, \ell_y)_{y \in I_{j+2}},$

      $\ldots, (\forall\, r_y)_{y \in I_{\log \beta(m)}}, (\exists\, \ell_y)_{y \in I_{\log \beta(m)}} : M_i^{A'}(0^m)$ accepts,

    where $A' = A \cup \{\langle 0, y\rangle \mid y \in I \text{ and } \ell_y = 1\} \cup \{\langle 1, y\rangle \mid y \in I \text{ and } r_y = 1\}$

    endfor

   $A \leftarrow A \cup \{0^m\} \cup \{\langle 0, y\rangle \mid y \in I \text{ and } \ell_y = 1\} \cup \{\langle 1, y\rangle \mid y \in I \text{ and } r_y = 1\}$

  endif

Figure 1: Stage i of the construction of the set A in Lemma 3.4

2. During stage i, we encode $K|_I$ in the left region iff we do not put 0^m into A; otherwise we encode $K|_I$ in the right region. So, for any $y \in I$, $K(y) = A(\langle A(0^m), y\rangle)$. Therefore, $K \leqslant^{\mathrm{P}}_{2-\mathrm{T}} A$.

3. First note that A only contains strings of the form 0^m with $m = n_i$ for some integer $i \geqslant 1$, and strings of the form $\langle b, y\rangle$ with $b \in \{0,1\}$ and $y \in \Sigma^*$.

   Assume we have executed the construction of A up to but not including stage i and stored the result in memory. The additional work to decide the membership to A of a string belonging to the i-th stage, is as follows.

   case 0^m
   Since $0^m \in A$ iff formula (2) does not hold and (2) is a $\mathrm{QBF}_{2\log\beta(m)}$-formula of size $2^{O(m^{\beta(m)})} \leqslant 2^{m^{\alpha(m)}}$, we can decide whether $0^m \in A$ in time $O(t'(m))$ and space $O(s'(m))$.

   case $\langle b, z\rangle$ where $b = A(0^m)$ and $z \in I$
   Then $\langle b, z\rangle \in A$ iff $z \in K$, which we can decide in time $t(|z|)$ and space $s(|z|)$.

   case $\langle b, z\rangle$ where $b = 1 - A(0^m)$ and $z \in I$
   Say $z \in I_k$, $0 \leqslant k \leqslant \log\beta(m)$. In order to compute whether $\langle b, z\rangle \in A$, we run the part of stage i corresponding to the values of j in Figure 1 up to and including k, and store the results in memory. This involves computing K on $\cup_{j=0}^{k} I_j$ and deciding $O(2^{|z|})$ $\mathrm{QBF}_{2\log\beta(m)}$-formulae of size $2^{O(m^{\beta(m)})} \leqslant 2^{m^{\alpha(m)}}$, namely one formula for each $y \in \cup_{j=0}^{k} I_j$ which precedes or equals z in lexicographic order. The latter takes $O(2^{|z|} \cdot t'(m))$ time and $O(2^{|z|} + s'(m))$ space. Since every string in $\cup_{j=0}^{k} I_j$ is of size no more than $|z|^2$, we can do the former in time $O(2^{|z|^2} \cdot t(|z|^2))$ and space $O(2^{|z|^2} + s(|z|^2))$. So, the requirements for this stage are $O(2^{|z|^2} \cdot t(|z|^2) + 2^{|z|} \cdot t'(|z|))$ time and $O(2^{|z|^2} + s(|z|^2) + s'(|z|))$ space.

   A similar analysis also shows that we can perform the stages up to but not including i in time $O(2^m \cdot (t(m) + t'(m)))$ and space $O(2^m + s(m) + s'(m))$.

   All together, this yields the time and space bounds claimed for A.

(Lemma 3.4) □

Using the upper bound $2^{n^{\alpha(n)}}$ for $s'(n)$, the smallest standard complexity class to which Lemma 3.4 applies, seems to be EEXPSPACE. This results in Theorem 3.1.

Proof (of Theorem 3.1)
In Lemma 3.4, set K a $\leqslant^{\mathrm{P}}_{\mathrm{m}}$-complete set for EEXPSPACE, and $\alpha(n) = n$. □

In section 4.2, we will see that $\leqslant^{\mathrm{P}}_{2-\mathrm{T}}$ in the statement of Theorem 3.1 is optimal: Theorem 4.5 shows that Theorem 3.1 fails for $\leqslant^{\mathrm{P}}_{2-\mathrm{tt}}$.

We note that the proof of Theorem 3.1 carries through for $\leqslant^{\mathrm{EXPSPACE}}_{\mathrm{T}}$-reductions with polynomially bounded query lengths. This implies the strengthening given by Theorem 3.2.

3.2 Nonadaptive Autoreductions

Diagonalizing against nonadaptive autoreductions M is easier. If M runs in time $\tau(n)$, there can be no more than $\tau(n)$ coding strings that interfere with the diagonalization, as opposed to $2^{\tau(n)}$ in the adaptive case. This allows us to reduce the complexity of the set constructed in Lemma 3.4 as follows.

Lemma 3.5 *Fix a set K, and suppose we can decide it simultaneously in time $t(n)$ and space $s(n)$. Let $\alpha : \mathbb{N} \to (0, \infty)$ be a constructible monotone unbounded function, and suppose there is a deterministic Turing machine accepting TQBF that takes time $t'(n)$ and space $s'(n)$ on QBF-formulae of size $n^{\alpha(n)}$ with at most $\log \alpha(n)$ alternations. Then there is a set A such that:*

1. *A is not nonadaptively autoreducible.*
2. *$K \leqslant^{\mathrm{P}}_{3-\mathrm{tt}} A$.*
3. *We can decide A simultaneously in time $O((2^n + n^{\alpha(n)}) \cdot (t(n^2) + t'(n)))$ and space $O(2^n + n^{\alpha(n)} + s(n^2) + s'(n))$.*

Proof (of Lemma 3.5)
The construction of the set A is the same as in Lemma 3.4 (see Figure 1) apart from the following differences:

- $M_1, M_2, \ldots$ now is a standard enumeration of nonadaptive autoreductions clocked such that M_i runs in time $n^{\beta(i)}$ on inputs of length n. Note that the set $Q_M(x)$ of possible queries M makes on input x contains no more than $|x|^{\beta(i)}$ elements.
- During stage $i \geqslant 1$ of the construction, I denotes the set of all strings y with lengths in $[m, m^{\beta(m)} + 1) = [n_i, n_{i+1})$ such that $\langle 0, y\rangle \in Q_{M_i}(0^m)$ or $\langle 1, y\rangle \in Q_{M_i}(0^m)$, and I_j for $0 \leqslant j \leqslant \log \beta(m)$ denotes the set of strings in I with lengths in $[m^{2^j}, \min(m^{2^{j+1}}, m^{\beta(m)} + 1))$. Note that the only ℓ_y's and r_y's that affect the validity of the predicate "$M_i^{A'}(0^m)$ accepts" in formula (2) and the corresponding formulae in Figure 1, are those for which $y \in I$.
- At the end of stage i in Figure 1, we add the following line:

  $$A \leftarrow A \cup \{\langle b, y\rangle \mid b \in \{0,1\}, y \in \Sigma^* \text{ with } m \leqslant |y| < m^{\beta(m)} + 1, y \notin I \text{ and } K(y) = 1\}.$$

  This ensures coding $K(y)$ for strings y with lengths in $[n_i, n_{i+1})$ such that $\langle 0, y\rangle$ nor $\langle 1, y\rangle$ are queried by M_i on input 0^m. Although not essential, we choose to encode them in both the left and the right region.

The proof that A satisfies the 3 properties claimed, carries over. Only the time and space analysis in the third point needs modification. The crucial simplification over the adaptive case lies in the fact that (2) and the similar formulae in Figure 1 now become $\mathrm{QBF}_{2\log\beta(n)}$-formulae of size $n^{O(\beta(m))}$ as opposed to of size $2^{O(m^{\beta(m)})}$ in Lemma 3.4. More specifically, referring to the proof of Lemma 3.4, we have the following cases regarding the work at stage i of the construction.

case 0^m
The above mentioned simplification takes care of this case.

case $\langle b, z\rangle$ where $b = A(0^m)$ and $z \in I$
The argument of Lemma 3.4 carries over as such.

case $\langle b, z\rangle$ where $b = 1 - A(0^m)$ and $z \in I$
Computing K on $\cup_{j=0}^{k} I_j$ and storing the result can be done in time $O(m^{\beta(m)} \cdot t(|z|^2))$ and space $O(m^{\beta(m)} + s(|z|^2))$. Deciding the $O(m^{\beta(m)})$ $\mathrm{QBF}_{2\log\beta(m)}$-formulae of size $m^{O(\beta(m))} \leqslant m^{\alpha(m)}$ involved requires no more than $O(m^{\beta(m)} \cdot t'(m))$ time and $O(m^{\beta(m)} + s'(m))$ space.

case $\langle b, z\rangle$ where $b \in \{0,1\}$, $m \leqslant |z| \leqslant m^{\beta(m)} + 1$, and $z \notin I$.

This is an additional case. By construction, $\langle b, z\rangle \in A$ iff $z \in K$, which we can decide in time $t(|z|)$ and space $s(|z|)$.

By similar analysis, a rough estimate of the resources required for the previous stages of the construction is $O(2^m \cdot (t(m) + t'(m)))$ time and $O(2^m + s(m) + s'(m))$ space, resulting in a total as stated in the lemma. (Lemma 3.5) □

As a consequence, we can lower the space complexity in the equivalent of Theorem 3.1 from doubly exponential to singly exponential, yielding Theorem 3.3. In section 4.2, we will show we cannot reduce the number of queries from 3 to 2 in Theorem 3.3.

If we restrict the number of queries the nonadaptive autoreduction is allowed to make to some fixed polynomial, the proof technique of Theorem 3.3 also applies to EXP. In particular, we obtain:

Theorem 3.6 *There is a $\leqslant^{\mathrm{P}}_{3-\mathrm{tt}}$-complete set for* EXP *that is not $\leqslant^{\mathrm{P}}_{\mathrm{btt}}$-autoreducible.*

4 Autoreducibility Results

For small complexity classes, all complete sets turn out to be autoreducible. Beigel and Feigenbaum [5] established this property of all levels of the polynomial-time hierarchy as well as of PSPACE, the largest class for which it was known to hold before our work. In this section, we will prove it for the Δ-levels of the exponential-time hierarchy.

As to nonadaptive reductions, the question was even open for all levels of the polynomial-time hierarchy. We will show here that the $\leqslant^{\mathrm{P}}_{\mathrm{tt}}$-complete sets for the Δ-levels of the polynomial-time hierarchy are nonadaptively autoreducible. For any complexity class containing EXP, we will prove that the $\leqslant^{\mathrm{P}}_{2-\mathrm{tt}}$-complete sets are $\leqslant^{\mathrm{P}}_{2-\mathrm{tt}}$-autoreducible.

Finally, we will also consider nonuniform and randomized autoreductions.

Throughout this section, we will assume without loss of generality an encoding γ of a computation of a given oracle Turing machine M on a given input x with the following properties. γ will be a marked concatenation of successive instantaneous descriptions of M, starting with the initial instantaneous description of M on input x, such that:

- Given a pointer to a bit in γ, we can find out whether that bit represents the answer to an oracle query by probing a constant number of bits of γ.
- If it is the answer to an oracle query, the corresponding query is a substring of the prefix of γ up to that point, and we can easily compute a pointer to the beginning of that substring without probing γ any further.
- If it is not the answer to an oracle query, we can perform a local consistency check for that bit which only depends on a constant number of previous bit positions of γ and the input x. Formally, there exist a function g_M and a predicate e_M, both polynomial-time computable, and a constant c_M such that the following holds: For any input x, any index i to a bit position in γ, and any j, $1 \leqslant j \leqslant c_M$, $g_M(x; i, j)$ is an index no larger than i, and

$$e_M(x; i, \gamma_{g_M(x;i,1)}, \gamma_{g_M(x;i,2)}, \ldots, \gamma_{g_M(x;i;c_M)}) \tag{5}$$

  indicates whether γ passes the local consistency test for its i-th bit γ_i. Provided the prefix of γ up to but not including position i is correct, the local consistency test is passed iff γ_i is correct.

We call such an encoding a *valid computation* of M on input x iff the local consistency tests (5) for all the bit positions i that do not correspond to oracle answers, are passed, and the other bits equal the oracle's answer to the corresponding query. Any other string we will call a *computation*.

4.1 Adaptive Autoreductions

We will first show that every $\leqslant_{\mathrm{T}}^{\mathrm{P}}$-complete set for EXP is autoreducible, and then generalize to all Δ-levels of the exponential-time hierarchy.

Theorem 4.1 *Every $\leqslant_{\mathrm{T}}^{\mathrm{P}}$-complete set for* EXP *is autoreducible.*

Here is the proof idea: For any of the standard deterministic complexity classes $\mathcal{C}$, we can decide each bit of the computation on a given input x within $\mathcal{C}$. So, if A is a $\leqslant_{\mathrm{T}}^{\mathrm{P}}$-complete set for $\mathcal{C}$ that can be decided by a machine M within the confines of the class $\mathcal{C}$, then we can $\leqslant_{\mathrm{T}}^{\mathrm{P}}$-reduce deciding the i-th bit of the computation of M on input x to A. Now, consider the two (possibly invalid) computations we obtain by applying for every bit position the above reduction, answering all queries except for x according to A, and assuming $x \in A$ for one computation, and $x \notin A$ for the other.

Note that the computation corresponding to the right assumption about $A(x)$, is certainly correct. So, if both computations yield the same answer (which we can efficiently check using A without querying x), that answer is correct. If not, the other computation contains a mistake. We cannot check both computations entirely to see which one is right, but given a pointer to the first incorrect bit of the wrong computation, we can efficiently verify that it is mistaken by checking only a constant number of bits of that computation. The pointer is again computable within $\mathcal{C}$.

In case $\mathcal{C} \subseteq$ EXP, using a $\leqslant_{\mathrm{T}}^{\mathrm{P}}$-reduction to A and assuming $x \in A$ or $x \notin A$ as above, we can determine the pointer with oracle A (but without querying x) in polynomial time, since the pointer's length is polynomially bounded.

We now fill out the details.

Proof (of Theorem 4.1)
Fix a $\leqslant_{\mathrm{T}}^{\mathrm{P}}$-complete set A for EXP. Say A is accepted by a Turing machine M such that the computation of M on an input of size n has length $2^{p(n)}$ for some fixed polynomial p. Without loss of generality the last bit of the computation gives the final answer. Let g_M, e_M, and c_M be the formalization of the local consistency test for M as described by (5).

Let $\mu(\langle x, i\rangle)$ denote the i-th bit of the computation of M on input x. We can compute μ in EXP, so there is an oracle Turing machine R_μ $\leqslant_{\mathrm{T}}^{\mathrm{P}}$-reducing μ to A.

Let $\sigma(x)$ be the first i, $1 \leqslant i \leqslant 2^{p(|x|)}$, such that $R_\mu^{A\setminus\{x\}}(\langle x, i\rangle) \neq R_\mu^{A\cup\{x\}}(\langle x, i\rangle)$, provided it exists. Again, we can compute σ in EXP, so there is a $\leqslant_{\mathrm{T}}^{\mathrm{P}}$-reduction R_σ from σ to A.

Consider the algorithm in Figure 2 for deciding A on input x. The algorithm is a polynomial-time oracle Turing machine with oracle A that does not query its own input x. We now argue that it correctly decides A on input x. We distinguish between two cases:

case $R_\mu^{A\setminus\{x\}}(\langle x, 2^{p(|x|)}\rangle) = R_\mu^{A\cup\{x\}}(\langle x, 2^{p(|x|)}\rangle)$

Since at least one of the computations $R_\mu^{A\setminus\{x\}}(\langle x, \cdot\rangle)$ or $R_\mu^{A\cup\{x\}}(\langle x, \cdot\rangle)$ coincides with the actual computation of M on input x, and the last bit of the computation equals the final decision, correctness follows.

case $R_\mu^{A\setminus\{x\}}(\langle x, 2^{p(|x|)}\rangle) \neq R_\mu^{A\cup\{x\}}(\langle x, 2^{p(|x|)}\rangle)$

If $x \in A$, $R_\mu^{A\setminus\{x\}}(\langle x, 2^{p(|x|)}\rangle) = 0$, so $R_\mu^{A\setminus\{x\}}(\langle x, \cdot\rangle)$ contains a mistake. Variable i gets the

$$
\begin{array}{l}
\textbf{if } R_\mu^{A\setminus\{x\}}(\langle x, 2^{p(|x|)}\rangle) = R_\mu^{A\cup\{x\}}(\langle x, 2^{p(|x|)}\rangle) \\
\quad \textbf{then accept} \text{ iff } R_\mu^{A\cup\{x\}}(\langle x, 2^{p(|x|)}\rangle) = 1 \\
\quad \textbf{else } i \leftarrow R_\sigma^{A\cup\{x\}}(x) \\
\qquad\quad \textbf{accept} \text{ iff } e_M(x; i, R_\mu^{A\setminus\{x\}}(\langle x, g_M(x;i;1)\rangle), R_\mu^{A\setminus\{x\}}(\langle x, g_M(x;i,2)\rangle), \ldots, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \ldots, R_\mu^{A\setminus\{x\}}(\langle x, g_M(x;i,c_M)\rangle)) = 0 \\
\quad \textbf{endif}
\end{array}
$$

Figure 2: Autoreduction for the set A of Theorem 4.1 on input x

correct value of the index of the first incorrect bit in this computation, so the local consistency test for $R_\mu^{A\setminus\{x\}}(\langle x, \cdot\rangle)$ being the computation of M on input x fails on the i-th bit, and we accept x.

If $x \notin A$, $R_\mu^{A\setminus\{x\}}(\langle x, \cdot\rangle)$ is a valid computation, so no local consistency test fails, and we reject x.

(Theorem 4.1) □

The local checkability property of computations used in the proof of Theorem 4.1 does not relativize, because the oracle computation steps depend on the entire query, i.e., on a number of bits that is only limited by the resource bounds of the base machine, in this case exponentially many. We next show that Theorem 4.1 itself also does not relativize.

Theorem 4.2 *Relative to some oracle,* EXP *has a* $\leqslant^{\mathrm{P}}_{2-\mathrm{T}}$*-complete set that is not autoreducible.*

Proof
Note that EXP has the following property:

Property 4.1 *There is an oracle Turing machine N running in* EXP *such that for any oracle B, the set accepted by N^B is* $\leqslant^{\mathrm{P}}_{\mathrm{m}}$*-complete for* EXP^B.

Without loss of generality, we assume that N runs in time 2^n. Let K^B denote the set accepted by N^B.

We will construct an oracle B and a set A such that A is $\leqslant^{\mathrm{P}}_{2-\mathrm{T}}$-complete for EXP^B and is not $\leqslant^{\mathrm{P}^B}_{\mathrm{T}}$-autoreducible.

The construction of A is the same as in Lemma 3.4 (see Figure 1) with $\beta(n) = \log n$ and $K = K^B$, except for that the reductions M_i now also have access to the oracle B.

We will encode in B information about the construction of A that reduces the complexity of A relative to B, but do it high enough so as not to destroy the $\leqslant^{\mathrm{P}}_{2-\mathrm{T}}$-completeness of A for EXP^B nor the diagonalizations against $\leqslant^{\mathrm{P}^B}_{\mathrm{T}}$-autoreductions.

We construct B in stages along with A. We start with B empty. Using the notation of Lemma 3.4, at the beginning of stage i, we add 0^{2^m} to B iff property (2) does not hold, and at the end of sub-stage j, we union B with

$$
\begin{array}{ll}
\{\langle 0^{2^{m^{2^{j+1}}}}, y\rangle \mid y \in I_j \text{ and } r(y) = 1\} & \text{if (2) holds at stage } i \\
\{\langle 0^{2^{m^{2^{j+1}}}}, y\rangle \mid y \in I_j \text{ and } \ell(y) = 1\} & \text{otherwise.}
\end{array}
$$

Note that this does not affect the value of $K^B(y)$ for $|y| < m^{2^{j+1}}$, nor the computations of M_i on inputs of size at most m (for sufficiently large i such that $m^{\log m} < 2^m$). It follows from the analysis in the proof of Lemma 3.4 that the set A is $\leqslant^{\mathrm{P}}_{2-\mathrm{T}}$-hard for EXP^B and not $\leqslant^{\mathrm{P}^B}_{\mathrm{T}}$-autoreducible.

Regarding the complexity of deciding A relative to B, note that the encoding in the oracle B allows us to eliminate the need for evaluating $\mathrm{QBF}_{\log \beta(n)}$-formulae of size $2^{n^{\beta(n)}}$. Instead, we just query B on easily constructed inputs of size $O(2^{n^2})$. Therefore, we can drop the terms corresponding to the $\mathrm{QBF}_{\log \beta(n)}$-formulae of size $2^{n^{\beta(n)}}$ in the complexity of A. Consequently, $A \in \mathrm{EXP}^B$.

(Theorem 4.2) □

Theorem 4.2 applies to any complexity class containing EXP that has Property 4.1, e.g., EXPSPACE, EEXP, EEXPSPACE, etc.

Sometimes, the structure of the oracle allows us to get around the lack of local checkability of oracle queries. This is the case for oracles from the polynomial-time hierarchy, and leads to the following extension of Theorem 4.1:

Theorem 4.3 *For any integer $k \geqslant 0$, every $\leqslant^{\mathrm{P}}_{\mathrm{T}}$-complete set for $\Delta^{\mathrm{EXP}}_{k+1}$ is autoreducible.*

The proof idea is as follows: Let A be a $\leqslant^{\mathrm{P}}_{\mathrm{T}}$-complete set accepted by the deterministic oracle Turing machine M with oracle TQBF_k. First note that there is a polynomial-time Turing machine N such that a query q belongs to the oracle TQBF_k iff

$$\exists\, y_1,\, \forall\, y_2,\, \ldots,\, Q_k\, y_k : N(q, y_1, y_2, \ldots, y_k) \text{ accepts}, \tag{6}$$

where the y_ℓ's are of size polynomial in $|q|$.

We consider the two purported computations of M on input x constructed in the proof of Theorem 4.1. One of them belongs to a party assuming $x \in A$, the other one to a party assuming $x \notin A$. The computation corresponding to the right assumption is correct; the other one might not be.

Now, suppose the computations differ, and we are given a pointer to the first bit position where they disagree, which turns out to be the answer to an oracle query q. Then we can have the two parties play the k-round game underlying (6): The party claiming $q \in \mathrm{TQBF}_k$ plays the existentially quantified y_ℓ's, the other one the universally quantified y_ℓ's. The players' strategies will consist of computing the game history so far, determining their optimal next move, $\leqslant^{\mathrm{P}}_{\mathrm{T}}$-reducing this computation to A, and finally producing the result of this reduction under their respective assumption about $A(x)$. This will guarantee that the party with the correct assumption plays optimally. Since this is also the one claiming the correct answer to the oracle query q, he will win the game, i.e., $N(q, y_1, y_2, \ldots, y_k)$ will equal his answer bit.

The only thing the autoreduction for A has to do, is determine the value of $N(q, y_1, y_2, \ldots, y_k)$ in polynomial time using A as an oracle but without querying x. It can do that along the lines of the base case algorithm given in Figure 2. If during this process, the local consistency test for N's computation requires the knowledge of bits from the y_ℓ's, we compute these via the reduction defining the strategy of the corresponding player. The bits from q we need, we can retrieve from the M-computations, since both computations are correct up to the point where they finished generating q. Once we know $N(q, y_1, y_2, \ldots, y_k)$, we can easily decide the correct assumption about $A(x)$.

The construction hinges on the hypothesis that we can $\leqslant^{\mathrm{P}}_{\mathrm{T}}$-reduce determining the player's moves to A. Computing these moves can become quite complex, though, because we have to recursively reconstruct the game history so far. The number of rounds k being constant, seems crucial for keeping the complexity under control. The conference version of this paper [6] erroneously claimed the proof works for EXPSPACE, which can be thought of as alternating exponential time with an exponential number of alternations. Establishing Theorem 4.3 for EXPSPACE would actually separate NL from NP, as we will see in Section 5.

Proof (of Theorem 4.3)
Let A be a $\leqslant_{\mathrm{T}}^{\mathrm{P}}$-complete set for $\Delta_{k+1}^{\mathrm{EXP}} = \mathrm{EXP}^{\Sigma_k^{\mathrm{P}}}$ accepted by the exponential-time oracle Turing machine M with oracle TQBF_k. Let g_M, e_M, and c_M be the formalization of the local consistency test for M as described by (5). Without loss of generality there is a polynomial p and a polynomial-time Turing machine N such that on inputs of size n, M makes exactly $2^{p(n)}$ oracle queries, all of the form

$$\exists\, y_1 \in \Sigma^{2^{p(n)}}, \forall\, y_2 \in \Sigma^{2^{p(n)}}, \ldots, Q_k\, y_k \in \Sigma^{2^{p(n)}} : N(q, y_1, y_2, \ldots, y_k) \text{ accepts}, \tag{7}$$

where q has length $2^{p^2(n)}$. Moreover, the computations of N in (7) each have length $2^{p^3(n)}$, and their last bit represents the answer; the same holds for the computations of M on inputs of length n. Let g_N, e_N, and c_N be the formalization of the local consistency test for N.

We first define a bunch of functions computable in $\Delta_{k+1}^{\mathrm{EXP}}$. For each of them, say ξ, we fix an oracle Turing machine R_ξ that $\leqslant_{\mathrm{T}}^{\mathrm{P}}$-reduces ξ to A, and which the final autoreduction for A will use. The proofs that we can compute these functions in $\Delta_{k+1}^{\mathrm{EXP}}$ are straightforward.

Let $\mu(\langle x, i\rangle)$ denote the i-th bit of the computation of M^{TQBF_k} on input x, and $\sigma(x)$ the first i (if any) such that $R_\mu^{A\setminus\{x\}}(\langle x, i\rangle) \neq R_\mu^{A\cup\{x\}}(\langle x, i\rangle)$. The roles of μ and σ are the same as in the proof of Theorem 4.1: We will use R_μ to figure out whether both possible answers for the oracle query "$x \in A$?" lead to the same final answer, and if not, use R_σ to find a pointer i to the first incorrect bit (in any) of the simulated computation getting the negative oracle answer $x \notin A$. If i turns out not to point to an oracle query, we can proceed as in the proof of Theorem 4.1. Otherwise, we will make use of the following functions and associated reductions to A.

We define the functions η_ℓ and y_ℓ inductively for $\ell = 1, \ldots, k$. At each level ℓ we first define η_ℓ, which induces a reduction R_{η_ℓ}, and then define y_ℓ based on R_{η_ℓ}. All of these functions take an input x such that the i-th bit of $R_\mu^{A\setminus\{x\}}(\langle x, \cdot\rangle)$ is the answer to an oracle query (7), where $i = R_\sigma^{A\cup\{x\}}(x)$. We define $\eta_\ell(x)$ as the lexicographically least $y_\ell \in \Sigma^{2^{p(|x|)}}$ such that

$$\begin{aligned}\chi[Q_{\ell+1}\, y_{\ell+1}, Q_{\ell+2}\, y_{\ell+2}, \ldots, Q_k\, y_k : N(q, y_1(x), y_2(x), \ldots, y_{\ell-1}(x), y_\ell, y_{\ell+1}, \ldots, y_k) \text{ accepts}]&\\ \equiv \ell \bmod 2;&\end{aligned} \tag{8}$$

if this value does not exist, we set $\eta_\ell(x) = 0^{2^{p(|x|)}}$. Note that the right-hand side of (8) is 1 iff y_ℓ is existentially quantified in (7).

$$y_\ell(x) = \begin{cases} R_{\eta_\ell}^{A\cup\{x\}}(x) & \text{if } \ell \equiv R_\mu^{A\cup\{x\}}(\langle x, i\rangle) \bmod 2 \\ R_{\eta_\ell}^{A\setminus\{x\}}(x) & \text{otherwise.} \end{cases} \tag{9}$$

The condition on the right-hand side of (9) means that we use the hypothesis $x \in A$ to compute $y_\ell(x)$ from R_{η_ℓ} in case:

- either y_ℓ is existentially quantified in (7) and the player assuming $x \in A$ claims (7) holds,
- or else y_ℓ is universally quantified and the player assuming $x \in A$ claims (7) fails.

Otherwise we use the hypothesis $x \notin A$.

In case i points to the answer to an oracle query (7), the functions η_ℓ and the reductions R_{η_ℓ} incorporate the moves during the successive rounds of the game underlying (7). The reduction

R_{η_ℓ} together with the player's assumption about membership of x to A, determines the actual move $y_\ell(x)$ during the ℓ-th round, namely $R_{\eta_\ell}^{A\cup\{x\}}(x)$ if the ℓ-th round is played by the opponent assuming $x \in A$, and $R_{\eta_\ell}^{A\setminus\{x\}}(x)$ otherwise. The condition on the right-hand side of (9) guarantees that the existentially quantified variables are determined by the opponent claiming the query (7) is a true formula, and the universally quantified ones by the other opponent. In particular, (9) ensures that the opponent with the correct claim about (7) has a wining strategy. Provided it exists, the function η_ℓ defines a winning move during the ℓ-th round of the game for the opponent playing that round, given the way the previous rounds were actually played (as described by the $y(x)$'s). For odd ℓ, i.e., y_ℓ is existentially quantified, it tries to set y_ℓ such that the remainder of (7) holds; otherwise it tries to set y_ℓ such that the remainder of (7) fails. The actual move may differ from the one given by η_ℓ in case the player's assumption about $x \in A$ is incorrect. The opponent with the correct assumption plays according to η_ℓ. Since that opponent also makes the correct claim about (7), he will win the game. In any case, $N(q, y_1, y_2, \ldots, y_k)$ will hold iff (7) holds.

Finally, we define the functions ν and τ, which have a similar job as the functions μ respectively σ, but for the computation of $N(q, y_1, y_2, \ldots, y_k)$ instead of the computation of $M_k^{\mathrm{TQBF}}(x)$. More precisely, $\nu(\langle x, r\rangle)$ equals the r-th bit of the computation of $N(q, y_1(x), y_2(x), \ldots, y_k(x))$, where the $y_\ell(x)$'s are defined by (9), and the bit with index $i = R_\sigma^{A\cup\{x\}}(x)$ in the computation $R_\mu^{A\setminus\{x\}}(\langle x, \cdot\rangle)$ is the answer to the oracle query (7). We define $\tau(x)$ to be the first r (if any) for which $R_\nu^{A\setminus\{x\}}(\langle x, r\rangle) \neq R_\nu^{A\cup\{x\}}(\langle x, r\rangle)$, provided the bit with index $i = R_\sigma^{A\cup\{x\}}(x)$ in the computation $R_\mu^{A\setminus\{x\}}(\langle x, \cdot\rangle)$ is the answer to an oracle query.

Now we have these functions and corresponding reductions, we can describe an autoreduction for A. On input x, it works as described in Figure 3. We next argue that the algorithm correctly decides A on input x. Checking the other properties required of an autoreduction for A is straightforward.

We only consider the cases where $R_\mu^{A\setminus\{x\}}(\langle x, 2^{p^3(|x|)}\rangle) \neq R_\mu^{A\cup\{x\}}(\langle x, 2^{p^3(|x|)}\rangle)$ and i points to the answer to an oracle query in $R_\mu^{A\setminus\{x\}}(\langle x, \cdot\rangle)$. We refer to the analysis in the proof of Theorem 4.1 for the remaining cases.

case $R_\nu^{A\setminus\{x\}}(\langle x, 2^{p^3(|x|)}\rangle) = R_\nu^{A\cup\{x\}}(\langle x, 2^{p^3(|x|)}\rangle)$

If $x \in A$, variable i points to the first incorrect bit of $R_\mu^{A\setminus\{x\}}(\langle x, \cdot\rangle)$, which turns out to be the answer to an oracle query, say (7). Since $R_\nu^{A\cup\{x\}}(\langle x, 2^{p^3(|x|)}\rangle)$ yields the correct oracle answer to (7),

$$R_\mu^{A\setminus\{x\}}(\langle x, i\rangle) \neq R_\nu^{A\cup\{x\}}(\langle x, 2^{p^3(|x|)}\rangle) = R_\nu^{A\setminus\{x\}}(\langle x, 2^{p^3(|x|)}\rangle),$$

and we accept x.

If $x \notin A$, both $R_\mu^{A\setminus\{x\}}(\langle x, i\rangle)$ and $R_\nu^{A\setminus\{x\}}(\langle x, 2^{p^3(|x|)}\rangle)$ give the correct answer to the oracle query i points to in the computation $R_\mu^{A\setminus\{x\}}(\langle x, \cdot\rangle)$. So, they are equal, and we reject x.

case $R_\nu^{A\setminus\{x\}}(\langle x, 2^{p^3(|x|)}\rangle) \neq R_\nu^{A\cup\{x\}}(\langle x, 2^{p^3(|x|)}\rangle)$

Then, as described in Figure 3, we will use the local consistency test for $R_\nu^{A\setminus\{x\}}(\langle x, \cdot\rangle)$ being the computation of $N(q, y_1(x), y_2(x), \cdots, y_k(x))$. Apart from bits in the purported computation $R_\nu^{A\setminus\{x\}}(\langle x, \cdot\rangle)$, this test may also need bits from q and from the $y_\ell(x)$'s. The $y_\ell(x)$'s can be computed straightforwardly using their definition (9). The bits from q we might need, can be retrieved from $R_\nu^{A\setminus\{x\}}(\langle x, \cdot\rangle)$. This is because our encoding scheme for computations has the property that the query q is a substring of the prefix of the computation up to the position indexed by i. Since either $R_\nu^{A\setminus\{x\}}(\langle x, \cdot\rangle)$ is correct everywhere, or else i is the first position

if $R_\mu^{A\setminus\{x\}}(\langle x, 2^{p^3(|x|)}\rangle) = R_\mu^{A\cup\{x\}}(\langle x, 2^{p^3(|x|)}\rangle)$

  then accept iff $R_\mu^{A\cup\{x\}}(\langle x, 2^{p^3(|x|)}\rangle) = 1$

  else $i \leftarrow R_\sigma^{A\cup\{x\}}(x)$

    if the i-th bit of $R_\mu^{A\setminus\{x\}}(\langle x, \cdot\rangle)$ is not the answer to an oracle query

      then accept iff $e_M(x; i, R_\mu^{A\setminus\{x\}}(\langle x, g_M(x;i,1)\rangle), R_\mu^{A\setminus\{x\}}(\langle x, g_M(x;i,2)\rangle), \ldots,$
      $\ldots, R_\mu^{A\setminus\{x\}}(\langle x, g_M(x;i,c_M)\rangle)) = 0$

      else if $R_\nu^{A\setminus\{x\}}(\langle x, 2^{p^3(|x|)}\rangle) = R_\nu^{A\cup\{x\}}(\langle x, 2^{p^3(|x|)}\rangle)$

        then accept iff $R_\mu^{A\setminus\{x\}}(\langle x, i\rangle) \neq R_\nu^{A\setminus\{x\}}(\langle x, 2^{p^3(|x|)}\rangle)$

        else $r \leftarrow R_\tau^{A\cup\{x\}}(x)$

          accept iff $e_N(q, y_1, y_2, \ldots, y_k; r, R_\nu^{A\setminus\{x\}}(\langle x, g_N(q, y_1, y_2, \ldots, y_k; r, 1)\rangle),$
          $R_\nu^{A\setminus\{x\}}(\langle x, g_N(q, y_1, y_2, \ldots, y_k; r, 2)\rangle), \ldots,$
          $\ldots, R_\nu^{A\setminus\{x\}}(\langle x, g_N(q, y_1, y_2, \ldots, y_k; r, c_N)\rangle)) = 0$

          where q denotes the query described in $R_\mu^{A\setminus\{x\}}(\langle x, \cdot\rangle)$
            to which the i-th bit in this computation is the answer

          and

$$y_\ell = \begin{cases} R_{\eta_\ell}^{A\cup\{x\}}(x) & \text{if } \ell \equiv R_\mu^{A\cup\{x\}}(\langle x, i\rangle) \bmod 2 \\ R_{\eta_\ell}^{A\setminus\{x\}}(x) & \text{otherwise} \end{cases}$$

        endif

    endif

endif

Figure 3: Autoreduction for the set A of Theorem 4.3 on input x

where it is incorrect, the description of q in $R_\nu^{A\setminus\{x\}}(\langle x, \cdot\rangle)$ is correct in any case. Moreover, we can easily compute a pointer to the beginning of the substring q of $R_\nu^{A\setminus\{x\}}(\langle x, \cdot\rangle)$ from i.

If $x \in A$, $R_\nu^{A\setminus\{x\}}(\langle x, 2^{p^3(|x|)}\rangle)$ is incorrect, so $R_\nu^{A\setminus\{x\}}(\langle x, \cdot\rangle)$ has an error as a computation of $N(q, y_1(x), y_2(x), \ldots, y_k(x))$. Variable r gets assigned the index of the first incorrect bit in this computation, so the local consistency check fails, and we accept x.

If $x \notin A$, $R_\nu^{A\setminus\{x\}}(\langle x, \cdot\rangle)$ is a valid computation of $N(q, y_1(x), y_2(x), \ldots, y_k(x))$, so every local consistency test is passed, and we reject x.

(Theorem 4.3) □

4.2 Nonadaptive Autoreductions

So far, we constructed autoreductions for $\leqslant_{\mathrm{T}}^{\mathrm{P}}$-complete sets A. On input x, we looked at the two candidate computations obtained by reducing to A, answering all oracle queries except for x according to A, and answering query x positively for one candidate, and negatively for the other. If the candidates disagreed, we tried to find out the right one, which always existed. We managed to get the idea to work for quite powerful sets A, e.g., EXP-complete sets, by exploiting the local checkability of computations. That allowed us to figure out the wrong computation without going through the entire computation ourselves: With help from A, we first computed a pointer to the first mistake in the wrong computation, and then verified it locally.

We cannot use this adaptive approach for constructing nonadaptive autoreductions. It seems like figuring out the wrong computation in a nonadaptive way, requires the autoreduction to perform the computation of the base machine itself, so the base machine has to run in polynomial time. Then checking the computation essentially boils down to verifying the oracle answers. Using the game characterization of the polynomial-time hierarchy along the same lines as in Theorem 4.3, we can do this for oracles from the polynomial-time hierarchy.

Theorem 4.4 *For any integer $k \geqslant 0$, every $\leqslant_{\mathrm{tt}}^{\mathrm{P}}$-complete set for $\Delta_{k+1}^{\mathrm{P}}$ is nonadaptively autoreducible.*

Parallel to the adaptive case, an earlier version of this paper [6] stated Theorem 4.4 for unbounded k, i.e., for PSPACE. However, we only get the proof to work for constant k. In Section 5, we will see that proving Theorem 4.4 for PSPACE would separate NL from NP.

The only additional difficulty in the proof is that in the nonadaptive setting, we do not know which player has to perform the even rounds, and which one the odd rounds in the k-round game underlying a query like (6). But we can just have them play both scenarios, and afterwards figure out the relevant run.

Proof (of Theorem 4.4)
Let A be a $\leqslant_{\mathrm{tt}}^{\mathrm{P}}$-complete set for $\Delta_{k+1}^{\mathrm{P}} = \mathrm{P}^{\Sigma_k^{\mathrm{P}}}$ accepted by the polynomial-time oracle Turing machine M with oracle TQBF_k. Without loss of generality there is a polynomial p and a polynomial-time Turing machine N such that on inputs of size n, M makes exactly $p(n)$ oracle queries q, all of the form

$$\exists\, y_1 \in \Sigma^{p(n)}, \forall\, y_2 \in \Sigma^{p(n)}, \ldots, Q_k\, y_k \in \Sigma^{p(n)} : N(q, y_1, y_2, \ldots, y_k) \text{ accepts}, \tag{10}$$

where q has length $p^2(n)$. Let $q(x,i)$ denote the i-th oracle query of M^{TQBF_k} on input x. Note that $q \in \mathrm{FP}^{\Sigma_k^{\mathrm{P}}}$.

Let $Q = \{\langle x, i\rangle \mid q(x,i) \in \mathrm{TQBF}_k\}$. The set Q belongs to $\Delta_{k+1}^{\mathrm{P}}$, so there is a $\leqslant_{\mathrm{tt}}^{\mathrm{P}}$-reduction R_Q from Q to A.

If for a given input x, $R_Q^{A\cup\{x\}}$ and $R_Q^{A\setminus\{x\}}$ agree on $\langle x, j\rangle$ for every $1 \leqslant j \leqslant p(|x|)$, we are home: We can simulate the base machine M using $R_Q^{A\cup\{x\}}(\langle x, j\rangle)$ as the answer to the j-th oracle query.

Otherwise, we will make use of the following functions $\eta_1, \eta_2, \ldots, \eta_k$ computable in $\Delta_{k+1}^{\mathrm{P}}$, corresponding oracle Turing machines $R_{\eta_1}, R_{\eta_2}, \ldots, R_{\eta_k}$ defining $\leqslant_{\mathrm{tt}}^{\mathrm{P}}$-reductions to A, and functions $y_1, y_2, \ldots, y_k$ also computable in $\Delta_{k+1}^{\mathrm{P}}$. As in the proof of Theorem 4.3, we define η_ℓ and y_ℓ inductively for $\ell = 1, \ldots, k$. They are defined for inputs x such that there is a smallest $1 \leqslant i \leqslant p(|x|)$ for which $R_Q^{A\setminus\{x\}}(\langle x, i\rangle) \neq R_Q^{A\cup\{x\}}(\langle x, i\rangle)$. The value of $\eta_\ell(x)$ equals the lexicographically least $y_\ell \in \Sigma^{p(|x|)}$ such that

$$\chi[Q_{\ell+1}\, y_{\ell+1}, Q_{\ell+2}\, y_{\ell+2}, \ldots, Q_k\, y_k : N(q(x,i), y_1(x), y_2(x), \ldots, y_{\ell-1}(x), y_\ell, y_{\ell+1}, \ldots, y_k) \text{ accepts}] \equiv \ell \bmod 2; \tag{11}$$

we set $\eta_\ell(x) = 0^{p(|x|)}$ if such string does not exist. The right-hand side of (11) is 1 iff y_ℓ is existentially quantified in (10).

$$y_\ell = \begin{cases} R_{\eta_\ell}^{A\cup\{x\}}(x) & \text{if } \ell \equiv R_Q^{A\cup\{x\}}(\langle x, i\rangle) \bmod 2 \\ R_{\eta_\ell}^{A\setminus\{x\}}(x) & \text{otherwise.} \end{cases} \tag{12}$$

The condition on the right-hand side of (12) means that we use the hypothesis $x \in A$ to compute $y_\ell(x)$ from R_{η_ℓ} in case:

- either y_ℓ is existentially quantified in (10) and the assumption $x \in A$ leads to claiming that (10) holds,
- or else y_ℓ is universally quantified and the assumption $x \in A$ leads to claiming that (10) fails.

The intuitive meaning of the functions η_ℓ and the reductions R_{η_ℓ} is similar to in the proof of Theorem 4.3: They capture the moves during the ℓ-th round of the game underlying (10) for $q = q(x,i)$. The function η_ℓ encapsulates an optimal move during round ℓ if it exists, and the reduction R_{η_ℓ} under the player's assumption regarding membership of x to A, produces the actual move in that round. The condition on the right-hand side of (12) guarantees the correct alternation of rounds. We refer to the proof of Theorem 4.3 for more intuition.

Consider the algorithm in Figure 4. Note that the only queries to A the algorithm in Figure 4

if $R_Q^{A\setminus\{x\}}(\langle x,j\rangle) = R_Q^{A\cup\{x\}}(\langle x,j\rangle)$ for every $1 \leqslant j \leqslant p(|x|)$
  then accept iff M accepts x when the j-th oracle query is answered $R_Q^{A\cup\{x\}}(\langle x,j\rangle)$
  else $i \leftarrow$ first j such that $R_Q^{A\setminus\{x\}}(\langle x,j\rangle) \neq R_Q^{A\cup\{x\}}(\langle x,j\rangle)$
    accept iff $N(q, y_1, y_2, \ldots, y_k) = R_Q^{A\cup\{x\}}(\langle x,i\rangle)$
      where q denotes the i-th query of M on input x
        when the answer to the j-th oracle query is given by $R_Q^{A\cup\{x\}}(\langle x,j\rangle)$
      and
$$y_\ell = \begin{cases} R_{\eta_\ell}^{A\cup\{x\}}(x) & \text{if } \ell \equiv R_Q^{A\cup\{x\}}(\langle x,i\rangle) \bmod 2 \\ R_{\eta_\ell}^{A\setminus\{x\}}(x) & \text{otherwise} \end{cases}$$
  endif

Figure 4: Nonadaptive autoreduction for the set A of Theorem 4.4 on input x

needs to make, are the queries of R_Q different from x on inputs $\langle x,j\rangle$ for $1 \leqslant j \leqslant p(|x|)$, and the queries of R_{η_ℓ} different from x on input x for $1 \leqslant \ell \leqslant k$. Since R_Q and the R_{η_ℓ}'s are nonadaptive, it follows that Figure 4 describes a $\leqslant_{\mathrm{tt}}^{\mathrm{P}}$-reduction to A that does not query its own input. A similar but simplified argument as in the proof of Theorem 4.3 shows that it accepts A. So, A is nonadaptively autoreducible. (Theorem 4.4) □

Next, we consider more restricted reductions. Using a different technique, we show:

Theorem 4.5 *For any complexity class $\mathcal{C}$, every $\leqslant_{2-\mathrm{tt}}^{\mathrm{P}}$-complete set for $\mathcal{C}$ is $\leqslant_{2-\mathrm{tt}}^{\mathrm{P}}$-autoreducible, provided $\mathcal{C}$ is closed under exponential-time reductions that only ask one query which is smaller in length.*

In particular, Theorem 4.5 applies to $\mathcal{C}$ = EXP, EXPSPACE, and EEXPSPACE. In view of Theorems 3.1 and 3.3, this implies that Theorems 3.1, 3.3, and 4.5 are optimal.

The proof exploits the ability of EXP to simulate all polynomial-time reductions to construct an auxiliary set D within $\mathcal{C}$ such that any $\leqslant_{2-\mathrm{tt}}^{\mathrm{P}}$-reductions of D to some fixed complete set A has a property that induces an autoreduction on A.

Proof (of Theorem 4.5)
Let $M_1, M_2, \ldots$ be a standard enumeration of $\leqslant^{\mathrm{P}}_{2-\mathrm{tt}}$-reductions such that M_i runs in time n^i on inputs of size n. Let A be a $\leqslant^{\mathrm{P}}_{2-\mathrm{tt}}$-complete set for $\mathcal{C}$.

Consider the set D that only contains strings of the form $\langle 0^i, x\rangle$ for $i \in \mathbb{N}$ and $x \in \Sigma^*$, and is decided by the algorithm of Figure 5 on such an input. Except for deciding $A(x)$, the algorithm

```
case truth-table of M_i on input ⟨0^i, x⟩ with the truth-value of query x set to A(x)
   constant:
      accept iff M_i^A rejects ⟨0^i, x⟩
   of the form "y ∉ A":
      accept iff x ∉ A
   otherwise:
      accept iff x ∈ A
   endcase
```

Figure 5: Algorithm for the set D of Theorem 4.5 on input $\langle 0^i, x\rangle$.

runs in exponential time. Therefore, under the given conditions on $\mathcal{C}$, $D \in \mathcal{C}$, so there is a $\leqslant^{\mathrm{P}}_{2-\mathrm{tt}}$-reduction M_j from D to A.

The construction of D diagonalizes against every $\leqslant^{\mathrm{P}}_{2-\mathrm{tt}}$-reduction M_i of D to A whose truth-table on input $\langle 0^i, x\rangle$ would become constant once we filled in the membership bit for x. Therefore, for every input x, one of the following cases holds for the truth-table of M_j on input $\langle 0^j, x\rangle$.

- The reduced truth-table is of the form "$y \in A$" with $y \neq x$.
  Then $y \in A \Leftrightarrow M_j$ accepts $\langle 0^j, x\rangle \Leftrightarrow x \in A$.
- The reduced truth-table is of the form "$y \notin A$" with $y \neq x$.
  Then $y \notin A \Leftrightarrow M_j$ accepts $\langle 0^j, x\rangle \Leftrightarrow x \notin A$.
- The truth-table depends on the membership to A of 2 strings different from x.
  Then M_j^A does not query x on input $\langle 0^j, x\rangle$, and accepts iff $x \in A$.

The above analysis shows that the algorithm of Figure 6 describes a $\leqslant^{\mathrm{P}}_{2-\mathrm{tt}}$-reduction of A.

```
if |Q_{M_j}(⟨0^j, x⟩) \ {x}| = 2
   then accept iff M_j^A accepts ⟨0^j, x⟩
   else { |Q_{M_j}(⟨0^j, x⟩) \ {x}| = 1 }
         y ← unique element of Q_{M_j}(⟨0^j, x⟩) \ {x}
         accept iff y ∈ A
   endif
```

Figure 6: Autoreduction constructed in the proof of Theorem 4.5

(Theorem 4.5) □

4.3 Probabilistic and Nonuniform Autoreductions

The previous results in this section trivially imply that the $\leqslant_{\mathrm{T}}^{\mathrm{P}}$-complete sets for the Δ-levels of the exponential-time hierarchy are probabilistically autoreducible, and the $\leqslant_{\mathrm{tt}}^{\mathrm{P}}$-complete sets for the Δ-levels of the polynomial-time hierarchy are probabilistically nonadaptively autoreducible. Randomness allows us the prove more in the nonadaptive case.

First, we can establish Theorem 4.4 for EXP:

Theorem 4.6 *Let f be a constructible function. Every $\leqslant_{f(n)-\mathrm{tt}}^{\mathrm{P}}$-complete set for* EXP *is probabilistically $\leqslant_{O(f(n))-\mathrm{tt}}^{\mathrm{P}}$-autoreducible. In particular, every $\leqslant_{\mathrm{tt}}^{\mathrm{P}}$-complete set for* EXP *is probabilistically nonadaptively autoreducible.*

Proof (of Theorem 4.6)
Let A be a $\leqslant_{f(n)-\mathrm{tt}}^{\mathrm{P}}$-complete set for EXP. We will apply the PCP Theorem for EXP [2] to A.

Lemma 4.7 ([2]) *There is a constant k such that for any set $A \in \mathrm{EXP}$, there is a polynomial-time Turing machine V and a polynomial p such that for any input x:*

- *If $x \in A$, then there exists a proof oracle π such that*

$$\Pr_{|r|=p(|x|)}[V^{\pi}(x,r) \text{ accepts }] = 1. \tag{13}$$

- *If $x \notin A$, then for any proof oracle π*

$$\Pr_{|r|=p(|x|)}[V^{\pi}(x,r) \text{ accepts }] \leqslant \frac{1}{3}.$$

Moreover, V never makes more than k proof oracle queries, and there is a proof oracle $\tilde{\pi} \in \mathrm{EXP}$ independent of x such that (13) holds for $\pi = \tilde{\pi}$ in case $x \in A$.

Translating Lemma 4.7 into our terminology, we obtain:

Lemma 4.8 *There is a constant k such that for any set $A \in \mathrm{EXP}$, there is a probabilistic $\leqslant_{k-\mathrm{tt}}^{\mathrm{P}}$-reduction N, and a set $B \in \mathrm{EXP}$ such that for any input x:*

- *If $x \in A$, then $N^B(x)$ always accepts.*
- *If $x \notin A$, then for any oracle C, $N^C(x)$ accepts with probability at most $\frac{1}{3}$.*

Let R be a $\leqslant_{f(n)-\mathrm{tt}}^{\mathrm{P}}$-reduction of B to A, and consider the probabilistic reduction M^A that on input x, runs N on input x with oracle $R^{A \cup \{x\}}$. M^A is a probabilistic $\leqslant_{k \cdot f(n)-\mathrm{tt}}^{\mathrm{P}}$-reduction to A that never queries its own input. The following shows it defines a reduction from A:

- If $x \in A$, $R^{A \cup \{x\}} = R^A = B$, so $M^A(x) = N^B(x)$ always accepts.
- If $x \notin A$, then for $C = R^{A \cup \{x\}}$, $M^A(x) = N^C(x)$ accepts with probability at most $\frac{1}{3}$.

(Theorem 4.6) $\square$

Note that Theorem 4.6 makes it plausible why we did not manage to scale down Theorem 3.2 by one exponent to EXPSPACE in the nonadaptive setting, as we were able to do for our other results in Section 3 when going from the adaptive to the nonadaptive case: This would separate EXP from EXPSPACE.

We suggest the extension of Theorem 4.6 to the Δ-levels of the exponential-time hierarchy as an interesting problem for further research.

Second, Theorem 4.4 also holds for NP:

Theorem 4.9 *All $\leqslant_{tt}^{P}$-complete sets for* NP *are probabilistically nonadaptively autoreducible.*

Proof (of Theorem 4.9)
Fix a $\leqslant_{tt}^{P}$-complete set A for NP. Let R_A denote a length nondecreasing $\leqslant_{m}^{P}$-reduction of A to SAT.

Define the set

$$W = \{\langle \phi, 0^i \rangle \mid \phi \text{ is a Boolean formula with, say } m \text{ variables and } \exists\, a \in \Sigma^m : [\phi(a) \text{ and } a_i = 1]\}.$$

Since $W \in$ NP, there is a $\leqslant_{tt}^{P}$-reduction R_W from W to A.

We will use the following probabilistic algorithm by Valiant and Vazirani [18]:

Lemma 4.10 ([18]) *There exists a polynomial-time probabilistic Turing machine N that on input a Boolean formula φ with n variables, outputs another quantifier free Boolean formula $\phi = N(\varphi)$ such that:*

- *If φ is satisfiable, then with probability at least $\frac{1}{4n}$, ϕ has a unique satisfying assignment.*
- *If φ is not satisfiable, then ϕ is never satisfiable.*

Now consider the following algorithm for A: On input x, run N on input $R_A(x)$, yielding a Boolean formula ϕ with, say m variables, and it accepts iff

$$\phi(R_W^{A\cup\{x\}}(\langle \phi, 0\rangle), R_W^{A\cup\{x\}}(\langle \phi, 00\rangle), \ldots, R_W^{A\cup\{x\}}(\langle \phi, 0^i\rangle), \ldots, R_W^{A\cup\{x\}}(\langle \phi, 0^m\rangle))$$

evaluates to true. Note that this algorithm describes a probabilistic $\leqslant_{tt}^{P}$-reduction to A that never queries its own input. Moreover:

- If $x \in A$, then with probability at least $\frac{1}{4|x|}$, the Valiant-Vazirani algorithm N produces a Boolean formula ϕ with a unique satisfying assignment $\tilde{a}_\phi$. In that case, the assignment we use $(R_W^{A\cup\{x\}}(\langle \phi, 0\rangle), R_W^{A\cup\{x\}}(\langle \phi, 00\rangle), \ldots, R_W^{A\cup\{x\}}(\langle \phi, 0^i\rangle), \ldots, R_W^{A\cup\{x\}}(\langle \phi, 0^m\rangle))$ equals $\tilde{a}_\phi$, and we accept x.
- If $x \notin A$, any Boolean formula ϕ which N produces has no satisfying assignment, so we always reject x.

Executing $\Theta(n)$ independent runs of this algorithm, and accepting iff any of them accepts, yields a probabilistic nonadaptive autoreduction for A. (Theorem 4.9) □

So, for probabilistic autoreductions, we get similar results as for deterministic ones: Low end complexity classes turn out to have the property that their complete sets are autoreducible, whereas high end complexity classes do not. As we will see in more detail in the next section, this structural difference yields separations.

If we allow nonuniformity, the situation changes dramatically. Since probabilistic autoreducibility implies nonuniform autoreducibility [5], all our positive results for small complexity classes carry over to the nonuniform setting. But, as we will see next, the negative results do not, because also the complete sets for large complexity classes become autoreducible, both in the adaptive and in the nonadaptive case. So, uniformity is crucial for separating complexity classes using autoreducibility, and the Razborov-Rudich result [14] does not apply.

Feigenbaum and Fortnow [7] define the following concept of #P-robustness, of which we also consider the nonadaptive variant.

Definition 4.1 *A set A is* #P-robust *if* $\#P^A \subseteq FP^A$; *A is* nonadaptively #P-robust *if* $\#P^A_{tt} \subseteq FP^A_{tt}$.

Nonadaptive #P-robustness implies #P-robustness. For the usual deterministic and nondeterministic complexity classes containing PSPACE, all $\leqslant^P_T$-complete sets are #P-robust. For the deterministic classes containing PSPACE, it is also true that the $\leqslant^P_{tt}$-complete sets are nonadaptively #P-robust.

The following connection with nonuniform autoreducibility holds:

Theorem 4.11 *All #P-robust sets are nonuniformly autoreducible. All nonadaptively #P-robust sets are nonuniformly nonadaptively autoreducible.*

Proof
Feigenbaum and Fortnow [7] show that every #P-robust language is random-self-reducible. Beigel and Feigenbaum [5] prove that every random-self-reducible set is nonuniformly autoreducible (or "weakly coherent" as they call it). Their proofs carry over to the nonadaptive setting. □

It follows that the $\leqslant^P_{tt}$-complete sets for the usual deterministic complexity classes containing PSPACE are all nonuniformly nonadaptively autoreducible. The same holds for adaptive reductions, in which case the property is also true of nondeterministic complexity classes containing PSPACE. In particular, we get the following:

Corollary 4.12 *All $\leqslant^P_T$-complete sets for* NEXP, EXPSPACE, EEXP, NEEXP, EEXPSPACE, *... are nonuniformly autoreducible. All $\leqslant^P_{tt}$-complete sets for* PSPACE, EXP, EXPSPACE, *... are nonuniformly nonadaptively autoreducible.*

5 Separation Results

In this section, we will see how we can use the structural property of all complete sets being autoreducible to separate complexity classes. Based on the results of Sections 3 and 4, we only get separations that were already known: EXPH $\neq$ EEXPSPACE (by Theorems 4.3 and 3.1), EXP $\neq$ EEXPSPACE (by Theorems 4.6 and 3.2), and PH $\neq$ EXPSPACE (by Theorems 4.4 and 3.3, and also by scaling down EXPH $\neq$ EEXPSPACE). However, settling the question for certain other classes, would yield impressive new separations.

We summarize the implications in Figure 7.

Theorem 5.1 *In Figure 7, a positive answer to a question from the first column, implies the separation in the second column, and a negative answer, the separation in the third column.*

Most of the entries in Figure 7 follow directly from the results of the previous sections. In order to finish the table, we use the next lemma:

Lemma 5.2 *If* NP = NL, *we can decide the validity of QBF-formulae of size t and with γ alternations on a deterministic Turing machine M_1 in time $t^{O(c^\gamma)}$ and on a nondeterministic Turing machine M_2 in space $O(c^\gamma \log t)$, for some constant c.*

Proof (of Lemma 5.2)
Since coNP = NP, by Cook's Theorem we can transform in polynomial time a Π_1-formula with free variables into an equivalent Σ_1-formula with the same free variables, and vice versa. Since

question	yes	no
Are all $\leqslant_{\mathrm{T}}^{\mathrm{P}}$-complete sets for EXPSPACE autoreducible?	NL $\neq$ NP	PH $\neq$ PSPACE
Are all $\leqslant_{\mathrm{T}}^{\mathrm{P}}$-complete sets for EEXP autoreducible?	NL $\neq$ NP P $\neq$ PSPACE	PH $\neq$ EXP
Are all $\leqslant_{\mathrm{tt}}^{\mathrm{P}}$-complete sets for PSPACE $\leqslant_{\mathrm{tt}}^{\mathrm{P}}$-autoreducible?	NL $\neq$ NP	PH $\neq$ PSPACE
Are all $\leqslant_{\mathrm{tt}}^{\mathrm{P}}$-complete sets for EXP $\leqslant_{\mathrm{tt}}^{\mathrm{P}}$-autoreducible?	NL $\neq$ NP P $\neq$ PSPACE	PH $\neq$ EXP
Are all $\leqslant_{\mathrm{tt}}^{\mathrm{P}}$-complete sets for EXPSPACE probabilistically $\leqslant_{\mathrm{tt}}^{\mathrm{P}}$-autoreducible?	NL $\neq$ NP	P $\neq$ PSPACE

Figure 7: Separation results using autoreducibility

NP = P, we can decide the validity of Σ_1-formulae in polynomial-time. Say both the transformation algorithm T and the satisfiability algorithm S run in time n^c for some constant c.

Let ϕ be a QBF-formula of size t with γ alternations. Consider the following algorithm for deciding ϕ: Repeatedly apply the transformation T to the largest suffix that constitutes a Σ_1- or Π_1-formula until the whole formula becomes Σ_1, and then run S on it.

This algorithm correctly decides the truth of ϕ. Since the number of alternations decreases by one during every iteration, it makes at most γ calls to T, each time at most raising the length of the formula to the power c. It follows that the algorithm runs in time $t^{O(c^\gamma)}$.

Moreover, since P = NL, a padding argument shows that DTIME$[\tau] \subseteq$ NSPACE$[\log \tau]$ for any time constructible function τ. Therefore the result holds. (Lemma 5.2) □

This allows us to improve Theorems 3.2 and 3.3 as follows under the hypothesis NP = NL:

Theorem 5.3 *If* NP = NL, *there is a* $\leqslant_{2-\mathrm{T}}^{\mathrm{P}}$*-complete set for* EXPSPACE *that is not probabilistically autoreducible. The same holds for* EEXP *instead of* EXPSPACE.

Proof
Combine Lemma 5.2 with the probabilistic extension of Lemma 3.4 used in the proof of Theorem 3.2. □

Theorem 5.4 *If* NP = NL, *there is a* $\leqslant_{3-\mathrm{tt}}^{\mathrm{P}}$*-complete set for* PSPACE *that is not nonadaptively autoreducible. The same holds for* EXP *instead of* PSPACE.

Proof
Combining Lemma 5.2 with Lemma 3.5 for $\alpha(n) = n$ yields the result for EXP. The one for PSPACE follows, since NP = NL implies that EXP = PSPACE. □

Now, we have all ingredients for establishing Figure 7:

Proof (of Theorem 5.1)
The NL $\neq$ NP implications in the "yes"-column of Figure 7 immediately follow from Theorems 5.3 and 5.4 by contraposition.

By Theorem 3.1, a positive answer to the 2nd question in Figure 7 would yield EEXP $\neq$ EEXPSPACE, and by Theorem 3.3, a positive answer to the 4th question would imply EXP $\neq$ EXPSPACE. By padding, both translate down to P $\neq$ PSPACE.

Similarly, by Theorem 4.3, a negative answer to the 2nd question would imply EXPH $\neq$ EEXP, which pads down to PH $\neq$ EXP. A negative answer to the 4th question would yield PH $\neq$ EXP directly by Theorem 4.4. By the same token, a negative answer to the 1st question results in EXPH $\neq$ EXPSPACE and PH $\neq$ PSPACE, and a negative answer to the 3rd question in PH $\neq$ PSPACE. By Theorem 4.6, a negative answer to the last question implies EXP $\neq$ EXPSPACE and P $\neq$ PSPACE. $\square$

We note that we can tighten all of the separations in Figure 7 a bit, because we can apply Lemmata 3.4 and 3.5 to smaller classes than in Theorems 3.1 respectively 3.3. One improvement along these lines that might warrant attention is that we can replace "NL $\neq$ NP" in Figure 7 by "coNP $\not\subseteq$ NP $\cap$ NSPACE$[\log^{O(1)} n]$." This is because that condition suffices for Theorems 5.3 and 5.4, since we can strengthen Lemma 5.2 as follows:

Lemma 5.5 *If* coNP $\subseteq$ NP $\cap$ NSPACE$[\log^{O(1)} n]$*, we can decide the validity of QBF-formulae of size t and with γ alternations on a deterministic Turing machine M_1 in time $t^{O(c^\gamma)}$ and on a nondeterministic Turing machine M_2 in space $O(d^\gamma \log^d t)$, for some constants c and d.*

6 Conclusion

We have studied the question whether all complete sets are autoreducible for various complexity classes and various reducibilities. We obtained a positive answer for lower complexity classes in Section 4, and a negative one for higher complexity classes in Section 3. This way, we separated these lower complexity classes from these higher ones by highlighting a structural difference. The resulting separations were not new, but we argued in Section 5 that settling the very same question for intermediate complexity classes, would provide major new separations.

We believe that refinements to our techniques may lead to them, and would like to end with a few words about some thoughts in that direction.

One does not have to look at complete sets only. Let $\mathcal{C}_1 \subseteq \mathcal{C}_2$. Suppose we know that all complete sets for $\mathcal{C}_2$ are autoreducible. Then it suffices to construct, e.g., along the lines of Lemma 3.4, a hard set for $\mathcal{C}_1$ that is not autoreducible, in order to separate $\mathcal{C}_1$ from $\mathcal{C}_2$.

As we mentioned at the end of Section 5, we can improve Theorem 3.1 a bit by applying Lemma 3.4 to smaller space-bounded classes than EEXPSPACE. We cannot hope to gain much, though, since the coding in the proof of Lemma 3.4 seems to be DSPACE$[2^{n^{\beta(n)}}]$-complete because of the $\text{QBF}_{2\log\beta(n)}$-formulae of size $2^{O(n^{\beta(n)})}$ involved for inputs of size n. The same holds for Theorem 3.3 and Lemma 3.5.

Generalizations of autoreducibility may allow us to push things further. For example, one could look at $k(n)$-autoreducibility where $k(n)$ bits of the set remain unknown to the querying machine. Theorem 4.3 goes through for $k(n) \in O(\log n)$. Perhaps one can exploit this leeway in the coding of Lemma 3.4 and narrow the gap between the positive and negative results. As discussed in Section 5, that would yield interesting separations.

Finally, one may want to look at other properties than autoreducibility to realize Post's Program in complexity theory. Perhaps another concept from computability theory or a more artificial property can be used to separate complexity classes.

Acknowledgments

We would like to thank Manindra Agrawal and Ashish Naik for very helpful discussions. We are also grateful to Carsten Lund and Muli Safra for answering questions regarding the PCP Theorem. We thank the anonymous referees for their nice suggestions on how to present our results and for their meticulous proof reading.

References

[1] 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-Verlag, 1984.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.

[3] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity II*, volume 22 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1990.

[4] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*, volume 11 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1995.

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

[6] H. Buhrman, L. Fortnow, and L. Torenvliet. Using autoreducibility to separate complexity classes. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, pages 520–528. IEEE, 1995.

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

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

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

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

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

[12] Ch. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

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

[14] A. Razborov and S. Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55(1):24–35, 1997.

[15] R. Soare. *Recursively Enumerable Sets and Degrees*. Springer-Verlag, 1987.

[16] B. Trakhtenbrot. On autoreducibility. *Dokl. Akad. Nauk SSSR*, 192:1224–1227, 1970. In Russian. English translation in [17].

[17] B. Trakhtenbrot. On autoreducibility. *Soviet Mathematics–Doklady*, 11:814–817, 1970.

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

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