Electronic Colloquium on Computational Complexity, Report No. 12 (2016)

Autoreducibility of NP-Complete Sets*

John M. Hitchcock and Hadi Shafei
Department of Computer Science
University of Wyoming

Abstract

We study the polynomial-time autoreducibility of NP-complete sets and obtain separations under strong hypotheses for NP. Assuming there is a p-generic set in NP, we show the following:

- For every $k \geq 2$, there is a k-T-complete set for NP that is k-T autoreducible, but is not k-tt autoreducible or $(k-1)$-T autoreducible.
- For every $k \geq 3$, there is a k-tt-complete set for NP that is k-tt autoreducible, but is not $(k-1)$-tt autoreducible or $(k-2)$-T autoreducible.
- There is a tt-complete set for NP that is tt-autoreducible, but is not btt-autoreducible.

Under the stronger assumption that there is a p-generic set in NP $\cap$ coNP, we show:

- For every $k \geq 2$, there is a k-tt-complete set for NP that is k-tt autoreducible, but is not $(k-1)$-T autoreducible.

Our proofs are based on constructions from separating NP-completeness notions. For example, the construction of a 2-T-complete set for NP that is not 2-tt-complete also separates 2-T-autoreducibility from 2-tt-autoreducibility.

1 Introduction

Autoreducibility measures the redundancy of a set. For a reducibility $\mathcal{R}$, a set A is $\mathcal{R}$-autoreducible if there is a $\mathcal{R}$-reduction from A to A where the instance is never queried [15]. Understanding the autoreducibility of complete sets is important because of applications to separating complexity classes [5]. We study the polynomial-time autoreducibility [1] of NP-complete sets.

Natural problems are paddable and easily shown to be m-autoreducible. In fact, Glaßer et al. [8] showed that all nontrivial m-complete sets for NP and many other complexity classes are m-autoreducible. Beigel and Feigenbaum [4] showed that T-complete sets for NP and the levels of the polynomial-time hierarchy are T-autoreducible. We focus on intermediate reducibilities between many-one and Turing.

Previous work has studied separations of these autoreducibility notions for larger complexity classes. Buhrman et al. [5] showed there is a 3-tt-complete set for EXP that is not btt-autoreducible. For NEXP, Nguyen and Selman [13] showed there is a 2-T-complete set that is not 2-tt-autoreducible and a tt-complete set that is not btt-autoreducible. We investigate whether similar separations hold for NP.

*This research was support in part by NSF grant 0917417.

ISSN 1433-8092

Since all NP sets are 1-tt-autoreducible if P = NP, it is necessary to use a hypothesis at least as strong as P $\neq$ NP to separate autoreducibility notions. We work with the *Genericity Hypothesis* that there is a p-generic set in NP [3, 2]. This is stronger than P $\neq$ NP, but weaker than the *Measure Hypothesis* [12, 10] that there is a p-random set in NP. Under the Genericity Hypothesis, we separate many autoreducibility notions for NP-complete sets. Our main results are summarized in Table 1.1.

Previous work has used the measure and genericity hypotheses to separate completeness notions for NP. Consider the set

$$C = G \dot{\cup} (G \cap \mathrm{SAT}) \dot{\cup} (G \cup \mathrm{SAT}),$$

where $G \in$ NP and $\dot{\cup}$ is disjoint union. Then C is 2-T-complete for NP, and if G is p-generic, C is not 2-tt-complete [12, 2]. There is a straightforward 3-T (also 5-tt) autoreduction of C based on padding SAT.[1] However, since C is 2-T-honest-complete, we indirectly obtain a 2-T (also 3-tt) autoreduction by first reducing through SAT (Lemma 2.1). In Theorem 3.1 we show C is not 2tt-autoreducible.

It turns out this idea works in general. We show that many sets which separate completeness notions also separate autoreducibility notions. Ambos-Spies and Bentzien [2] also separated both k-T-completeness and $(k+1)$-tt-completeness from both k-tt-completeness and $(k-1)$-T-completeness for every $k \geq 3$ under the Genericity Hypothesis. We show that the same sets also separate k-T-autoreducibility and $(k+1)$-tt-autoreducibility from k-tt-autoreducibility and $(k-1)$-T-autoreducibility (Theorems 3.4 and 3.5). We also obtain that there is a tt-complete set for NP that is tt-autoreducible and not btt-autoreducible (Theorem 3.6), again using a construction of Ambos-Spies and Bentzien.

In the aforementioned results, there is a gap – we only separate k-tt-autoreducibility from $(k-2)$-T-autoreducibility (for $k \geq 3$), where we can hope for a separation from $(k-1)$-T-autoreducibility. The separation of k-tt from $(k-1)$-T is also open for completeness under the Genericity Hypothesis (or the Measure Hypothesis). To address this gap, we use a stronger hypothesis on the class NP $\cap$ coNP. Pavan and Selman [14] showed that if NP $\cap$ coNP contains a DTIME(2^{n^ϵ})-bi-immune set, then 2-tt-completeness is different from 1-tt-completeness for NP. We show that if NP $\cap$ coNP contains a p-generic set, then k-tt-completeness is different from $(k-1)$-T-completeness for all $k \geq 3$ (Theorem 4.2). We then show these constructions also separate autoreducibility: if there is a p-generic set in NP $\cap$ coNP, then for every $k \geq 2$, there is a k-tt-complete set for NP that is k-tt autoreducible, but is not $(k-1)$-T autoreducible (Theorems 4.1 and 4.3).

This paper is organized as follows. Preliminaries are in Section 2. The results using the Genericity Hypothesis are presented in Section 3. We use the stronger hypothesis on NP $\cap$ coNP in Section 4. Section 5 concludes with some open problems.

[1] Given an instance x of C, pad x to an instance y such that SAT$[x]$ = SAT$[y]$. We query $G[y]$ and then query either $G \cap$ SAT$[y]$ if $G[y] = 1$ or $G \cup$ SAT$[y]$ if $G[y] = 0$ to learn SAT$[y]$. Finally, if our instance is $G[x]$ the answer is obtained by querying $G \cap$SAT$[x]$ if SAT$[y] = 1$ or by querying $G \cup$SAT$[x]$ if SAT$[y] = 0$. If our instance is $G \cup$SAT$[x]$ or $G \cap$ SAT$[x]$, we query $G[x]$ and combine that answer with SAT$[y]$.

$\mathcal{C}$	$\mathcal{S}$	$\mathcal{R}$	notes
NP	k-T	k-tt	Theorem 3.1 ($k = 2$), Theorem 3.4 ($k \geq 3$)
NP	k-T	$(k-1)$-T	Theorem 3.1 ($k = 2$), Theorem 3.5 ($k \geq 3$)
NP	k-tt	$(k-1)$-tt	Corollary 3.2 ($k = 3$), Theorem 3.4 ($k \geq 4$)
NP	k-tt	$(k-2)$-T	Corollary 3.3 ($k = 3$), Theorem 3.5 ($k \geq 4$)
NP	tt	btt	Theorem 3.6
NP $\cap$ coNP	k-tt	$(k-1)$-T	Theorem 4.1 ($k = 2$), Theorem 4.3 ($k \geq 3$)

Table 1.1: If $\mathcal{C}$ contains a p-generic set, then there is a $\mathcal{S}$-complete set in NP that is $\mathcal{S}$-autoreducible but not $\mathcal{R}$-autoreducible.

2 Preliminaries

We use the standard enumeration of binary strings, i.e $s_0 = \lambda, s_1 = 0, s_2 = 1, s_3 = 00, ...$ as an order on binary strings. All languages in this paper are subsets of $\{0,1\}^*$ identified with their characteristic sequences. In other words, every language $A \in \{0,1\}^*$ is identified with $\chi_A = A[s_0]A[s_1]A[s_2]....$ If X is a set, equivalently a binary sequence, and $x \in \{0,1\}^*$ then $X \upharpoonright x$ is the initial segment of X for all strings before x, i.e the subset of X that contains every $y \in X$ that $y < x$.

All reductions in this paper are polynomial-time reductions, therefore we may not emphasize this every time we define a reduction. We use standard notions of reducibilities [11].

Given A, B, and $\mathcal{R} \in \{\text{m, T, tt, } k\text{-T, } k\text{-tt, btt}\}$, A is *polynomial-time $\mathcal{R}$-honest reducible* to B ($A \leq^{\text{p}}_{\mathcal{R}\text{-}h}$) if $A \leq^{\text{p}}_{\mathcal{R}}$ and there exist a constant c such that for every input x, every query q asked from B has the property $|x|^{1/c} < |q|$. In particular, a reduction $\mathcal{R}$ is called *length-increasing* if on every input the queries asked from the oracle are all longer than the input.

For any reduction $\mathcal{R} \in \{\text{m, T, tt, } k\text{-T, } k\text{-tt, btt}\}$ a language A is *$\mathcal{R}$-autoreducible* if $A \leq^{\text{p}}_{\mathcal{R}}$ via a reduction where on every instance x, x is not queried.

The following lemma states that any honest-complete set for NP is also autoreducible under the same type of reduction. This follows because NP has a paddable, length-increasing complete set.

Lemma 2.1. *Let $\mathcal{R} \in \{\text{m, T, tt, } k\text{-T, } k\text{-tt, btt, } \ldots\}$ be a reducibility. Then every $\mathcal{R}$-honest-complete set for* NP *is $\mathcal{R}$-autoreducible.*

Proof. Let $A \in$ NP be $\mathcal{R}$-honest-complete. Then there is an $\mathcal{R}$-honest reduction M from SAT to A. There exists $m \geq 1$ such that every query q output by M on an instance x satisfies $|q| \geq |x|^{\frac{1}{m}}$.

Since SAT is NP-complete via length-increasing many-one reductions, $A \leq^{\text{p}}_{\text{m}}$ SAT via a length-increasing reduction g. Since SAT is paddable, there is a polynomial-time function h such that for any y, SAT$[h(y)] =$ SAT$[y]$ and $|h(y)| > |y|^m$.

To obtain our $\mathcal{R}$-autoreduction of A, we combine g, h, and M. On instance x of A, compute the instance $h(g(x))$ of SAT and use M to reduce $h(g(x))$ to A. Since $|h(g(x))| > |g(x)|^m > |x|^m$, every query q of M has $|q| > |h(g(x))|^{\frac{1}{m}} > |x|$. Therefore all queries are different than x and this is an autoreduction. □

Most of the results in this paper are based on a non-smallness hypothesis for NP called the *Genericity Hypothesis* that NP contains a p-generic set [3, 2]. In order to define genericity first we need to define what a *simple extension function* is. For any k, a simple n^k-extension function is a partial function from $\{0,1\}^*$ to $\{0,1\}$ that is computable in $O(n^k)$. Given a set A and an extension function f we say that f is *dense along* A if f is defined on infinitely many initial segments of A. A set A *meets* a simple extension function f at x if $f(A \upharpoonright x)$ is defined and equal to $A[x]$. We say A meets f if A meets f at some x. A set G is called p-*generic* if it meets every simple n^k-extension function for any $k \geq 1$ [2]. A partial function $f : \{0,1\}^* \to (\{0,1\}^* \times \{0,1\})^*$ is called a *k-bounded extension function* if whenever $f(X \upharpoonright x)$ is defined, $f(X \upharpoonright x) = (y_0, i_0)...(y_m, i_m)$ for some $m < k$, and $x \leq y_0 < y_1 < ... < y_m$, where y_j's are strings and i_j's are either 0 or 1. A set A meets f at x if $f(A \upharpoonright x)$ is defined, and A agrees with f on all y_j's, i.e. if $f(A \upharpoonright x) = (y_0, i_0)...(y_m, i_m)$ then $A[y_j] = i_j$ for all $j \leq m$ [2].

We will use the following routine extension of a lemma in [2].

Lemma 2.2. *Let $l, c \geq 1$ and let f be an l-bounded partial extension function defined on initial segments $\alpha = X \upharpoonright 0^n$ of length 2^n ($n \geq 1$). Whenever $f(\alpha)$ is defined we have*

$$f(\alpha) = (y_{\alpha,1}, i_{\alpha,1}), ..., (y_{\alpha,l_\alpha}, i_{\alpha,l\alpha}),$$

where $l_\alpha \leq l$, $pos(\alpha) = (y_{\alpha,1}, ..., y_{\alpha,l_\alpha})$ is computable in 2^{cn} steps and $i_{\alpha,j}$ is computable in $2^{c|y_{\alpha,j}|}$ steps. Then for every p-*generic set G, if f is dense along G then G meets f.*

3 Autoreducibility Under the Genericity Hypothesis

We begin by showing the Genericity Hypothesis implies there is a 2-T-complete set that separates 2-T-autoreducibility from 2-tt-autoreducibility. The proof utilizes the construction of [12, 2] that of a set that separates 2-T-completeness from 2-tt-completeness.

Theorem 3.1. *If* NP *contains a* p-*generic language, then there exists a* 2-T-*complete set in* NP *that is* 2-T-*autoreducible, but not* 2-tt-*autoreducible.*

Proof. Let $G \in \text{NP}$ be p-generic and define $C = G \,\dot\cup\, (G \cap \text{SAT}) \,\dot\cup\, (G \cup \text{SAT})$, where $\dot\cup$ stands for disjoint union [12, 2]. Disjoint union can be implemented by adding a unique prefix to each set and taking their union. To be more clear, let $C = 0G \,\cup\, 10(G \cap \text{SAT}) \,\cup\, 11(G \cup \text{SAT})$. It follows from closure properties of NP that $C \in \text{NP}$.

To see that C is 2-T-complete, consider an oracle Turing machine M that on input x first queries $0x$ from C. If the answer is positive, i.e. $x \in G$, M queries $10x$ from C, and outputs the result. Otherwise, M queries $11x$ from C, and outputs the answer. This Turing machine always makes two queries from C, runs in polynomial time, and $M^C(x) = \text{SAT}[x]$. This completes the proof that C is also 2-T-completeness. Since all queries from SAT to C are length-increasing, it follows from Lemma 2.1 that C is 2-T-autoreducible.

The more involved part of the proof is to show that C is not 2-tt-autoreducible. To get a contradiction assume that C is 2-tt-autoreducible. This means there exist polynomial-time computable functions h, g_1, and g_2 such that for every $x \in \{0,1\}^*$,

$$C[x] = h(x, C[g_1(x)], C[g_2(x)])$$

and moreover $g_i(x) \neq x$ for $i = 1, 2$. Note that W.L.O.G. we can assume that $g_1(x) < g_2(x)$.
For $x = 0z$, $10z$, or $11z$ define the value of x to be z, and let $x = 0z$ for some string z. We have:

$$C[x] = G[z] = h(x, C[g_1(x)], C[g_2(x)])$$

To get a contradiction, we consider different cases depending on whether some of the queries have the same value as x or not, and the Boolean function $h(x, ., .)$. For some of these cases we show they can happen only for finitely many z's, and for the rest we show that SAT$[z]$ can be decided in polynomial time. As a result SAT is decidable in polynomial time a.e., which contradicts the assumption that NP contains a p-generic language.

- The first case is when values of $g_1(x)$ and $g_2(x)$ are different from z, and also different from each other. Assume this happens for infinitely many z's. We define an extension function f that is dense along G, so G has to meet it, but f is defined in a way that if G meets f, the autoreduction will be refuted. In order to define the value that f forces to $G[z]$ on the right hand side of the reduction, we define a function α that assigns 0 or 1 to queries of our autoreduction. The idea behind defining α is that its value on queries q_i is equal to $C[q_i]$ after we forced appropriate values into G, but computation of α can be done in at most 2^{2n} steps (given access to the partial characteristic sequence of G).

$$\alpha(w) = \begin{cases} C[w] & \text{if } w < x \\ 0 & \text{if } w > x \text{ and } w = 0\text{y or } 10\text{y for some y} \\ 1 & \text{if } w > x \text{ and } w = 11\text{y for some y} \end{cases}$$

  Note that in the first case, since $w < x$, the value of $C[w]$ is computable in 2^{2n} steps. Let $j = h(x, \alpha(g_1(x)), \alpha(g_2(x)))$. Later, when defining the extension function, we force the value of $C[x] = G[z]$ to be $1 - j$, hence refuting the autoreduction.
  The extension function f is defined whenever this case happens, and it forces three values into G. If $g_i(x) = 0v$ or $10v$ for some v, then $f(x)$ forces $G[v] = 0$. If $g_i(x) = 11v$ for some string v then $f(x)$ forces $G[v] = 1$. Finally, $f(x)$ forces $G[z] = 1 - j$. Since we assumed that this case happens for infinitely many x's, f is dense along G. Therefore G must meet f at some string $x = 0z$. But by the very definition of f this refutes the autoreduction. Hence this case can happen only for finitely many x's.

- In this case we consider the situation that $g_1(x)$ and $g_2(x)$ have the same value, say y, but $y \neq z$. If $y < z$ we can compute $C[g_1(x)]$ and $C[g_2(x)]$ and force $G[z] = 1 - h(x, C[g_1(x)], C[g_2(x)]$, which refutes the autoreduction. Therefore this cannot happen i.o. Now based on the prefixes of $g_1(x)$ and $g_2(x)$ we consider the following cases:

  1. If $g_1(x) = 0v$ and $g_2(x) = 10v$ we force $G[v] = 0$ and $G[x] = 1 - h(x, 0, 0)$. This refutes the autoreduction, therefore this case can happen only finitely many times.
  2. If $g_1(x) = 0v$ and $g_2(x) = 11v$ we force $G[v] = 1$ and $G[x] = 1 - h(x, 1, 1)$. This also refutes the autoreduction, so it cannot happen i.o.

  The only possibility that remains in this case is $g_1(x) = 10v$ and $g_2(x) = 11v$. In this case the autoreduction equality can be stated as:

$$G[z] = h(x, G \cap \text{SAT}[v], G \cup \text{SAT}[v])$$

  To show that this also cannot happen i.o. we need to look into different cases of the Boolean function $h(x, ., .)$.

1. If $h(x, a, b) = 0$, or 1, then force $G[z] = 1$ or 0 respectively. Therefore this Boolean function can occur only finitely many times.
2. If $h(x, a, b) = a$, in other words $G[z] = G \cap \mathrm{SAT}[v]$, force $G[z] = 1$ and $G[v] = 0$. This refutes the autoreduction, so this Boolean function cannot happen i.o.
3. If $h(x, a, b) = \neg a$, in other words $G[z] = \neg G \cap \mathrm{SAT}[v]$, force $G[z] = 0$ and $G[v] = 0$. This refutes the autoreduction, so this Boolean function cannot happen i.o.
4. If $h(x, a, b) = b$, in other words $G[z] = G \cup \mathrm{SAT}[v]$, force $G[z] = 0$ and $G[v] = 1$. This refutes the autoreduction, so this Boolean function cannot happen i.o.
5. If $h(x, a, b) = \neg b$, in other words $G[z] = \neg G \cup \mathrm{SAT}[v]$, force $G[z] = 1$ and $G[v] = 1$. This refutes the autoreduction, so this Boolean function cannot happen i.o.
6. If $h(x, a, b) = a \wedge b$, in other words $G[z] = (G \cap \mathrm{SAT}[v]) \wedge (G \cup \mathrm{SAT}[v])$, but this is equal to $G \cap \mathrm{SAT}[v]$. Therefore this case is similar to the second case.
7. If $h(x, a, b) = \neg a \wedge b$, in other words $G[z] = \neg(G \cap \mathrm{SAT}[v]) \wedge (G \cup \mathrm{SAT}[v])$. Force $G[z] = 1$ and $G[v] = \mathrm{SAT}[v]$. This contradicts the autoreduction equality. Therefore this case can happen only finitely many times.
8. If $h(x, a, b) = a \wedge \neg b$, in other words $G[z] = (G \cap \mathrm{SAT}[v]) \wedge \neg(G \cup \mathrm{SAT}[v])$, forcing $G[z] = 1$ refutes the autoreduction.
9. If $h(x, a, b) = \neg a \wedge \neg b$, in other words $G[z] = \neg(G \cap \mathrm{SAT}[v]) \wedge \neg(G \cup \mathrm{SAT}[v])$, but this is equal to $\neg G \cup \mathrm{SAT}[v]$. Therefore this case is similar to the fifth case.
10. If $h(x, a, b) = a \vee b$, in other words $G[z] = (G \cap \mathrm{SAT}[v]) \vee (G \cup \mathrm{SAT}[v])$, but this is equal to $G \cup \mathrm{SAT}[v]$. Therefore this case is similar to the fourth case.
11. If $h(x, a, b) = \neg a \vee b$, in other words $G[z] = \neg(G \cap \mathrm{SAT}[v]) \vee (G \cup \mathrm{SAT}[v])$. In this case forcing $G[z] = 0$ refutes the autoreduction.
12. If $h(x, a, b) = a \vee \neg b$, in other words $G[z] = (G \cap \mathrm{SAT}[v]) \vee \neg(G \cup \mathrm{SAT}[v])$. In this case forcing $G[z] = 0$ and $G[v] = \mathrm{SAT}[v]$ refutes the autoreduction.
13. If $h(x, a, b) = \neg a \vee \neg b$, in other words $G[z] = \neg(G \cap \mathrm{SAT}[v]) \vee \neg(G \cup \mathrm{SAT}[v])$, but this is equal to $\neg(G \cap \mathrm{SAT}[v])$. Therefore this case is similar to the third case.
14. If $h(x, a, b) = a \leftrightarrow b$, in other words $G[z] = (G \cap \mathrm{SAT}[v]) \leftrightarrow (G \cup \mathrm{SAT}[v])$. In this case $G[z] = 0$ and $G[v] = \mathrm{SAT}[v]$ refutes the autoreduction.
15. If $h(x, a, b) = \neg a \leftrightarrow b$, in other words $G[z] = \neg(G \cap \mathrm{SAT}[v]) \leftrightarrow (G \cup \mathrm{SAT}[v])$. In this case $G[z] = 1$ and $G[v] = \mathrm{SAT}[v]$ refutes the autoreduction.

We exhaustively went through all possible Boolean functions for the case where both queries have the same value which is different from the value of x, and showed that each one of them can happen only for finitely many x's. As a result this case can happen only for finitely many x's.

- This is the case when one of the queries, say $g_1(x)$ has the same value as x, but the other query has a different value. We only consider the case where $g_1(x) = 10z$. The other case, i.e. $g_1(x) = 11z$ can be done in a similar way. Again, we need to look at different possibilities for the Boolean function $h(x, ., .)$.

  1. $h(x, a, b) = 0$ or 1. Forcing $G[z] = 1$ or 0 respectively refutes the autoreduction.

2. $h(x,a,b) = a$, i.e. $G[z] = G \cap \mathrm{SAT}[z]$. If this happens i.o with $\mathrm{SAT}[z] = 0$ then we can refute the autoreduction by forcing $G[z] = 0$. Therefore in this case $\mathrm{SAT}[z] = 1$ a.e.
3. $h(x,a,b) = \neg a$, i.e. $G[z] = \neg(G \cap \mathrm{SAT}[z])$. By forcing $G[z] = 0$ we can refute the reduction. Therefore this case cannot happen i.o.
4. $h(x,a,b) = b$ or $\neg b$. Similar to previous cases.
5. $h(x,a,b) = a \wedge b$, i.e. $G[z] = (G \cap \mathrm{SAT}[z]) \wedge C[g_2(x)]$. In this case $\mathrm{SAT}[z]$ has to be 1 a.e.
6. $h(x,a,b) = \neg a \wedge b$, i.e. $G[z] = \neg(G \cap \mathrm{SAT}[z]) \wedge C[g_2(x)]$. If $g_2(x) = 0y$ or $10y$ for some y, then forcing $G[z] = 1$ and $G[v] = 0$ refutes the reduction. If $g_2(x) = 11y$ then we have $G[z] = \neg(G \cap \mathrm{SAT}[z]) \wedge (G \cup \mathrm{SAT}[y]$. Here we force $G[z] = 0$ and $G[y] = 1$.
7. $h(x,a,b) = a \wedge \neg b$, i.e. $G[z] = (G \cap \mathrm{SAT}[z]) \wedge \neg C[g_2(x)]$. In this case $\mathrm{SAT}[z] = 1$ a.e.
8. $h(x,a,b) = \neg a \wedge \neg b$, i.e. $G[z] = \neg(G \cap \mathrm{SAT}[z]) \wedge \neg C[g_2(x)]$. If $g_2(x) = 0y$ or $11y$ for some y, then forcing $G[z] = 1$ and $G[v] = 1$ refutes the reduction. If $g_2(x) = 10y$ then we have $G[z] = \neg(G \cap \mathrm{SAT}[z]) \wedge \neg(G \cap \mathrm{SAT}[y])$. Here we force $G[z] = 0$ and $G[y] = 0$.
9. $h(x,a,b) = a \vee b$, i.e. $G[z] = (G \cap \mathrm{SAT}[z]) \vee C[g_2(x)]$. If $g_2(x) = 0y$ or $11y$ for some y, then forcing $G[z] = 0$ and $G[v] = 1$ refutes the reduction. If $g_2(x) = 10y$ then we have $G[z] = (G \cap \mathrm{SAT}[z]) \vee (G \cap \mathrm{SAT}[y]$. This implies that $\mathrm{SAT}[z]$ must be 1 a.e.
10. $h(x,a,b) = \neg a \vee b$, i.e. $G[z] = \neg(G \cap \mathrm{SAT}[z]) \vee C[g_2(x)]$. In this case forcing $G[z] = 0$ refutes the reduction.
11. $h(x,a,b) = a \vee \neg b$, i.e. $G[z] = (G \cap \mathrm{SAT}[z]) \vee \neg C[g_2(x)]$. If $g_2(x) = 0y$ or $10y$ for some y, then forcing $G[z] = 0$ and $G[v] = 0$ refutes the reduction. If $g_2(x) = 11y$ then we have $G[z] = (G \cap \mathrm{SAT}[z]) \vee \neg(G \cup \mathrm{SAT}[y]$. This implies that $\mathrm{SAT}[z]$ must be 1 a.e.
12. $h(x,a,b) = \neg a \vee \neg b$, i.e. $G[z] = \neg(G \cap \mathrm{SAT}[z]) \vee \neg C[g_2(x)]$. In this case forcing $G[z] = 0$ refutes the reduction.
13. $h(x,a,b) = a \leftrightarrow b$, i.e. $G[z] = (G \cap \mathrm{SAT}[z]) \leftrightarrow C[g_2(x)]$. If $g_2(x) = 0y$ or $10y$ for some string y, then by forcing $G[z] = 0$ and $G[y] = 0$ we can refute the autoreduction. If $g_2(x) = 11y$, then we have $G[z] = (G \cap \mathrm{SAT}[z]) \leftrightarrow (G \cup \mathrm{SAT}[y])$. This implies that $\mathrm{SAT}[z] = 1$ a.e.
14. $h(x,a,b) = \neg a \leftrightarrow b$, i.e. $G[z] = \neg(G \cap \mathrm{SAT}[z]) \leftrightarrow C[g_2(x)]$. If $g_2(x) = 0y$ or $11y$ for some string y, then by forcing $G[z] = 0$ and $G[y] = 1$ we can refute the autoreduction. If $g_2(x) = 10y$, then we have $G[z] = \neg(G \cap \mathrm{SAT}[z]) \leftrightarrow (G \cup \mathrm{SAT}[y])$. This implies that $\mathrm{SAT}[z] = 1$ a.e.

- In this case we consider the situation where both queries $g_1(x)$ and $g_2(x)$ have the same value as x. In other words, in this case we have $g_1(x) = 10z$ and $g_2(x) = 11z$. Therefore we have:

$$G[z] = h(x, G \cap \mathrm{SAT}[z], G \cup \mathrm{SAT}[z])$$

To investigate this case we need to look at different Boolean functions for $h(x,.,.)$.

1. $h(x,a,b) = 0,\ 1,\ a,\ \neg a,\ b$, or $\neg b$. Each of these cases is similar to one of the cases discussed previously.

2. $h(x, a, b) = a \wedge b$, i.e. $G[z] = G \cap \mathrm{SAT}[z]$. This is also similar to one of the cases that we discussed previously.
3. $h(x, a, b) = \neg a \wedge b$, i.e. $G[z] = \neg(G \cap \mathrm{SAT}[z]) \wedge (G \cup \mathrm{SAT}[z])$. In this case $\mathrm{SAT}[z]$ must be 0 a.e.
4. $h(x, a, b) = a \wedge \neg b$, i.e. $G[z] = (G \cap \mathrm{SAT}[z]) \wedge \neg(G \cup \mathrm{SAT}[z])$. Forcing $G[z] = 1$ refutes the reduction.
5. $h(x, a, b) = \neg a \wedge \neg b$, i.e. $G[z] = \neg(G \cap \mathrm{SAT}[z]) \wedge \neg(G \cup \mathrm{SAT}[z])$. This is equal to $\neg(G \cup \mathrm{SAT}[z])$. Therefore forcing $G[z] = 0$ refutes the reduction.
6. $h(x, a, b) = a \vee b$, i.e. $G[z] = (G \cap \mathrm{SAT}[z]) \vee (G \cup \mathrm{SAT}[z])$, which is equal to $G \cup \mathrm{SAT}[z]$. Therefore $\mathrm{SAT}[z]$ must be 0 a.e.
7. $h(x, a, b) = \neg a \vee b$, i.e. $G[z] = \neg(G \cap \mathrm{SAT}[z]) \vee (G \cup \mathrm{SAT}[z])$. In this case $\mathrm{SAT}[z]$ must be 0 a.e.
8. $h(x, a, b) = a \vee \neg b$, i.e. $G[z] = (G \cap \mathrm{SAT}[z]) \vee \neg(G \cup \mathrm{SAT}[z])$. This implies that $\mathrm{SAT}[z]$ must be 1 a.e.
9. $h(x, a, b) = \neg a \vee \neg b$, i.e. $G[z] = \neg(G \cap \mathrm{SAT}[z]) \vee \neg(G \cup \mathrm{SAT}[z])$, which is equal to $\neg(G \cap \mathrm{SAT}[z])$. Therefore forcing $G[z] = 0$ refutes the autoreduction.
10. $h(x, a, b) = a \leftrightarrow b$, i.e. $G[z] = (G \cap \mathrm{SAT}[z]) \leftrightarrow (G \cup \mathrm{SAT}[z])$. In this case $\mathrm{SAT}[z]$ has to be 1 a.e.
11. $h(x, a, b) = \neg a \leftrightarrow b$, i.e. $G[z] = \neg(G \cap \mathrm{SAT}[z]) \leftrightarrow (G \cup \mathrm{SAT}[z])$. This implies that $\mathrm{SAT}[z]$ has to be 0 a.e.

□

Corollary 3.2. *If* NP *contains a* p*-generic language, then there exists a* 3-tt*-complete set for* NP *that is* 3-tt*-autoreducible, but not* 2-tt*-autoreducible.*

Proof. This follows immediately from Theorem 3.1 and the fact that every 2-T reduction is a 3-tt reduction. □

Corollary 3.3. *If* NP *contains a* p*-generic language, then there exists a* 3-tt*-complete set for* NP *that is* 3-tt*-autoreducible, but not* 1-T*-autoreducible.*

Our next theorem separates $(k + 1)$-tt-autoreducibility from k-tt-autoreducibility and k-T-autoreducibility from k-tt-autoreducibility under the Genericity Hypothesis. The proof uses the construction of Ambos-Spies and Bentzien [2] that separates the corresponding completeness notions.

Theorem 3.4. *If* NP *contains a* p*-generic language, then for every* $k \geq 3$ *there exists a set that is*

- $(k+1)$-tt*-complete for* NP *and* $(k+1)$-tt*-autoreducible,*
- k-T*-complete for* NP *and* k-T*-autoreducible, and*
- *not* k-tt*-autoreducible.*

Proof. Let $G \in \mathrm{NP}$ be a p-generic language, and $z_1, ..., z_{(k+1)}$ be the first $k+1$ strings of length k. For $m = 1, ..., k-1$ define

$$\hat{G_m} = \{x \mid xz_m \in G\} \tag{3.1}$$

$$\hat{G} = \bigcup_{m=1}^{k-1} \hat{G_m} \tag{3.2}$$

$$A = \bigcup_{m=1}^{k-1} \{xz_m \mid x \in \hat{G_m}\} \bigcup \{xz_k \mid x \in \hat{G} \cap \mathrm{SAT}\} \bigcup \{xz_{k+1} \mid x \in \hat{G} \cup \mathrm{SAT}\} \tag{3.3}$$

Here are some properties of the sets defined above:

- For every x, $x \in \hat{G} \Leftrightarrow \exists 1 \leq i \leq k-1.\ xz_i \in G$.
- A contains strings in G that end with $z_1, ...,$ or $z_{(k-1)}$, i.e. $A(xz_i) = G(xz_i)$ for every x and $1 \leq i \leq k-1$.
- $xz_k \in A$ if and only if $x \in \mathrm{SAT} \wedge (\exists 1 \leq i \leq k-1. xz_i \in G)$.
- $xz_{(k+1)} \in A$ if and only if $x \in \mathrm{SAT} \vee (\exists 1 \leq i \leq k-1. xz_i \in G)$.
- $xz_j \notin A$ for $j > k+1$.

It is easy to show that $\mathrm{SAT} \leq^{\mathrm{p}}_{(k+1)-\mathrm{tt}} A$. On input x, make queries $xz_1, ..., xz_{(k+1)}$ from A. If at least one of the answers to the first $k-1$ queries is positive, then $\mathrm{SAT}[x]$ is equal to the kth query, i.e. $\mathrm{SAT}[x] = A[xz_k]$. Otherwise $\mathrm{SAT}[x]$ is equal to $A[xz_{(k+1)}]$. As a result, A is $(k+1)$-tt-complete for NP. If the queries are allowed to be dependent, we can choose between xz_k and $xz_{(k+1)}$ based on the answers to the first $(k-1)$ queries. Therefore A is also k-T-complete for NP. Since all these queries are honest, in fact length-increasing, it follows from Lemma 2.1 that A is both $(k+1)$-tt-autoreducible and k-T-autoreducible.

To get a contradiction, assume A is k-tt-autoreducible via $h, g_1, ..., g_k$. In other words, assume that for every x:

$$A[x] = h(x, A[g_1(x)], ..., A[g_k(x)]) \tag{3.4}$$

and $\forall 1 \leq i \leq k.\ g_i(x) \neq x$. In particular, we are interested in the case where $x = 0^n z_1 = 0^{n+k}$, and we have:

$$A(0^{n+k}) = h(0^{n+k}, A[g_1(0^{n+k})], ..., A[g_k(0^{n+k})]) \tag{3.5}$$

and all $g_i(0^{n+k})$'s are different from 0^{n+k} itself.

In the following we will define a bounded extension function f that satisfies the condition in Lemma 2.2 such that if G meets f at 0^{n+k} then (3.5) will fail. We use the p-genericity of G to show that G has to meet f at 0^{n+k} for some n which completes the proof. In other words, we define a bounded extension function f such that given n and $X \upharpoonright 0^n$, $f(X \upharpoonright 0^n) = (y_0, i_0)...(y_m, i_m)$ and if

$$\begin{aligned} &G \upharpoonright 0^n = X \upharpoonright 0^n \ \text{ and} \\ &\forall 0 \leq j \leq m.\ G(y_j) = i_j \end{aligned} \tag{3.6}$$

then

$$A(0^{n+k}) \neq h(0^{n+k}, A[g_1(0^{n+k})], ..., A[g_k(0^{n+k})]) \tag{3.7}$$

Moreover, m is bounded by some constant that does not depend on n and $X \upharpoonright 0^n$. Note that we want f to satisfy the conditions in Lemma 2.2, so y_j's and i_j's must be computable in $O(2^n)$ and $O(2^{|y_j|})$ steps respectively. After defining such f, by Lemma 2.2 G must meet f at 0^{n+k} for some n. This means (3.6) must hold. As a result, (3.7) must happen for some n, which is a contradiction.
f can force values of $G[y_i]$'s for a constant number of y_i's. Because of the dependency between G and A we can force values for $A[w]$, where w is a query, by using f to force values in G. This is done based on the strings that have been queried, and their indices as follows.

- If $w = vz_i$ for some $1 \le i \le k-1$ then $A[w] = G[w]$. Therefore we can force $A[w]$ to 0 or 1 by forcing the same value for $G[w]$.
- If $w = vz_k$ then $A[w] = \mathrm{SAT}[v] \wedge (\bigvee_{l=1}^{k-1} G[vz_l])$, so by forcing all $G[vz_l]$'s to 0 we can make $A[w] = 0$.
- If $w = vz_{k+1}$ then $A[w] = \mathrm{SAT}[v] \vee (\bigvee_{l=1}^{k-1} G[vz_l])$. In this case by forcing one of the $G[vz_l]$'s to 1 we can make $A[w] = 1$.

We will use these facts to force the value of A on queries on input 0^{n+k} on the left hand side of (3.5), and then force a value for $A[0^{n+k}]$ such that (3.5) fails. The first problem that we encounter is the case where we have both vz_k and vz_{k+1} among our queries. If this happens for some v then the strategy described above does not work. To force $A[vz_k]$ and $A[vz_{k+1}]$ to 0 and 1 respectively, we need to compute $\mathrm{SAT}[v]$. If $\mathrm{SAT}[v] = 0$ then $A[vz_k] = 0$, and $A[vz_{k+1}]$ can be forced to 1 by forcing $G[vz_l] = 1$ for some $1 \le l \le k-1$. On the other hand, if $\mathrm{SAT}[v] = 1$ then $A[vz_{k+1}] = 1$, and forcing all $G[vz_l]$'s to 0 makes $A[vz_k] = 0$. This process depends on the value of $\mathrm{SAT}[v]$, and v can be much longer that 0^{n+k}. Because of the time bounds in Lemma 2.2 the value forced for $A[0^{n+k}]$ cannot depend on $\mathrm{SAT}[v]$. But note that we have k queries, and two of them are vz_k and vz_{k+1}. Therefore at least one of the strings $vz_1, ..., vz_{k-1}$ is not among the queries. We use this string as vz_l, and make $G[vz_l] = 1$ when $\mathrm{SAT}[v] = 0$.

Now we define an auxiliary function α from the set of queries, called QUERY, to 0 or 1. The idea is that α computes the value of A on queries without computing $G[v]$, given that G meets the extension function. α is defined in two parts based on the length of the queries. For queries $w = vz_p$ that are shorter than 0^{n+k}, i.e. $|w| < n+k$, we define:

$$\alpha(w) = \begin{cases} X[w] & \text{if } 1 \le p \le k-1 \\ 1 & \text{if } p = k \ \wedge\ v \in \mathrm{SAT} \ \wedge\ \exists 1 \le l \le k-1.\ vz_l \in X \\ 1 & \text{if } p = k+1 \ \wedge\ (v \in \mathrm{SAT} \ \vee\ \exists 1 \le l \le k-1.\ vz_l \in X) \\ 0 & \text{otherwise} \end{cases}$$

This means that if $X \upharpoonright 0^{n+k} = G \upharpoonright 0^{n+k}$ then $\alpha(w) = A(w)$ for every query $w = vz_p$ with $|w| < n+k$.
On the other hand, for queries $w = vz_p$ that $|w| \ge n+k$, α is defined as:

$$\alpha(w) = \begin{cases} 1 & \text{if } v = 0^n \ \wedge p = 2 \\ \mathrm{SAT}[v] & \text{if } v = 0^n \ \wedge\ p = k \\ 1 & \text{if } v = 0^n \ \wedge\ p = k+1 \\ 1 & \text{if } v \ne 0^n \wedge p = k+1 \\ 1 & \text{if } v \ne 0^n \wedge p = k-1 \wedge \forall l \in \{1, ..., k-1, k+1\}.\ vz_l \in \mathrm{QUERY} \\ 0 & \text{otherwise} \end{cases}$$

For this part of α, our definition of the extension function, which is provided below, guarantees that $\alpha(w) = A[w]$ if (3.6) holds. Note that the first case in the definition above implies that k must be greater than or equal to 3, and that is the reason this proof does not work for separating 3-tt-autoreducibility from 2-tt-autoreducibility.

Now we are ready to define the extension function f. For any string v which is the value for some query, i.e. $\exists 1 \leq p \leq k+1. vz_p \in$ QUERY, we define pairs of strings and 0 or 1's. These pairs will be part of our extension function. Fix some value v, and let r be the smallest index that $vz_r \notin$ QUERY, or $k-1$ if such index does not exist, i.e.

$$r = min\{s \geq 1 | vz_s \notin \text{QUERY} \vee s = k-1\} \tag{3.8}$$

We will have one of the following cases:

1. If $v = 0^n$ then pairs $(vz_2, 1), (vz_3, 0), ..., (vz_{k-1}, 0)$ must be added to f.
2. If $v \neq 0^n$ and $vz_{k+1} \notin$ QUERY then add pairs $(vz_1, 0)$,...,$(vz_{k-1}, 0)$ to f.
3. If $v \neq 0^n$, $vz_{k+1} \in$ QUERY and $vz_k \notin$ QUERY add pairs (vz_i, j) for $1 \leq i \leq k-1$ where $j = 0$ for all i's except $i = r$ where $j = 1$.
4. If $v \neq 0^n$, $vz_{k+1} \in$ QUERY and $vz_k \in$ QUERY add pairs (vz_i, j) for $1 \leq i \leq k-1$ where $j = 0$ for all i's except $i = r$ where $j = 1 - \text{SAT}[v]$.

This process must be repeated for every v that is the value of some query. Finally, we add $(0^{n+k}, 1 - h(0^{n+k}, \alpha(g_1(0^{n+k})), ..., \alpha(g_k(0^{n+k})))$ to f in order to refute the autoreduction. It is worth mentioning that in the fourth case above, since both vz_k and vz_{k+1} are among queries, at least one of the strings vz_1,...,$vz_{k-1}$is not queried. Therefore by definition of r, $vz_r \notin$ QUERY. This is important, as we describe in more detail later, because we forced $G[vz_r] = 1 - \text{SAT}[v]$, and if $vz_r \in$ QUERY then $\alpha(vz_r) = G[vz_r] = 1 - \text{SAT}[v]$. But α must be compuatable in $O(2^n)$ steps, which is not possible if v is much longer than 0^{n+k}.

Now that the extension function is defined completely, we need to show that it has the desired properties. First, we will show that if G meets f at 0^{n+k}, i.e. (3.6) holds, then α and A agree on every query w with $|w| \geq n+k$, i.e. $\alpha(w) = A[w]$.
Let $w = vz_p$, and $|w| \geq n+k$.

- If $v = 0^n$ and $p = 2$ then $\alpha(w) = 1$ and $A[w] = G[w] = 1$.
- If $v = 0^n$ and $p = k$ then $\alpha(w) = \text{SAT}[v]$ and $A[w] = \text{SAT}[v] \wedge (\bigvee_{l=1}^{k-1} G[vz_l])$. Since $G[vz_2] = 1$ is forced, $A[w] = \text{SAT}[v]$.
- If $v = 0^n$ and $p = k+1$ then $\alpha(w) = 1$ and $A[w] = \text{SAT}[v] \vee (\bigvee_{l=1}^{k-1} G[vz_l]) = 1$ since $G[vz_2] = 1$.
- If $v = 0^n$ and $p \neq 2, k, k+1$ then $\alpha(w) = A[w] = 0$.
- If $v \neq 0^n$ and $p < k-1$ then $\alpha(w) = 0$. Since $p < k-1$, and $vz_p \in$ QUERY, by definition of r, $r \neq p$. Therefore $G[vz_p]$ is forced to 0 by f. As a result, $A[w] = A[vz_p] = G[vz_p] = 0 = \alpha(w)$.
- If $v \neq 0^n$, $p = k-1$, and vz_1,...,vz_{k-1},$vz_{k+1} \in$ QUERY then $\alpha(w) = 1$. In this case $r = k-1$, so it follows from definition of f that $G[vz_{k-1}] = 1$. As a result, $A[w] = A[vz_{k-1}] = G[vz_{k-1}] = 1 = \alpha(w)$.

- If $v \neq 0^n$, $p = k-1$, and at least one of the strings vz_1,...,vz_{k-1},vz_{k+1} is not queried then we consider two cases. If $vz_{k+1} \notin$ QUERY then f forces $G[vz_{k-1}]$ to 0. On the other hand, if $vz_{k=1} \in$ QUERY, then at least one of vz_1,...,vz_{k-1} is not a query. Therefore by definition of r, $r \neq k-1$. This implies that $G[vz_{k-1}] = 0$ by f.

- If $v \neq 0^n$, $p = k$ then $\alpha(w) = 0$. Consider two cases. If $vz_{k+1} \notin$ QUERY then $G[vz_i] = 0$ for every $1 \leq i \leq k-1$. Therefore $A[w] = \text{SAT}[v] \wedge (\bigvee_{l=1}^{k-1} G[vz_l]) = 0$. Otherwise, when $vz_{k+1} \in$ QUERY, since we know that vz_k also belongs to QUERY, f forces $G[vz_r] = 1 - \text{SAT}[v]$, and $G[vz_i] = 0$ for every other $1 \leq i \leq k-1$. Therefore $A[w] = \text{SAT}[v] \wedge (\bigvee_{l=1}^{k-1} G[vz_l]) = \text{SAT}[v] \wedge (1 - \text{SAT}[v]) = 0$.

- If $v \neq 0^n$, $p = k+1$ then $\alpha(w) = 1$. If $vz_k \notin$ QUERY then $G[vz_r] = 1$ by f. Therefore $A[w] = \text{SAT}[v] \vee (\bigvee_{l=1}^{k-1} G[vz_l]) = 1$. On the other hand, if $vz_k \in$ QUERY then f forces $G[vz_r] = 1 - \text{SAT}[v]$. As a result, $A[w] = \text{SAT}[v] \vee (\bigvee_{l=1}^{k-1} G[vz_l]) = 1$.

This shows that in any case, $\alpha(w) = A[w]$ for $w \in$ QUERY, given that (3.6) holds, i.e G meets f. By combining this with (3.5) we have

$$\begin{aligned} A(0^{n+k}) =& h(0^{n+k}), A(g_1(0^{n+k})), ..., A(g_k(0^{n+k}))) \\ =& h(0^{n+k}, \alpha(g_1(0^{n+k})), ..., \alpha(g_k(0^{n+k}))) \end{aligned}$$

On the other hand, we forced $A[0^{n+k}] = 1 - h(0^{n+k}, \alpha(g_1(0^{n+k})), ..., \alpha(g_k(0^{n+k})))$ which gives us the desired contradiction.

The last part of our proof is to show that f satisfies the conditions in Lemma 2.2. For every value v which is the value of some query we added $k-1$ pairs to f, and there are k queries, which means at most k different values. Therefore, the number of pairs in f is bounded by k^2, i.e. f is a bounded extension function.

If $f(X \restriction 0^{n+k}) = (y_0, j_0), ..., (y_m, j_m)$ then y_i's are computable in polynomial ime in n, and j_i's are computable in $O(2^{|y_i|})$ because the most time consuming situation is when we need to compute SAT[v] which is doable in $O(2^n)$. For the condition forced to the left hand side of (3.5), i.e $G[0^{n+k}] = 1 - h(0^{n+k}, \alpha(g_1(0^{n+k})), ..., \alpha(g_k(0^{n+k})))$, note that $\alpha(w)$ can be computed in at most $O(2^n)$ steps for $w \in$ QUERY, and h is computable in polynomial time. □

Next we separate (k+1)-tt-autoreducibility and k-T-autoreducibility from (k−1)-T-autoreducibility. The proof uses the same construction from the previous theorem, which Ambos-Spies and Bentzien [2] showed separates these completeness notions.

Theorem 3.5. *If* NP *contains a* p-*generic language, then for every* $k \geq 3$ *there exists a set that is*

- $(k+1)$-tt-*complete for* NP *and* $(k+1)$-tt-*autoreducible,*

- k-T-*complete for* NP *and* k-T-*autoreducible, and*

- *not* $(k-1)$-T-*autoreducible.*

Proof. We use the same sets G and A as defined in the proof of Theorem 3.4. We proved that A is $(k+1)$-tt-complete, k-T-complete, $(k+1)$-tt-autoreducible, and k-T-autoreducible. What remains is to show that it is not $(k-1)$-T-autoreducible. The proof is very similar to what we did in the previous theorem, so we will not go through every detail here. Assume A is k-T-autoreducible via an oracle Turing machine M. In other words,

$$\forall x.\ A[x] = M^A(x) \tag{3.9}$$

and we assume that on input x, M will not query x itself. By using p-genericity of G we will show that there exists some n such that 3.9 fails for $x = 0^{n+k}$. In other words,

$$\exists n.\ A[0^{n+k}] = M^A(0^{n+k}) \tag{3.10}$$

Similar to what we did in the previous theorem, we define a bounded extension function f such that given n and an initial segment $X \upharpoonright 0^n$, f returns a set of pairs (y_i, j_i) for $0 \leq i \leq m$. y_i's are the positions, and must be computable in $O(2^n)$ steps, and j_i's are the values that f forces to y_i's. Each j_i must be computable in $O(2^{|y_i|})$. Then we will show that if G meets f at 0^{n+k}, i.e. if 3.6 holds, then 3.9 fails for $x = 0^{n+k}$. We will define a function α that under the right conditions simulates A on queries. We use α instead of A, as the oracle, in the computation of M on input 0^{n+k}. Similar to the previous theorem, α must be computable in $O(2^n)$ steps. Since in a Turing reduction each query may depend on the answers to the previous queries, we cannot know which queries will be asked in the computation of $M^A(0^{n+k})$ in $O(2^n)$ steps. Therefore we define α on every string rather than just on the set of queries.

Let $w = vz_p$ be some string. If $|w| < n + k$, then α is defined as:

$$\alpha(w) = \begin{cases} X[w] & \text{if } 1 \leq p \leq k-1 \\ 1 & \text{if } p = k \ \wedge \ v \in \text{SAT} \ \wedge \ \exists 1 \leq l \leq k-1.\ vz_l \in X \\ 1 & \text{if } p = k+1 \ \wedge \ (v \in \text{SAT} \ \vee \ \exists 1 \leq l \leq k-1.\ vz_l \in X) \\ 0 & \text{otherwise} \end{cases}$$

and if $|w| \geq n + k$ then:

$$\alpha(w) = \begin{cases} 1 & \text{if } v = 0^n \ \wedge p = 2 \\ \text{SAT}[v] & \text{if } v = 0^n \ \wedge \ p = k \\ 1 & \text{if } p = k+1 \\ 0 & \text{otherwise} \end{cases}$$

Now we run the same oracle Turing machine M, but we use α as the oracle instead of A. Let QUERY be the set of queries asked in this process. f will be defined in a similar fashion, except that the final pair which completes the diagonalization would be $(0^{n+k}, 1 - M^\alpha(0^{n+k}))$. Note that because there are at most $k - 1$ queries in both cases 3 and 4 in the definition of f, $vz_r \notin$ QUERY. In other words, the string we are forcing into G (hence into A) will never be queried.

Similar to the previous theorem, it can be verified that α and A agree on all queries, i.e. $M^A(0^{n+k}) = M^\alpha(0^{n+k})$, if 3.6 holds. It is also easy to prove that α is computable in $O(2^n)$ steps, therefore f satisfies the time bounds in Lemma 2.2. □

We now separate unbounded truth-table autoreducibility from bounded truth-table autoreducibility under the Genericity Hypothesis. This is based on the technique of Ambos-Spies and Bentzien [2] separating the corresponding completeness notions.

Theorem 3.6. *If* NP *has a* p*-generic language, then there exists a* tt*-complete set for* NP *that is* tt*-autoreducible, but not* btt*-autoreducible.*

Before proving Theorem 3.6, we need a few definitions and two lemmas.

A complexity class C is *computably presentable* if there is a computable function $f : \mathbb{N} \to \mathbb{N}$ such that $C = \{L(M_{f(i)}) \mid i \in \mathbb{N}\}$. A sequence of classes $C_0, C_1, \ldots$ is *uniformly computably presentable* if there is a computable function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that $C_j = \{L(M_{f(j,i)}) \mid i \in \mathbb{N}\}$ for all $j \in \mathbb{N}$. A reducibility $\mathcal{R}$ is *computably presentable* if there is a computable function $f : \mathbb{N} \to \mathbb{N}$ such that $M_{f(1)}, M_{f(2)}, \ldots$ is an enumeration of all $\mathcal{R}$-reductions.

Lemma 3.7. *If C is a computably presentable class which is closed under finite variants and $\mathcal{R}$ is a computably presentable reducibility, then $C_{\mathcal{R}\text{-}auto} = \{B \in C \mid B$ is $\mathcal{R}$-autoreducible$\}$ is also computably presentable.*

Proof. We prove the lemma for polynomial-time Turing autoreducibility, but similar proofs can be constructed for any kind of autoreduction that is computably presentable. For simplicity, we use C_{auto} for $C_{poly\text{-}T\text{-}auto}$ in the rest of the proof. If $C_{auto} = \emptyset$ then it is computably presentable by convention. Assume $C_{auto} \neq \emptyset$, and fix some set $A \in C_{auto}$. Since C is closed under finite variants, any finite variation of A must also belong to C_{auto}.

Let N_1,N_2,... be a presentation of C, and T_1,T_2,... be an enumeration of deterministic polynomial-time oracle Turing machines. For every pair $n = \langle i, j\rangle$ where $i, j \geq 1$ we define a Turing machine M_n as follows:

M_n	
input x	
for each y with $y < x$ do	
	test that $y \in L[N_i] \Leftrightarrow y \in L(T_j, L(N_i))$,
	and y itself has not been queried by T_j
if tests are true	
	then accept x iff $x \in L(N_i)$
	else accept x iff $x \in A$

Let L be an arbitrary language in C_{auto}. There must be some $i, j \geq 1$ such that $L = L(N_i)$ and T_j computes an $\mathcal{R}$-autoreduction on L. Therefore M_n computes L when $n = \langle i, j\rangle$. This means that every language in C_{auto} is accepted by some Turing machine M_n. On the other hand, for every $n = \langle i, j\rangle$, if T_j does not compute an $\mathcal{R}$-autoreduction on $L(N_i)$, then $L(M_n)$ is a finite variant of A. Since C is assumed to be closed under finite variants, $L(M_n) \in C_{auto}$. □

Lemma 3.8. (Ambos-Spies and Bentzien [2]) *Let $C_0, C_1, \ldots$ be classes such that,*
(1). *$C_0, C_1, \ldots$ is uniformly computably presentable.*
(2). *Each C_i is closed under finite variants.*
(3). *There is a decidable set D such that $D \subseteq \{0\}^* \times \Sigma^*$,*
and $D^{[n]} = \{x| < 0^n, x >\in D\} \notin C_n$.
(4). *$f : N \to N$ is a non-decreasing unbounded computable function.*
Then there exists a set A and a function $g : N \to N$ such that:
(5). *$A \notin \bigcup_{n=0}^{\infty} C_n$.*
(6). *$\forall n.\ A_{=n} = D^{[g(n)]}_{=n}$.*
(7). *g is polynomial-time computable with respect to the unary representation of numbers.*
(8). *$\forall n.\ g(n) \leq f(n)$.*

Proof of Theorem 3.6. Let $\widetilde{\mathrm{SAT}} = \{0^n1x \mid n \geq 0$ and $x \in \mathrm{SAT}\}$. It is easy to see that $\widetilde{\mathrm{SAT}}$ is NP-complete, and $\widetilde{\mathrm{SAT}} \in \mathrm{DTIME}(2^n)$. For every $k \geq 0$, let A_k be a $(k+3)$-tt-complete set constructed as before, by using $\widetilde{\text{SAT}}$ instead of SAT, and fix a p-generic set $G \in$ NP for the rest of the proof. Note that A_k is also $(k+3)$-tt-autoreducible, but not $(k+2)$-tt-complete or $(k+2)$-tt-autoreducible. Define $D = \{< 0^k, x > \mid k \geq 0 \text{ and } x \in A_k\}$. Since $A_k \in$ NP uniformly in k, $D \in$ NP. Let $C_k = \{B \in \text{NP} \mid B \text{ is } k\text{-tt-autoreducible}\}$ for $k \geq 1$ and $C_0 = C_1$. NP is computably presentable and closed under finite variants, therefore by Lemma 3.7, C_k's are computably presentable. In fact, they are uniformly computably presentable by applying the proof of Lemma 3.7 uniformly. It is also easy to see that each C_k is closed under finite variants. Therefore C_k's satisfy the conditions of Lemma 3.8. It follows from the definition of D that $D^{[k]} = A_k$, and we know that $A_k \notin C_k$ by construction of A_k. Therefore, if we take $f(n) = min\{m \mid 2m + 3 \geq n\}$, by Lemma 3.8 there exist A and g such that properties (5)-(8) from the lemma hold.

It follows from (6) and (7) that $\forall n.\ A_{=n} = D^{[g(n)]}_{=n}$, and g is polynomial time computable with respect to unary representation of numbers. This implies that $A \leq^{\text{p}}_{\text{m}} D$, therefore $A \in$ NP. Moreover, by (5) from the lemma, $A \notin \bigcup_{n \geq 0} C_n$, which means for every $k \geq 1$, A is not k-tt-autoreducible. In other words A is not btt-autoreducible.

To show that A is tt-autoreducible, we will show that SAT $\leq^{\text{p}}_{\text{tt}} A$ via honest reductions, and then it follows from Lemma 2.1 that A is tt-autoreducible. To define the truth-table reduction from SAT to A, fix x with $|x| = n$. For every $k, m \geq 0$ we have $\text{SAT}[x] = \widetilde{\text{SAT}}[0^m 1x]$, and $\widetilde{\text{SAT}}[0^m 1x]$ can be computed by making $(k+3)$ independent queries from $(A_k)_{=m+1+n+k+2}$ in polynomial time, uniformly in x, k, and m(This follows from $(k+3)$-tt-completeness of A_k, and the way A_k is defined using $\widetilde{\text{SAT}}$. (7) from Lemma 3.8 implies that:

$$A_{=2n+3} = (D^{[g(2n+3)]})_{=2n+3} = (A_{g(2n+3)})_{=2n+3} \tag{3.11}$$

We also know that $g(2n+3) \leq f(2n+3) \leq n$ for all n. Using all these facts, here is the truth-table reduction from SAT to A:

For x with $|x| = n$, compute $g(2n+3)$, and let $k = g(2n+3)$ and $m = n - k$. Therefore:

$$(A_k)_{=m+1+n+k+2} = (A_{g(2n+3)})_{=2n+3} = A_{=2n+3} \tag{3.12}$$

We know that $\text{SAT}[x] = \widetilde{\text{SAT}}[0^m 1x]$ can be computed by making $(k+3)$ independent queries from $(A_k)_{=m+1+n+k+2}$. This means $\text{SAT}[x] = \widetilde{\text{SAT}}[0^m 1x]$ can be recovered by making $g(2n+3)$ queries from $A_{=2n+3}$.

Note that all these queries are longer than x. Therefore, by Lemma 2.1, A is tt-autoreducible. □

4 Stronger Separations Under a Stronger Hypothesis

Our results so far only separate k-tt-autoreducibility from $(k-2)$-T-autoreducibility for $k \geq 3$ under the genericity hypothesis. In this section we show that a stronger hypothesis separates k-tt-autoreducibility from $(k-1)$-T-autoreducibility, for all $k \geq 2$. We note that separating k nonadaptive queries from $k-1$ adaptive queries is an optimal separation of bounded query reducibilities.

First we consider 2-tt-autoreducibility versus 1-tt-autoreducibility (equivalently, 1-T-autoreducibility). Pavan and Selman [14] showed that if NP ∩ coNP contains a DTIME(2^{n^ϵ})-bi-immune set, then 2-tt-completeness is different from 1-tt-completeness for NP. We show under the stronger hypothesis that NP ∩ coNP contains a p-generic set, we can separate the autoreducibility notions.

Theorem 4.1. *If* NP $\cap$ coNP *has a* p*-generic language, then there exists a* 2-tt*-complete set for* NP *that is* 2-tt*-autoreducible, but neither* 1-tt*-complete nor* 1-tt*-autoreducible.*

Proof. Assume $G \in \mathrm{NP} \cap \mathrm{coNP}$ is p-generic, and let $A = (G \cap \mathrm{SAT}) \dot\cup (\overline{G} \cap \mathrm{SAT})$, where $\overline{G}$ is G's complement, and $\dot\cup$ stands for disjoint union. We implement disjoint union as $A = (G \cap \mathrm{SAT})0 \,\dot\cup\, (\overline{G} \cap \mathrm{SAT})1$. It follows from closure properties of NP and the fact that $G \in \mathrm{NP} \cap \mathrm{coNP}$ that $A \in \mathrm{NP}$. It follows from definition of A that for every x, $x \in \mathrm{SAT} \leftrightarrow (x0 \in A \vee x1 \in A)$. This means $\mathrm{SAT} \leq^{\mathrm{p}}_{2\mathrm{tt}} A$. Therefore A is 2-tt-complete for NP. Since both queries in the above reduction are honest, in fact length increasing, it follows from Lemma 2.1 that A is 2-tt-autoreducible. To get a contradiction assume that A is 1-tt-autoreducible via polynomial-time computable functions h and g. In other words,

$$\forall x.\ A(x) = h(x, A[g(x)]) \tag{4.1}$$

and $g(x) \neq x$. Let $x = y0$ for some string y, then (4.1) turns into

$$\forall y.\ G \cap \mathrm{SAT}[y] = h(y0, A[g(y0)]) \tag{4.2}$$

and $g(y0) \neq y0$. We define a bounded extension function f whenever $\mathrm{SAT}[y] = 1$ as follows.

- Consider the case where $g(y0) = z0$ or $z1$ and $z > y$. If $g(y0) = z0$ then f forces $G[z] = 0$, and if $g(y0) = z1$ then f forces $G[z] = 1$. f also forces $G[y] = 1 - h(y0, 0)$. Since g and h are computable in polynomial time, so is f.

- On the other hand, if $g(y0) = z0$ or $z1$ and $z < y$ then define f such that it forces $G[y] = 1 - h(y0, A[g(y0)])$. Then f polynomial-time computable in this case as well because A may be computed on $g(y0)$ by looking up $G[z]$ from the partial characteristic sequence and deciding $\mathrm{SAT}[z]$ in $2^{O(|z|)}$ time.

- If $g(y0) = y1$ and $h(y0, .) = c$ is a constant function, then define f such that it forces $G[y] = 1 - c$.

If $g(y0) \neq y1 \wedge \mathrm{SAT}[y] = 1$ for infinitely many y, it follows from the p-genericity of G that G has to meet f, but this refutes the autoreduction. Similarly, $g(y0) = y1 \wedge h(y0, .) = \mathit{const} \wedge \mathrm{SAT}[y] = 1$ cannot happen for infinitely many y's. As a result, $(g(y0) = y1 \vee \mathrm{SAT}[y] = 0)$ and $h(y0, .)$ is not constant for all but finitely many y's. If $g(y0) = y1$ then h says either $G \cap \mathrm{SAT}[y] = \overline{G} \cap \mathrm{SAT}[y]$ or $G \cap \mathrm{SAT}[y] = \neg(\overline{G} \cap \mathrm{SAT}[y])$. It is easy to see this implies $\mathrm{SAT}[y]$ has to be 0 or 1, respectively. Based on the facts above, we define Algorithm 4.1 that decides SAT in polynomial time. This contradicts the assumption that NP $\cap$ coNP has a p-generic language.

It is proved in [8] that every nontrival 1-tt-complete set for NP is 1-tt-autoreducible, so it follows that A is not 1-tt-complete. □

We will show the same hypothesis on NP $\cap$ coNP separates k-tt-autoreducibilty from $(k-1)$-T-autoreducibility for all $k \geq 3$. First, we show the corresponding separation of completeness notions.

Theorem 4.2. *If* NP $\cap$ coNP *contains a* p*-generic set, then for every* $k \geq 3$ *there exists a* k-tt*-complete set for* NP *that is not* $(k-1)$-T*-complete.*

```
input y;
if g(y0) ≠ y1 ∨ h(y0, .) is constant then
│ Output NO;
else
│ if h(y0, .) is the identity function then
│ │ Output YES;
│ else
│ │ Output NO;
│ end
end
```

Algorithm 4.1: A polynomial-time algorithm for SAT

Proof. Assume $G \in \mathrm{NP} \cap \mathrm{coNP}$ is p-generic, and let $G_m = \{x \mid xz_m \in G\}$ for $1 \leq m \leq k$ where $z_1, ..., z_k$ are the first k strings of length k as before. Define

$$A = \Big[\bigcup_{m=1}^{k-1} \{xz_m \mid x \in G_m \cap \mathrm{SAT}\}\Big] \cup \{xz_k \mid x \in \big[\cap_{m=1}^{k-1} \overline{G_m}\big] \cap \mathrm{SAT}\} \tag{4.3}$$

It is easy to check that $x \in \mathrm{SAT} \Leftrightarrow \bigvee_{m=1}^{k}(xz_m \in A)$, therefore $\mathrm{SAT} \leq_{k-\mathrm{tt}}^{p} A$. It also follows from the fact that $G \in \mathrm{NP} \cap \mathrm{coNP}$ and the closure properties of NP that $A \in \mathrm{NP}$, so A is k-tt-complete for NP, in fact k-dtt-complete.

We claim that A is not $(k-1)$-T-hard for NP. For a contradiction, assume that $G_k \leq_{(k-1)-\mathrm{T}}^{p} A$. In other words, assume that there exists an oracle Turing machine M such that

$$\forall x.\ G_k[X] = M^A[x] \tag{4.4}$$

where M runs in polynomial time, and makes at most $(k-1)$ queries on every input. Given n and $X \upharpoonright 0^n$, we define a function α as follows.
If $w = vz_p$ and $|w| < n + k$ then

$$\alpha(w) = \begin{cases} X[w] \wedge \mathrm{SAT}[v] & \text{if } 1 \leq p \leq k-1 \\ \big[\bigwedge_{l=1}^{k-1} (1 - X[vz_l])\big] \wedge \mathrm{SAT}[v] & \text{if } p = k \\ 0 & \text{otherwise} \end{cases}$$

It is easy to see that α is defined in a way that if $X \upharpoonright 0^n = G \upharpoonright 0^n$ then $\alpha(w) = A[w]$.
On the other hand, if $|w| \geq n + k$ then $\alpha(w) = 0$ all the time. Later when we define the extension function we guarantee that $A[w] = 0$ for all long queries, by forcing the right values into G, which implies $A[w] = \alpha(w)$ for all queries. But before doing that, we run M on input 0^n with α as the oracle, and define QUERY to be the set of all queries made in this computation. We know that $|\mathrm{QUERY}| \leq k - 1$ therefore one of the following cases must happen:

1. $xz_k \notin$ QUERY.
2. $xz_k \in$ QUERY, and $\exists 1 \leq l \leq k-1.\ xz_l \notin$ QUERY.

Define a bounded extension function f based on the above cases. Given n and $X \upharpoonright 0^n$, $f(X \upharpoonright 0^n)$ contains the pairs described below. For every v which is the value of some element of QUERY,

1. If $vz_k \notin$ QUERY, then put $(vz_1, 0)$,...,$(vz_{(k-1)}, 0)$ into f. In other words, f forces $G[vz_l]$ to 0 for every $1 \leq l \leq k-1$.
2. If $vz_k \in$ QUERY then there must be some $1 \leq l \leq k-1$ such that $vz_l \notin$ QUERY. In this case f forces $G[vz_i] = 0$ for every $1 \leq i \leq k-1$ except for $i = l$ for which $G[vz_l] = 1$.

It can be shown that if G meets f at 0^n, i.e. if (3.6) holds, then $\alpha(w) = A[w]$ for every $w \in$ QUERY. As a result,

$$M^{\alpha}(0^n) = M^{A}(0^n) \tag{4.5}$$

To complete the diagonalization, we add one more pair to f that forces the value of $G_k[0^n] = G[0^{n+k}]$ to $1 - M^{\alpha}(0^n)$, i.e. $(0^{n+k}, 1 - M^{\alpha}(0^n))$. Then it follows from (4.5) that the reduction from G_k to A fails. The last part of the proof, is to show that G has to meet f at 0^n for some n. α is computable in $O(2^n)$ steps for short queries, and constant time for long queries, and M is a polynomial time Turing machine, which implies f can be computed in at most $O(2^{2n})$ steps. It is also easy to see that the number of pairs in f is bounded by k^2, which means f is a bounded extension function. As a result f satisfies the conditions of Lemma 2.2, hence G has to meet f at 0^n for some n, which completes the proof. □

Now we show the same sets separate k-tt-autoreducibility from $(k-1)$-T-autoreducibility.

Theorem 4.3. *If* NP ∩ coNP *contains a* p*-generic set, then for every* $k \geq 3$ *there exists a* k-tt-*complete set for* NP *that is* k-tt-*autoreducible, but is not* $(k-1)$-T-*autoreducible.*

Proof. Assume $G \in \mathrm{NP} \cap \mathrm{coNP}$ is p-generic, and let $G_m = \{x \mid xz_m \in G\}$ for $1 \leq m \leq k$ where $z_1, ..., z_k$ are the first k strings of length k as before. Define

$$A = \Big[\bigcup_{m=1}^{k-1} \{xz_m \mid x \in G_m \cap \mathrm{SAT}\}\Big] \cup \{xz_k \mid x \in \big[\cap_{m=1}^{k-1} \overline{G_m}\big] \cap \mathrm{SAT}\} \tag{4.6}$$

We showed that SAT $\leq^{\mathrm{p}}_{k-\mathrm{tt}} A$ via length-increasing queries, therefore by Lemma 2.1 A is k-tt-autoreducible. For a contradiction, assume that A is $(k-1)$-T-autoreducible. This means there exists an oracle Turing machine M such that

$$\forall x.\ A[x] = M^{A}(x) \tag{4.7}$$

M runs in polynomial time, and on every input x makes at most $k-1$ queries, none of which is x.
Given n and $X \upharpoonright 0^n$, we define a function α as follows.
If $w = vz_p$ and $|w| < n + k$ then

$$\alpha(w) = \begin{cases} X[w] \wedge \mathrm{SAT}[v] & \text{if } 1 \leq p \leq k-1 \\ \big[\bigwedge_{l=1}^{k-1} (1 - X[vz_l])\big] \wedge \mathrm{SAT}[v] & \text{if } p = k \\ 0 & \text{otherwise} \end{cases}$$

It is easy to see that if $X \upharpoonright 0^n = G \upharpoonright 0^n$ then $\alpha(w) = A[w]$.
If $w = vz_p$ and $|w| \geq n + k$, α is defined as:

$$\alpha(w) = \begin{cases} 1 & \text{if } v = 0^n \wedge 2 \leq p \leq k-1 \\ 0 & \text{if } v = 0^n \wedge p = k \\ 0 & \text{otherwise} \end{cases}$$

Note that α is not defined on 0^{n+k}, but that is fine because we are using α to compute $A[w]$ for w's that are queried when the input is 0^{n+k}, therefore 0^{n+k} will not be queried. Later we will define the extension function f in a way that if G meets f at 0^n then $\alpha(w) = A[w]$ for all queries.
Before defining f, we run M on input 0^{n+k} with α as the oracle instead of A, and define QUERY to be the set of all queries made in this computation. We know that M makes at most $k-1$ queries, therefore $|\text{QUERY}| \le k-1$. This implies that for every $v \ne 0^n$ which is the value of some element of QUERY one of the following cases must happen:

1. $vz_k \notin \text{QUERY}$
2. $vz_k \in \text{QUERY}$ and $\exists 1 \le l \le k-1 \, . \, vz_l \notin \text{QUERY}$

Given n and $X \restriction 0^n$, $f(X \restriction 0^n)$ is defined as follows if $\text{SAT}[0^n] = 1$.
For every v which is the value of some element of QUERY,

1. If $v = 0^n$, then add $(vz_2, 1), ..., (vz_{k-1}, 1)$ to f. In other words, f forces $G[0^n z_i] = 1$ for $2 \le i \le k-1$.
2. If $v \ne 0^n$ and $vz_k \notin \text{QUERY}$, then add $(vz_1, 0), ..., (vz_{k-1}, 0)$ to f.
3. If $v \ne 0^n$ and $vz_k \in \text{QUERY}$, then there must be some $1 \le l \le k-1$ such that $vz_l \notin \text{QUERY}$. In this case f forces $G[vz_i] = 0$ for every $1 \le i \le k-1$ except when $i = l$ for which we force $G[vz_l] = 1$.

To complete the diagonalization we add one more pair to f which is $(0^{n+k}, 1 - M^\alpha(0^n))$. It is straightforward, and similar to what has been done in the previous theorem, to show that if G meets f at 0^n for some n then α and A agree on every element of QUERY. Therefore $M^\alpha(0^n) = M^A(0^n)$, which results in a contradiction. It only remains to show that G meets f at 0^n for some n. This depends on the details of the encoding used for SAT. If $\text{SAT}[0^n] = 1$ for infinitely many n's, then f satisfies the conditions in Lemma 2.2. Therefore G has to meet f at 0^n for some n. On the other hand, if $\text{SAT}[0^n] = 0$ for almost all n, then we redefine A as:

$$A = \Big[\bigcup_{m=1}^{k-1} \{xz_m \mid x \in G_m \cup \text{SAT}\} \Big] \cup \{xz_k \mid x \in \big[\cup_{m=1}^{k-1} \overline{G_m} \big] \cup \text{SAT}\} \tag{4.8}$$

It can be proved, in a similar way and by using the assumption that $\text{SAT}[0^n] = 0$ for almost all n, that A is k-tt-complete, k-tt-autoreducible, but not $(k-1)$-T-autoreducible. □

5 Conclusion

We conclude with a few open questions.

For some k, is there a k-tt-complete set for NP that is not btt-autoreducible? We know this is true for EXP [5], so it may be possible to show under a strong hypothesis on NP. We note that by Lemma 2.1 any construction of a k-tt-complete set that is not k-tt-autoreducible must not be honest k-tt-complete. In fact, the set must be complete under reductions that are neither honest nor dishonest. On the other hand, for any $k \ge 3$, proving that all k-tt-complete sets for NP are btt-autoreducible would separate NP $\ne$ EXP.

Are the 2-tt-complete sets for NP 2-tt-autoreducible? The answer to this question is yes for EXP [7], so in this case a negative answer for NP would imply NP $\neq$ EXP. We believe that it may be possible to show the 2-tt-complete sets are nonuniformly 2-tt-autoreducible under the Measure Hypothesis – first show they are nonuniformly 2-tt-honest complete as an extension of [9, 6].

Nguyen and Selman [13] showed there is T-complete set for NEXP that is not tt-autoreducible. Can we do this for NP as well? Note that Hitchcock and Pavan [9] showed there is a T-complete set for NP that is not tt-complete.

Acknowledgment. We thank A. Pavan for extremely helpful discussions.

References

[1] K. Ambos-Spies. P-mitotic sets. In *Logic and Machines: Decision Problems and Complexity, Proceedings of the Symposium "Rekursive Kombinatorik" held from May 23-28, 1983 at the Institut für Mathematische Logik und Grundlagenforschung der Universität Münster/Westfalen*, pages 1–23, 1983.

[2] K. Ambos-Spies and L. Bentzien. Separating NP-completeness notions under strong hypotheses. *Journal of Computer and System Sciences*, 61(3):335–361, 2000.

[3] K. Ambos-Spies, H. Fleischhack, and H. Huwig. Diagonalizations over polynomial time computable sets. *Theoretical Computer Science*, 51:177–204, 1987.

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

[5] H. Buhrman, L. Fortnow, D. van Melkebeek, and L. Torenvliet. Separating complexity classes using autoreducibility. *SIAM Journal on Computing*, 29(5):1497–1520, 2000.

[6] H. Buhrman, B. Hescott, S. Homer, and L. Torenvliet. Non-uniform reductions. *Theory of Computing Systems*, 47(2):317–341, 2010.

[7] H. Buhrman and L. Torenvliet. A Post's program for complexity theory. *Bulletin of the EATCS*, 85:41–51, 2005.

[8] C. Glaßer, M. Ogihara, A. Pavan, A. L. Selman, and L. Zhang. Autoreducibility, mitoticity, and immunity. *J. Comput. Syst. Sci.*, 73(5):735–754, 2007.

[9] J. M. Hitchcock and A. Pavan. Comparing reductions to NP-complete sets. *Information and Computation*, 205(5):694–706, 2007.

[10] J. M. Hitchcock and A. Pavan. Hardness hypotheses, derandomization, and circuit complexity. *Computational Complexity*, 17(1):119–146, 2008.

[11] R. E. Ladner, N. A. Lynch, and A. L. Selman. A comparison of polynomial-time reducibilities. *Theoretical Computer Science*, 1(2):103–123, 1975.

[12] J. H. Lutz and E. Mayordomo. Cook versus Karp-Levin: Separating completeness notions if NP is not small. *Theoretical Computer Science*, 164(1–2):141–163, 1996.

[13] D. T. Nguyen and A. L. Selman. Non-autoreducible sets for NEXP. In *31st International Symposium on Theoretical Aspects of Computer Science*, pages 590–601, 2014.

[14] A. Pavan and A. L. Selman. Bi-immunity separates strong NP-completeness notions. *Information and Computation*, 188(1):116–126, 2004.

[15] B. Trakhtenbrot. On autoreducibility. *Dokl. Akad. Nauk SSSR*, 192(6):1224–1227, 1970. Translation in Soviet Math. Dokl. 11(3): 814817, 1970.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de