Electronic Colloquium on Computational Complexity, Report No. 188 (2013)

Autoreducibility and Mitoticity of Logspace-Complete Sets for NP and Other Classes

Christian Glaßer* Maximilian Witek*

13th December 2013

Abstract

We study the autoreducibility and mitoticity of complete sets for NP and other complexity classes, where the main focus is on logspace reducibilities. In particular, we obtain:

- For NP and all other classes of the PH: each $\leq^{\log}_{\mathrm{m}}$-complete set is $\leq^{\log}_{\mathrm{T}}$-autoreducible.
- For P, Δ^{p}_k, NEXP: each $\leq^{\log}_{\mathrm{m}}$-complete set is a disjoint union of two $\leq^{\log}_{\text{2-tt}}$-complete sets.
- For PSPACE: each $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-complete set is a disjoint union of two $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-complete sets.

1 Introduction

Complete sets for NP and other complexity classes are one of the main objects of research in theoretical computer science. However, basic questions regarding complete sets are still open. For instance: Is it possible to split each complete set of a certain class into two complete sets? If the answer is yes, then all complete sets of this class are in some sense redundant. In this paper we study two types of redundancy of sets.

- *Autoreducibility* of A: $A(x)$ can be efficiently computed from the values $A(y)$ for $y \neq x$.
- *Mitoticity* of A: The set A is a disjoint union of two sets that are equivalent to A.

There are several notions of autoreducibility depending on the computing resources and the number of values $A(y)$ that we can ask for. For each reducibility $\leq$, a set A is $\leq$-autoreducible, if there exists a $\leq$-reduction from A to A that on input x does not query x. Similarly, there are several notions of mitoticity depending on the notion of equivalence that is used. For each reducibility $\leq$, a set A is weakly $\leq$-mitotic, if there exists a set S such that A, $A \cap S$, and $A \cap \overline{S}$ are pairwise $\leq$-equivalent. If in addition it holds that $S \in \mathrm{P}$, then A is called $\leq$-mitotic.

Typical complete problems for NP, PSPACE, and other classes are not only polynomial-time-complete, but even logspace-complete, which brings us to the main question of this paper:

Does logspace-completeness imply logspace-autoreducibility or logspace-mitoticity?

We study this question for general complexity classes and conclude results for P, NP, Δ^{p}_k, Σ^{p}_k, Π^{p}_k, and NEXP.

Related Work. The notions of autoreducibility and mitoticity were originally studied in computability theory. Trakhtenbrot [Tra70] defined a set A to be *autoreducible* if there exists an oracle Turing machine M such that $A = L(M^A)$ and M on input x never queries x. Ladner [Lad73] defined a set A to be *mitotic* if it is the disjoint union of two sets of the same degree. He showed that a computably enumerable set is mitotic if and only if it is autoreducible.

Motivated by the hope to gain insight in the structure of sets in NP, Ambos-Spies [AS84] introduced and studied the variants of autoreducibility and mitoticity that are defined by polynomial-time many-one reducibility ($\leq^{\mathrm{p}}_{\mathrm{m}}$) and polynomial-time Turing reducibility ($\leq^{\mathrm{p}}_{\mathrm{T}}$).

*Julius-Maximilians-Universität Würzburg, {glasser,witek}@informatik.uni-wuerzburg.de

ISSN 1433-8092

Moreover, he introduced the distinction between mitoticity (splitting by some $S \in \mathrm{P}$) and weak mitoticity (splitting by an arbitrary S). For the study of sets inside P one needs refined notions of autoreducibility and mitoticity, which are obtained by using logspace reducibilities [GNR$^+$13].

It is easy to see that in general, mitoticity implies autoreducibility. Ambos-Spies [AS84] showed that $\leq^{\mathrm{p}}_{\mathrm{T}}$-autoreducibility does not imply $\leq^{\mathrm{p}}_{\mathrm{T}}$-mitoticity. Moreover, $\leq^{\mathrm{p}}_{\mathrm{m}}$-autoreducibility and $\leq^{\mathrm{p}}_{\mathrm{m}}$-mitoticity are equivalent [GPSZ08]. The same paper showed that $\leq^{\mathrm{p}}_{\mathrm{T}}$-autoreducibility does not imply weak $\leq^{\mathrm{p}}_{\mathrm{T}}$-mitoticity.

A matter of particular interest is the question of whether complete sets are autoreducible or mitotic. Ladner [Lad73] showed that there are Turing-complete sets for RE that are not mitotic. Over the years, researchers showed the polynomial-time mitoticity or at least the polynomial-time autoreduciblity of complete sets of prominent complexity classes: Beigel and Feigenbaum [BF92] proved that $\leq^{\mathrm{p}}_{\mathrm{T}}$-complete sets for all levels of the polynomial hierarchy and PSPACE are $\leq^{\mathrm{p}}_{\mathrm{T}}$-autoreducible. The same result holds for $\leq^{\mathrm{p}}_{\mathrm{m}}$ reducibility [GOP$^+$07, GPSZ08]. Buhrman, Hoene, and Torenvliet [BHT98] showed that $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete sets for EXP are weakly $\leq^{\mathrm{p}}_{\mathrm{m}}$-mitotic, which was later improved to $\leq^{\mathrm{p}}_{\mathrm{m}}$-mitotic [BT05]. Buhrman et al. [BFvMT00] proved that all $\leq^{\mathrm{p}}_{\mathrm{T}}$-complete sets for EXP are $\leq^{\mathrm{p}}_{\mathrm{T}}$-autoreducible. Moreover, the same paper contains interesting negative results like the existence of polynomial-time bounded-truth-table complete sets in EXP that are not polynomial-time bounded-truth-table autoreducible. Nguyen and Selman [NS14] showed negative autoreducibility results for NEXP. These and other results for polynomial-time reducibilities are summarized in Table 2. In a recent paper [GNR$^+$13], the authors studied autoreducibility and mitoticity also for logspace reducibilities (cf. Table 1).

Our Contribution. We prove the following general results on the autoreducibility and mitoticity of complete sets. Let $\mathcal{C} \supseteq (\mathrm{DSPACE}(\log \cdot \log^{(c)}) \cap \mathrm{P})$ be a complexity class that is closed under intersection, for some $c > 0$.

(a) If A is $\leq^{\log}_{\mathrm{m}}$-complete for $\mathcal{C}$ and $\leq^{\log}_{\mathrm{m}}$-autoreducible, then A is weakly $\leq^{\log}_{\text{2-dtt}}$-mitotic.
(b) If A is $\leq^{\log}_{\text{1-tt}}$-complete for $\mathcal{C}$ and $\leq^{\log}_{\text{1-tt}}$-autoreducible, then A is weakly $\leq^{\log}_{\text{2-tt}}$-mitotic.
(c) If A is $\leq^{\log}_{\mathrm{T}}$-hard for P and $\leq^{\mathrm{p}}_{\mathrm{tt}}$-autoreducible, then A is $\leq^{\log}_{\mathrm{T}}$-autoreducible.

The results (a) and (b) are particularly interesting for P, the other Δ^{p}_k levels of the polynomial hierarchy, and NEXP. Previously it was known that $\leq^{\log}_{\mathrm{m}}$-complete sets are $\leq^{\log}_{\mathrm{m}}$-autoreducible (resp., $\leq^{\log}_{\text{1-tt}}$-autoreducible). From (a) and (b) it follows:

- All $\leq^{\log}_{\mathrm{m}}$-complete sets for P and all other Δ^{p}_k levels are weakly $\leq^{\log}_{\text{2-tt}}$-mitotic.
- All $\leq^{\log}_{\mathrm{m}}$-complete sets for NEXP are weakly $\leq^{\log}_{\text{2-dtt}}$-mitotic.

So each of these sets is a disjoint union of two $\leq^{\log}_{\text{2-tt}}$-complete sets (resp., $\leq^{\log}_{\text{2-dtt}}$-complete sets). It remains an open question whether this can be improved to $\leq^{\log}_{\mathrm{m}}$-mitoticity. This question is interesting, since in contrast to $\leq^{\mathrm{p}}_{\mathrm{m}}$ reducibility, we do not know whether $\leq^{\log}_{\mathrm{m}}$-autoreducibility and $\leq^{\log}_{\mathrm{m}}$-mitoticity are equivalent (they are inequivalent relative to some oracle [Gla10]).

The result (c) is particularly interesting for NP, coNP and the other Σ^{p}_k and Π^{p}_k levels of the polynomial hierarchy, where only the polynomial-time autoreducibility and mitoticity of complete sets was known. Here we obtain logspace Turing autoreducibility:

- All $\leq^{\log}_{\mathrm{dtt}}$-complete or $\leq^{\log}_{\text{1-tt}}$-complete sets for NP, coNP, Σ^{p}_k, Π^{p}_k are $\leq^{\log}_{\mathrm{T}}$-autoreducible.

Finally, with our technique we also obtain a new result for the polynomial-time setting. Previously it was known that all $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-complete sets for PSPACE are $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-autoreducible. We obtain:

- All $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-complete sets for PSPACE are weakly $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-mitotic.

Again this means that each such set is a disjoint union of two $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-complete sets.

Table 1 and Table 2 summarize known results [Ber77, BF92, BFvMT00, BHT98, BT05, Buh93, GH92, GNR$^+$13, GOP$^+$07, GPSZ08, HKR93] for logspace and polynomial-time complete sets and emphasize the new results we obtained in this paper.

2 Preliminaries

We use standard notation for intervals of natural numbers, i.e., $[a,b] = \{a, a+1, \ldots, b\}$,$[a,b) = \{a, a+1, \ldots, b-1\}$, $(a,b] = \{a+1, a+2, \ldots, b\}$, and $(a,b) = \{a+1, a+2, \ldots, b-1\}$ for $a, b \in \mathbb{N}$. A set is called *trivial* if it is either finite or cofinite, and *non-trivial* otherwise. We will only consider non-trivial sets. For a set A let c_A denote its characteristic function, i.e., $c_A(x) = 1 \iff x \in A$. We denote the Boolean exclusive or by $\oplus$. For functions f and g, by $(f \circ g)$ we denote the composition of the functions, i.e., $(f \circ g)(x) := f(g(x))$. Let $f^{(i)}$ denote the i-th iteration of the function f. More formally, we define $f^{(0)}(x) := x$, and $f^{(i)}(x) := f(f^{(i-1)}(x))$ for $i > 0$. For a function f and some x, we will refer to the sequence $f^{(0)}(x), f^{(1)}(x), f^{(2)}(x), \ldots$ as *f's trace on x*. For $k \geq 1$, we say that a set S is a *k-ruling set (for f)* if for every x there exists some $i \leq k$ with $x \in S \iff f^{(i)}(x) \notin S$. Let log denote the logarithm to base 2. We will often use the iterated logarithm $\log^{(k)}$ for some $k > 0$. For the sake of simplicity, we define $\log(x) := 0$ for all $x < 1$, hence log and its iterations are total functions and are always greater than or equal to 0. For every x, by $|x|$ we denote the length of x's binary representation, by $\mathrm{abs}(x)$ we denote its absolute value, and by $\mathrm{sgn}(x)$ we denote the sign of x. We say that a function f is polynomially length-bounded if there exists a polynomial p such that $|f(x)| \leq p(|x|)$ holds for all x. When we use functions s and t as space and time bounds, we will assume that s and t are monotone functions.

Oracle Access. In general, Turing reducibility and truth-table reducibility are defined via oracle Turing machines that ask adaptive or nonadaptive queries to their oracle. However, for space-bounded relativized computations, there is no canonical way to define oracle access of Turing machines and thus to define logspace Turing reducibility. Below we define our oracle model and motivate the particular choice.

Ladner and Lynch [LL76] define logspace Turing reducibility by the restrictive oracle access model where a logspace oracle Turing machine has a single one-way write-only oracle tape that is erased after each query and that is not subject to space bounds. By Ladner [Lad75], logspace Turing reducibility thus defined is reflexive and transitive, so this model is reasonable. However, Ladner and Lynch [LL76] show that under this model, logspace Turing reducibility and logspace truth-table reducibility are the same. Hence, this model is so restrictive that the oracle Turing machine cannot really react on answers to queries it asked. So, in order to obtain adaptive Turing reductions, we need to relax some of the restrictions.

Lynch [Lyn78] notes that already minor modifications (nonerasure of oracle tapes, counting space on the oracle tape, and more than one oracle tape, for instance) of the restrictive oracle access model used in [LL76] can lead to a different reducibility notion (see also Hemaspaandra and Jiang [HJ97] and Glaßer [Gla10] for further models and a comparison of their strength).

Since we want to distinguish between adaptive and nonadaptive reductions, we will use the less restrictive multi-tape oracle access model proposed by Lynch [Lyn78]. In this setting, an oracle Turing machine consists of a single read-write working tape that is subject to the space bounds and an arbitrary but fixed number of write-only oracle tapes that are not subject to the space bounds. In each step, the oracle Turing machine may ask the query that is written on some particular oracle tape, after which the oracle Turing machine enters an answer state accordingly, and erases the particular oracle tape again. Under this oracle access model, Lynch [Lyn78] shows transitivity of logspace Turing reducibility and separates this notion from logspace

$\leq$	P	NP, coNP	Δ_k^{p}	$\Sigma_k^{\mathrm{p}}, \Pi_k^{\mathrm{p}}$	PSPACE	EXP	NEXP
$\leq_{\mathrm{m}}^{\log}$	$\mathrm{A}_{\text{1-tt}}^{\log}$, $\boxed{\mathrm{W}_{\text{2-tt}}^{\log}}$		$\mathrm{A}_{\text{1-tt}}^{\log}$, $\boxed{\mathrm{W}_{\text{2-tt}}^{\log}}$		$\mathrm{M}_{\mathrm{m}}^{\log}$	$\mathrm{M}_{\mathrm{m}}^{\log}$	$\mathrm{A}_{\mathrm{m}}^{\log}$, $\boxed{\mathrm{W}_{\text{2-dtt}}^{\log}}$
$\leq_{l\text{-ctt}}^{\log}$	$\mathrm{A}_{l\text{-tt}}^{\log}$		$\mathrm{A}_{l\text{-tt}}^{\log}$		$\mathrm{M}_{l\text{-ctt}}^{\log}$	$\mathrm{M}_{l\text{-ctt}}^{\log}$	$\mathrm{A}_{l\text{-ctt}}^{\log}$
$\leq_{l\text{-dtt}}^{\log}$	$\mathrm{A}_{l\text{-tt}}^{\log}$		$\mathrm{A}_{l\text{-tt}}^{\log}$		$\mathrm{M}_{l\text{-dtt}}^{\log}$	$\mathrm{M}_{l\text{-dtt}}^{\log}$	$\mathrm{A}_{l\text{-dtt}}^{\log}$
$\leq_{\mathrm{ctt}}^{\log}$					$\mathrm{M}_{\mathrm{ctt}}^{\log}$	$\mathrm{M}_{\mathrm{ctt}}^{\log}$	$\mathrm{A}_{\mathrm{ctt}}^{\log}$
$\leq_{\mathrm{dtt}}^{\log}$		$\boxed{\mathrm{A}_{\mathrm{T}}^{\log}}$		$\boxed{\mathrm{A}_{\mathrm{T}}^{\log}}$	$\mathrm{M}_{\mathrm{dtt}}^{\log}$	$\mathrm{M}_{\mathrm{dtt}}^{\log}$	$\mathrm{A}_{\mathrm{dtt}}^{\log}$
$\leq_{\text{1-tt}}^{\log}$	$\mathrm{A}_{\text{2-tt}}^{\log}$	$\boxed{\mathrm{A}_{\mathrm{T}}^{\log}}$		$\boxed{\mathrm{A}_{\mathrm{T}}^{\log}}$	$\mathrm{M}_{\mathrm{m}}^{\log}$	$\mathrm{M}_{\mathrm{m}}^{\log}$	$\mathrm{A}_{\mathrm{m}}^{\log}$, $\boxed{\mathrm{W}_{\text{2-dtt}}^{\log}}$
$\leq_{\text{2-tt}}^{\log}$					$\mathrm{M}_{\text{2-tt}}^{\log}$	$\mathrm{M}_{\text{2-tt}}^{\log}$	$\mathrm{A}_{\text{2-tt}}^{\log}$
$\leq_{\mathrm{btt}}^{\log}$	$\mathrm{A}_{\text{log-T}}^{\log[1]}$		$\mathrm{A}_{\text{log-T}}^{\log[1]}$		X_1	X_2	
$\leq_{\mathrm{tt}}^{\log}$	$\mathrm{A}_{\mathrm{tt}}^{\log}$		$\mathrm{A}_{\mathrm{tt}}^{\log}$				

Table 1: Redundancy of logspace complete sets, where $l \geq 1$ and $k \geq 2$. For the cell in row $\leq_r$ and column $\mathcal{C}$, the entry A_s means that every $\leq_r$-complete set for $\mathcal{C}$ is $\leq_s$-autoreducible. Analogously, the entry W_s means that every $\leq_r$-complete set for $\mathcal{C}$ is weakly $\leq_s$-mitotic, and the entry M_s means that every $\leq_r$-complete set for $\mathcal{C}$ is $\leq_s$-mitotic. For the cells marked with X_1 and X_2, negative results are known: There is a $\leq_{\mathrm{btt}}^{\log}$-complete set for PSPACE that is not $\leq_{\mathrm{btt}}^{\log}$-autoreducible [GNR$^+$13] and a $\leq_{\mathrm{btt}}^{\log}$-complete set for EXP that is not $\leq_{\mathrm{btt}}^{\mathrm{p}}$-autoreducible (and hence not $\leq_{\mathrm{btt}}^{\log}$-autoreducible) [BFvMT00]. Results implied by universal relations between reductions are omitted, and results obtained in this paper are framed. For the definitions of the reductions, see section 2.

$\leq$	NP	Δ_k^{p}	$\Sigma_k^{\mathrm{p}}, \Pi_k^{\mathrm{p}}$	PSPACE	EXP	NEXP
$\leq_{\mathrm{m}}^{\mathrm{p}}$	$\mathrm{M}_{\mathrm{m}}^{\mathrm{p}}$	$\mathrm{M}_{\mathrm{m}}^{\mathrm{p}}$	$\mathrm{M}_{\mathrm{m}}^{\mathrm{p}}$	$\mathrm{M}_{\mathrm{m}}^{\mathrm{p}}$	$\mathrm{M}_{\mathrm{m}}^{\mathrm{p}}$	$\mathrm{M}_{\mathrm{m}}^{\mathrm{p}}$
$\leq_{\text{1-tt}}^{\mathrm{p}}$	$\mathrm{M}_{\text{1-tt}}^{\mathrm{p}}$	$\mathrm{M}_{\text{1-tt}}^{\mathrm{p}}$	$\mathrm{M}_{\text{1-tt}}^{\mathrm{p}}$	$\mathrm{M}_{\text{1-tt}}^{\mathrm{p}}$	$\mathrm{M}_{\mathrm{m}}^{\mathrm{p}}$	$\mathrm{M}_{\mathrm{m}}^{\mathrm{p}}$
$\leq_{\text{2-tt}}^{\mathrm{p}}$					$\mathrm{M}_{\text{2-tt}}^{\mathrm{p}}$	$\mathrm{A}_{\text{2-tt}}^{\mathrm{p}}$
$\leq_{l\text{-ctt}}^{\mathrm{p}}$					$\mathrm{M}_{l\text{-ctt}}^{\mathrm{p}}$	$\mathrm{A}_{l\text{-ctt}}^{\mathrm{p}}$
$\leq_{l\text{-dtt}}^{\mathrm{p}}$	$\mathrm{A}_{l\text{-dtt}}^{\mathrm{p}}$	$\mathrm{A}_{l\text{-dtt}}^{\mathrm{p}}$	$\mathrm{A}_{l\text{-dtt}}^{\mathrm{p}}$	$\mathrm{A}_{l\text{-dtt}}^{\mathrm{p}}$, $\boxed{\mathrm{W}_{l'\text{-dtt}}^{\mathrm{p}}}$	$\mathrm{M}_{l\text{-dtt}}^{\mathrm{p}}$	$\mathrm{A}_{l\text{-dtt}}^{\mathrm{p}}$
$\leq_{\mathrm{btt}}^{\mathrm{p}}$					X_3	
$\leq_{\mathrm{ctt}}^{\mathrm{p}}$					$\mathrm{M}_{\mathrm{ctt}}^{\mathrm{p}}$	$\mathrm{A}_{\mathrm{ctt}}^{\mathrm{p}}$
$\leq_{\mathrm{dtt}}^{\mathrm{p}}$	$\mathrm{A}_{\mathrm{dtt}}^{\mathrm{p}}$	$\mathrm{A}_{\mathrm{dtt}}^{\mathrm{p}}$	$\mathrm{A}_{\mathrm{dtt}}^{\mathrm{p}}$	$\mathrm{A}_{\mathrm{dtt}}^{\mathrm{p}}$, $\boxed{\mathrm{W}_{\mathrm{dtt}}^{\mathrm{p}}}$	$\mathrm{M}_{\mathrm{dtt}}^{\mathrm{p}}$	$\mathrm{A}_{\mathrm{dtt}}^{\mathrm{p}}$
$\leq_{\mathrm{tt}}^{\mathrm{p}}$	$\mathrm{A}_{\mathrm{tt}}^{\mathrm{BPP}}$	$\mathrm{A}_{\mathrm{tt}}^{\mathrm{p}}$			$\mathrm{A}_{\mathrm{tt}}^{\mathrm{BPP}}$	
$\leq_{\mathrm{T}}^{\mathrm{p}}$	$\mathrm{A}_{\mathrm{T}}^{\mathrm{p}}$	$\mathrm{A}_{\mathrm{T}}^{\mathrm{p}}$	$\mathrm{A}_{\mathrm{T}}^{\mathrm{p}}$	$\mathrm{A}_{\mathrm{T}}^{\mathrm{p}}$	$\mathrm{A}_{\mathrm{T}}^{\mathrm{p}}$	

Table 2: Redundancy of polynomial-time complete sets, where $l \geq 1$, $l' = l(l^2 + l + 1)$, and $k \geq 2$. The entries of the table are read analogously to the entries in Table 1. Recall that there is a $\leq_{\mathrm{btt}}^{\log}$-complete (and hence also $\leq_{\mathrm{btt}}^{\mathrm{p}}$-complete) set for EXP that is not $\leq_{\mathrm{btt}}^{\mathrm{p}}$-autoreducible [BFvMT00], hence we have a negative result marked by X_3.

truth-table reducibility. Furthermore, in this setting, logspace reducibility implies polynomial-time reducibility. We also note that the introduction of additional oracle tapes does not change the computational power of polynomial-time oracle Turing machines, since within a polynomial time bound, all queries can be stored on a working tape as well.

Reductions. For sets A and B we say that A is polynomial-time Turing reducible to B ($A \leq^{\mathrm{p}}_{\mathrm{T}} B$), if there exists a polynomial-time oracle Turing machine that accepts A with B as its oracle. If M on input x asks at most $O(\log |x|)$ queries, then A is polynomial-time log-Turing reducible to B ($A \leq^{\mathrm{p}}_{\text{log-T}} B$). If M's queries are nonadaptive (i.e., independent of the oracle), then A is polynomial-time truth-table reducible to B ($A \leq^{\mathrm{p}}_{\mathrm{tt}} B$). If M asks at most k nonadaptive queries, then A is polynomial-time k-truth-table reducible to B ($A \leq^{\mathrm{p}}_{k\text{-tt}} B$). A is polynomial-time bounded-truth-table reducible to B ($A \leq^{\mathrm{p}}_{\mathrm{btt}} B$), if $A \leq^{\mathrm{p}}_{k\text{-tt}} B$ for some k. A is polynomial-time disjunctive-truth-table reducible to B ($A \leq^{\mathrm{p}}_{\mathrm{dtt}} B$), if there exists a polynomial-time-computable function f such that for all x, $f(x) = \langle y_1, y_2, \ldots, y_n \rangle$ for some $n \geq 1$ and ($x \in A \iff c_B(y_1) \vee c_B(y_2) \vee \cdots \vee c_B(y_n)$). If n is bounded by some constant k, then A is polynomial-time k-disjunctive-truth-table reducible to B ($A \leq^{\mathrm{p}}_{k\text{-dtt}} B$). A is polynomial-time bounded-disjunctive-truth-table reducible to B ($A \leq^{\mathrm{p}}_{\mathrm{bdtt}} B$), if $A \leq^{\mathrm{p}}_{k\text{-dtt}} B$ for some k. The polynomial-time conjunctive-truth-table reducibilities $\leq^{\mathrm{p}}_{\mathrm{ctt}}$, $\leq^{\mathrm{p}}_{k\text{-ctt}}$, and $\leq^{\mathrm{p}}_{\mathrm{bctt}}$ are defined analogously. A is polynomial-time many-one reducible to B ($A \leq^{\mathrm{p}}_{\mathrm{m}} B$), if there exists a polynomial-time-computable function f such that ($x \in A \iff f(x) \in B$). We also use the following logspace reducibilities which are defined analogously in terms of logspace oracle Turing machines and logspace-computable functions: $\leq^{\log}_{\mathrm{T}}$, $\leq^{\log}_{\text{log-T}}$, $\leq^{\log}_{\mathrm{tt}}$, $\leq^{\log}_{k\text{-tt}}$, $\leq^{\log}_{\mathrm{btt}}$, $\leq^{\log}_{\mathrm{dtt}}$, $\leq^{\log}_{k\text{-dtt}}$, $\leq^{\log}_{\mathrm{bdtt}}$, $\leq^{\log}_{\mathrm{ctt}}$, $\leq^{\log}_{k\text{-ctt}}$, $\leq^{\log}_{\mathrm{bctt}}$, $\leq^{\log}_{\mathrm{m}}$.

We have already argued that in the polynomial-time setting, the number of oracle tapes is not significant. Similarly, one can easily see that all truth-table reductions can be performed by some oracle Turing machine that uses only one oracle tape. So, the number of oracle tapes used is only significant for logspace Turing reductions. We further define $A \leq^{\log[k]}_{\mathrm{T}} B$ if $A \leq^{\log}_{\mathrm{T}} B$ via some logspace oracle Turing machine with k oracle tapes, and furthermore $A \leq^{\log[k]}_{\text{log-T}} B$ if $A \leq^{\log[k]}_{\mathrm{T}} B$ with an oracle machine that on input x asks at most $O(\log(|x|))$ queries. By Ladner and Lynch [LL76] it holds that $A \leq^{\log[1]}_{\mathrm{T}} B \iff A \leq^{\log}_{\mathrm{tt}} B$.

Autoreducibility. A set A is called $\leq^{\mathrm{p}}_{\mathrm{T}}$-*autoreducible* if $A \leq^{\mathrm{p}}_{\mathrm{T}} A$ via some polynomial-time oracle Turing machine M with the restriction that M on input x never queries x. For the reductions $\leq^{\mathrm{p}}_{\text{log-T}}$, $\leq^{\mathrm{p}}_{\mathrm{tt}}$, $\leq^{\mathrm{p}}_{k\text{-tt}}$, $\leq^{\mathrm{p}}_{\mathrm{btt}}$, $\leq^{\log}_{\mathrm{T}}$, $\leq^{\log}_{\text{log-T}}$, $\leq^{\log[k]}_{\mathrm{T}}$, $\leq^{\log[k]}_{\text{log-T}}$, $\leq^{\log}_{\mathrm{tt}}$, $\leq^{\log}_{k\text{-tt}}$, $\leq^{\log}_{\mathrm{btt}}$, we define autoreducibility analogously.

A set A is called $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-*autoreducible* if $A \leq^{\mathrm{p}}_{\mathrm{dtt}} A$ via some $f \in \mathrm{FP}$ where from $f(x) = \langle y_1, y_2, \ldots, y_n \rangle$ it follows that $x \notin \{y_1, y_2, \ldots, y_n\}$ for all x. We define autoreducibility analogously for $\leq^{\mathrm{p}}_{k\text{-dtt}}$, $\leq^{\mathrm{p}}_{\mathrm{bdtt}}$, $\leq^{\mathrm{p}}_{\mathrm{ctt}}$, $\leq^{\mathrm{p}}_{k\text{-ctt}}$, $\leq^{\mathrm{p}}_{\mathrm{bctt}}$, $\leq^{\mathrm{p}}_{\mathrm{m}}$, $\leq^{\log}_{\mathrm{dtt}}$, $\leq^{\log}_{k\text{-dtt}}$, $\leq^{\log}_{\mathrm{bdtt}}$, $\leq^{\log}_{\mathrm{ctt}}$, $\leq^{\log}_{k\text{-ctt}}$, $\leq^{\log}_{\mathrm{bctt}}$, $\leq^{\log}_{\mathrm{m}}$.

Mitoticity and Weak Mitoticity. A set A is called *weakly* $\leq^{\mathrm{p}}_{\mathrm{T}}$-*mitotic* if $A \equiv^{\mathrm{p}}_{\mathrm{T}} A \cap S \equiv^{\mathrm{p}}_{\mathrm{T}} A \cap \overline{S}$ for some set S. We refer to S as a *separator*. If in addition it holds that $S \in \mathrm{P}$, we call $A \leq^{\mathrm{p}}_{\mathrm{T}}$-*mitotic*. For the reductions $\leq^{\mathrm{p}}_{\text{log-T}}$, $\leq^{\mathrm{p}}_{\mathrm{tt}}$, $\leq^{\mathrm{p}}_{k\text{-tt}}$, $\leq^{\mathrm{p}}_{\mathrm{btt}}$, $\leq^{\mathrm{p}}_{\mathrm{dtt}}$, $\leq^{\mathrm{p}}_{k\text{-dtt}}$, $\leq^{\mathrm{p}}_{\mathrm{bdtt}}$, $\leq^{\mathrm{p}}_{\mathrm{ctt}}$, $\leq^{\mathrm{p}}_{k\text{-ctt}}$, $\leq^{\mathrm{p}}_{\mathrm{bctt}}$, $\leq^{\mathrm{p}}_{\mathrm{m}}$, mitoticity and weak mitoticity are defined analogously, where, in the case of non-transitive reductions, we require that the sets A, $A \cap S$, and $A \cap \overline{S}$ are pairwise equivalent.

For the reductions $\leq^{\log}_{\mathrm{T}}$, $\leq^{\log[k]}_{\mathrm{T}}$, $\leq^{\log}_{\text{log-T}}$, $\leq^{\log[k]}_{\text{log-T}}$, $\leq^{\log}_{\mathrm{tt}}$, $\leq^{\log}_{k\text{-tt}}$, $\leq^{\log}_{\mathrm{btt}}$, $\leq^{\log}_{\mathrm{dtt}}$, $\leq^{\log}_{k\text{-dtt}}$, $\leq^{\log}_{\mathrm{bdtt}}$, $\leq^{\log}_{\mathrm{ctt}}$, $\leq^{\log}_{k\text{-ctt}}$, $\leq^{\log}_{\mathrm{bctt}}$, $\leq^{\log}_{\mathrm{m}}$, weak mitoticity is defined analogously to the polynomial-time setting, and for mitoticity we further require that the separator is logspace decidable.

3 Ruling Sets for Autoreductions

For transforming many-one autoreducibility into mitoticity, we consider the trace of words obtained by the repeated application of the autoreduction to some input x. Now the challenge is to define a set S of low complexity such that when we follow such a trace for r steps, then we visit at least one word in S and at least one word in $\overline{S}$. Cole and Vishkin [CV86] developed the deterministic coin tossing, which is a technique for the construction of such S. In their terminology, the set S is called an r-ruling set.

In this section we show that we can obtain a 2-ruling set S from autoreduction functions, where S is only slightly more complex than f. The proof of the following lemma is a generalization of a proof presented in [Gla10].

Lemma 3.1 *Let f be a polynomially length-bounded function such that $f(x) \neq x$ for all x. For all $k \geq 1$ there exists a set S, a constant c_0, and a polynomial q such that:*

1. *For all x there exists some $i \leq c_0 \cdot (\log^{(k)}(|x|) + 1)$ such that $x \in S \iff f^{(i)}(x) \notin S$, and for all $j \leq i$ it holds that $|f^{(j)}(x)| \leq q(|x|)$.*
2. *If $s(n) \geq \log(n)$ and $f \in \mathrm{FSPACE}(s)$, then $S \in \mathrm{DSPACE}(s)$.*
3. *If $t(n) \geq n$ and $f \in \mathrm{FTIME}(t)$, then $S \in \mathrm{DTIME}(O(t \circ q))$.*

Proof Let f be a function and $p(n) = n^c + c$ be a polynomial such that $f(x) \neq x$ and $|f(x)| \leq p(|x|)$ for all x. We assume that $c \geq 2$ is large enough such that $\log^{(k)}(p^{(3)}(n)) \geq 3$ holds for all n. According to this length bound we now define the tower function

$$l(i) := \begin{cases} 2 & \text{if } i = 0, \text{ and} \\ p(p(l(i-1))) & \text{otherwise.} \end{cases}$$

Note that for the inverse tower function $l^{-1}(n) := \min\{i \mid l(i) \geq n\}$ and for all n,

$$l^{-1}(p(p(n))) = l^{-1}(n) + 1. \tag{1}$$

So from f's length bound we obtain for all x,

$$l^{-1}(|f(x)|) \leq l^{-1}(|x|) + 1 \quad \text{and} \quad l^{-1}(|f(f(x))|) \leq l^{-1}(|x|) + 1. \tag{2}$$

We partition the set of all words as follows.

$$\begin{aligned} S_0 &:= \{x \mid l^{-1}(|x|) \equiv 0 \mod 2\} \\ S_1 &:= \{x \mid l^{-1}(|x|) \equiv 1 \mod 2\} \end{aligned}$$

We can decide S_0 and S_1 as follows. On input x, compute and store $n = |x|$. Then, starting with $m = 2$, determine how often we need to apply $p^{(2)}$ to m until we obtain a value larger than n. For this we only need to store values less than or equal to n, which is possible in space $\log(|x|)$. Hence $S_0, S_1 \in \mathrm{L}$.

We use the following distance function for integers.

$$d(x,y) := \begin{cases} 0 & \text{if } x = y, \text{ and} \\ \mathrm{sgn}(y-x) \cdot \lfloor \log(2\mathrm{abs}(y-x)) \rfloor & \text{otherwise.} \end{cases}$$

Note that $d \in \mathrm{FL}$, and furthermore, $d(x,y) = 0$ if and only if $x = y$.

Claim 3.2 *If z_1, z_2, and z_3 are integers such that $d(z_1, z_2) = d(z_2, z_3) \neq 0$, then there exist $i, j \in [1, 3]$ such that for $r \stackrel{df}{=} d(z_1, z_2)$,*

$$\lfloor z_i / 2^{\mathrm{abs}(r)} \rfloor \text{ is even } \Leftrightarrow \lfloor z_j / 2^{\mathrm{abs}(r)} \rfloor \text{ is odd.}$$

Proof Assume that the claim does not hold and let z_1, z_2, and z_3 be counter examples. Let $r := d(z_1, z_2) = d(z_2, z_3)$, $a_1 := \lfloor z_1 / 2^{\mathrm{abs}(r)} \rfloor$, $a_2 := \lfloor z_2 / 2^{\mathrm{abs}(r)} \rfloor$, and $a_3 := \lfloor z_3 / 2^{\mathrm{abs}(r)} \rfloor$. So by assumption, either a_1, a_2, and a_3 are all even or a_1, a_2, and a_3 are all odd. Without loss of generality let us assume that a_1, a_2, and a_3 are even (the other case is analogous).

Case 1: Assume $r > 0$. So $z_1 < z_2 < z_3$ and hence $a_1 \leq a_2 \leq a_3$.

Assume $a_1 = a_3$. From $d(z_1, z_2) = r$ it follows that $\log(2\mathrm{abs}(z_2 - z_1)) \geq r$ and hence $z_2 - z_1 \geq 2^{r-1}$. The same argument shows $z_3 - z_2 \geq 2^{r-1}$. So $z_3 \geq z_1 + 2^r = z_1 + 2^{\mathrm{abs}(r)}$ and hence $a_3 \geq a_1 + 1$ which contradicts the assumption $a_1 = a_3$.

So it holds that $a_1 < a_3$. This implies $a_3 - a_1 \geq 2$, since both values are even. Since a_2 is even as well, we obtain $a_2 - a_1 \geq 2$ or $a_3 - a_2 \geq 2$. If $a_2 - a_1 \geq 2$, then $z_2 - z_1 > 2^r$ and so $d(z_1, z_2) \geq r + 1$. If $a_3 - a_2 \geq 2$, then $z_3 - z_2 > 2^r$ and so $d(z_2, z_3) \geq r + 1$. Both conclusions contradict the definition of r.

Case 2: Assume $r < 0$. So $z_1 > z_2 > z_3$ and hence $a_1 \geq a_2 \geq a_3$.

Assume $a_1 = a_3$. From $d(z_1, z_2) = r$ it follows that $\log(2\mathrm{abs}(z_2 - z_1)) \geq \mathrm{abs}(r)$ and hence $z_1 - z_2 \geq 2^{\mathrm{abs}(r)-1}$. The same argument shows $z_2 - z_3 \geq 2^{\mathrm{abs}(r)-1}$. So $z_1 \geq z_3 + 2^{\mathrm{abs}(r)}$ and hence $a_1 \geq a_3 + 1$ which contradicts the assumption $a_1 = a_3$.

So it holds that $a_1 > a_3$. This implies $a_1 - a_3 \geq 2$, since both values are even. Since a_2 is even as well, we obtain $a_1 - a_2 \geq 2$ or $a_2 - a_3 \geq 2$. If $a_1 - a_2 \geq 2$, then $z_1 - z_2 > 2^{\mathrm{abs}(r)}$ and so $d(z_1, z_2) \leq -\mathrm{abs}(r) - 1 < r$. If $a_2 - a_3 \geq 2$, then $z_2 - z_3 > 2^{\mathrm{abs}(r)}$ and so $d(z_2, z_3) \leq -\mathrm{abs}(r) - 1 < r$. Both conclusions contradict the definition of r. This proves Claim 3.2. □

We will iteratively apply the distance function to some value x and its successors in f. We define

$$d_i(x) := \begin{cases} d(d_{i-1}(x), d_{i-1}(f(x))) & \text{if } i > 0, \text{ and} \\ x & \text{if } i = 0, \end{cases}$$

for all x and all $i \geq 0$. Next, we define S to be the set decided by the algorithm described in Figure 1.

```
On input x:
 1. let y := f^(1)(x) and z := f^(2)(x)
 2. if |x| < |y| and c_{S_0}(x) = c_{S_1}(y) then accept
 3. if |y| < |z| and c_{S_0}(y) = c_{S_1}(z) then reject
 4. if d_{k+1}(x) > d_{k+1}(y) then accept
 5. if d_{k+1}(x) < d_{k+1}(y) then reject
 6. for j := k + 1 downto 1 do
 7.   if d_j(x) ≠ 0 then accept ⟺ ⌊d_{j-1}(x)/2^{abs(d_j(x))}⌋ is even
 8. reject
```

Figure 1: Algorithm for the set S.

Claim 3.3 *If $t(n) \geq n$ and $f \in \mathrm{FTIME}(t)$, then $S \in \mathrm{DTIME}(O(t(q(n))))$ for a polynomial q.*

Proof Let $t(n) \geq n$ such that $f \in \mathrm{FTIME}(t)$. On input x, let $n = |x|$. We can proceed by computing and storing $f^{(1)}(x), f^{(2)}(x), \ldots, f^{(k+2)}(x)$. Since k is constant, this takes time

$O((k+2) \cdot t(q'(n))) = O(t(q'(n)))$, where q' is a polynomial such that $q'(n)$ bounds the length of each $f^{(i)}(x)$ for $i \leq k+2$.

Next, consider lines 2 and 3. Those lines can be executed in time $O(q''(n))$ for some polynomial q'', because x, y, z are stored on some tape, are polynomially long, and $S_0, S_1 \in \mathrm{L} \subseteq \mathrm{P}$.

Observe that for each $i \leq k+1$, $d_i(x)$ (and $d_i(y)$) can be computed by iteratively applying $d \in \mathrm{FL} \subseteq \mathrm{FP}$ at most $(k+1)^2$ times. This means that lines 4 and 5 can be executed in time $O((k+1)^2 \cdot q'''(n)) = O(q'''(n))$ for some polynomial q'''. Moreover, the remaining loop is iterated at most $k+1$ times, and again, the values that are computed inside the loop can be computed in time $O(q'''(n))$, hence the entire loop can be executed in time $O((k+1) \cdot q'''(n)) = O(q'''(n))$.

Overall, the set S can be decided in time $O(t(q(n)))$ for some polynomial q. □

Claim 3.4 *If $s(n) \geq \log(n)$ and $f \in \mathrm{FSPACE}(s)$, then $S \in \mathrm{DSPACE}(s)$.*

Proof Let $s(n) \geq \log(n)$ such that $f \in \mathrm{FSPACE}(s)$. First, observe that for all $i \leq k+2$ we have $f^{(i)}(x) \in \mathrm{FSPACE}(s)$, which is shown by induction on i using the facts that $s \geq \log$ and that f is polynomially length-bounded. By the polynomial length bound of $f^{(i)}$, and by $f^{(i)} \in \mathrm{FSPACE}(s)$ and $d \in \mathrm{FL}$, we also obtain $d_i \in \mathrm{FSPACE}(s)$ for all $i \leq k+1$. Hence, all variables and functions can be computed in $\mathrm{FSPACE}(s)$. Also recall that $S_0, S_1 \in \mathrm{L}$, so we can decide $x, y, z \in S_0, S_1$ in $\mathrm{DSPACE}(s)$. □

By the above two claims, item 2 and item 3 of the lemma hold (for some polynomial q). We now continue to prove the first item as well. First, choose some constant c' large enough such that

$$2 \cdot \log^{(k-1)}(3c \cdot \log(n+3c)) \leq c' \cdot (\log^{(k)}(n) + 1) \tag{3}$$

holds for all n. We further define h as the function computed by the algorithm described in Figure 2.

```
On input x:
 1. n := |x|, m := 6c' · ⌈log^(k)(n) + 1⌉ + k + 7
 2. for i := 1 to m
 3.    // here |f^(i)(x)| ≤ p^(3)(n)
 4.    if x ∈ S ⇔ f^(i)(x) ∉ S then return f^(i)(x)
 5. next i
 6. // this line is never reached
```

Figure 2: Algorithm for the function h.

Claim 3.5 *In the algorithm for h, the invariant in line 3 holds.*

Proof Assume there exists an $i \in [1, m]$ such that the algorithm reaches the i-th iteration of the loop and there it holds that $|f^{(i)}(x)| > p^{(3)}(n)$. Choose the smallest such i and let $r = l^{-1}(n)$. Note that $i \geq 4$, since f's length is bounded by $p(n)$.

Observe that by (1), $l^{-1}(|f^{(i)}(x)|) \geq l^{-1}(p(p(n))) = r+1$. From (1) we also obtain that either

$$l^{-1}(p(n)) = r \quad \text{and} \quad l^{-1}(p(p(n))) = r+1 \tag{4}$$

or

$$l^{-1}(p(n)) = l^{-1}(p(p(n))) = r+1 \quad \text{and} \quad l^{-1}(p^{(3)}(n)) = r+2. \tag{5}$$

Recall that by (2), if $j_0 < j_2$ such that $l^{-1}(|f^{(j_0)}(x)|) = r$ and $l^{-1}(|f^{(j_2)}(x)|) = r+2$, then there exists $j_1 \in (j_0, j_2 - 2]$ such that $l^{-1}(|f^{(j_1)}(x)|) = r+1$ and $l^{-1}(|f^{(j_1+1)}(x)|) = r+1$. If (4) holds, then $l^{-1}(|f(x)|) \leq r$ and so there exists $j \in [2, i]$ such that for $u = f^{(j-2)}(x)$, $v = f^{(j-1)}(x)$ and $w = f^{(j)}(x)$ it holds that $l^{-1}(|u|) = l^{-1}(|v|) = r$ and $l^{-1}(|w|) = r+1$. If (5) holds, then there exists $j \in [2, i]$ such that for $u = f^{(j-2)}(x)$, $v = f^{(j-1)}(x)$ and $w = f^{(j)}(x)$ it holds that $l^{-1}(|u|) = l^{-1}(|v|) = r+1$ and $l^{-1}(|w|) = r+2$. In both cases we have $(u \in S_0 \Leftrightarrow v \in S_0)$ and $(v \in S_0 \Leftrightarrow w \in S_1)$. If we consider the algorithm for S, then we see that $u \notin S$ and $v \in S$. Therefore, in the algorithm for h, the condition in line 4 is either satisfied for $i = j-2$ or is satisfied for $i = j-1$. This contradicts our assumption that we reach the i-th iteration of the loop and proves the claim. □

Claim 3.6 *The algorithm for h never reaches line 6.*

Proof Assume that for some input x, the algorithm reaches line 6. Let $n = |x|$ and $m = 6c' \cdot \lceil \log^{(k)}(n) + 1 \rceil + k + 7$, and $x_i = f^{(i)}(x)$ for $i \geq 0$. Hence, for all $i \in [1, m]$ it holds that $x \in S \Leftrightarrow x_i \in S$. Without loss of generality let us assume that $x_i \in S$ for all $i \in [0, m]$.

All remaining arguments refer to the algorithm for S. For $i \in [1, m]$ it holds that the algorithm on input x_i does not stop in line 2, since otherwise the algorithm on input x_{i-1} stops in line 3 which contradicts the assumption $x_{i-1} \in S$. (Here one has to note that if the algorithm on input x_i stops in line 2, then by (2), on input x_{i-1} it cannot stop in line 2 as well.) So for all $i \in [1, m]$, the algorithm on input x_i reaches line 4.

For $i \geq 1$, let y_i and z_i denote the values of the program variables y and z when the algorithm for S works on input x_i. We show that there are not too many elements $i \in [1, m]$ such that the algorithm on input x_i stops in line 4.

By Claim 3.5, for $i \in [1, m]$, $|x_i| \leq p^{(3)}(n)$. First, we inductively show that for all $j \in [0, k]$ and all $i \in [1, m-j-1]$ it holds that $\mathrm{abs}(d_{j+1}(x_i)) \leq 2 \cdot \log^{(j)}(p^{(3)}(n))$.

- Let $j = 0$ and $i \in [1, m-1]$. For $\mathrm{abs}(d_{j+1}(x_i))$ we obtain:

$$\begin{aligned}\mathrm{abs}(d_1(x_i)) &= \mathrm{abs}(d(d_0(x_i), d_0(x_{i+1}))) = \lfloor \log(2 \cdot \mathrm{abs}(d_0(x_{i+1}) - d_0(x_i)) \rfloor \\ &\leq \log(2 \cdot 2^{p^{(3)}(n)}) = 1 + p^{(3)}(n) \leq 2 \cdot p^{(3)}(n) = 2 \cdot \log^{(0)}(p^{(3)}(n))\end{aligned}$$

- Let $0 < j \leq k$ and suppose the inequality holds for $j-1$ and all $i \in [1, m-(j-1)-1] = [1, m-j]$. For $i \in [1, m-j-1]$ we obtain:

$$\begin{aligned}\mathrm{abs}(d_{j+1}(x_i)) &= \mathrm{abs}(d(d_j(x_i), d_j(x_{i+1}))) = \lfloor \log(2 \cdot \mathrm{abs}(d_j(x_{i+1}) - d_j(x_i)) \rfloor \\ &\leq \log(2 \cdot (\mathrm{abs}(d_j(x_{i+1})) + \mathrm{abs}(d_j(x_i)))) \\ &\leq \log(2 \cdot 2 \cdot (2 \cdot \log^{(j-1)}(p^{(3)}(n)))) \\ &\leq 3 + \log(\cdot \log^{(j-1)}(p^{(3)}(n))) \\ &\leq 2 \cdot \log^{(j)}(p^{(3)}(n))\end{aligned}$$

  Note that the last step holds because we have chosen c (and hence p) large enough such that it holds that $\log^{(k)}(p^{(3)}(n)) \geq 3$ for all n.

We obtain that for all $i \in [1, m-k-1]$ it holds that

$$\begin{aligned}\mathrm{abs}(d_{k+1}(x_i)) &\leq 2 \cdot \log^{(k)}(p^{(3)}(n)) = 2 \cdot \log^{(k)}(((n^c + c)^c + c)^c + c) \\ &\leq 2 \cdot \log^{(k-1)}(\log((n+3c)^{3c})) = 2 \cdot \log^{(k-1)}(3c \cdot \log(n+3c)) \\ &\leq c' \cdot (\log^{(k)}(n) + 1),\end{aligned}$$

where the last step holds because of inequation (3).

We now consider the sequence of $d_{k+1}(x_i)$ for $i \in [1, m-k-4]$. This sequence is not increasing, since otherwise we stop in line 5 which contradicts the assumption $x_i \in S$. We have seen that the values in this sequence are integers in $[-c' \cdot (\log^{(k)}(n)+1), c' \cdot (\log^{(k)}(n)+1)]$. So the number of positions where the sequence decreases is at most

$$2c' \cdot (\log^{(k)}(n)+1) \leq \frac{m-k-7}{3}.$$

This shows that the number of $i \in [1, m-k-4]$ such that the algorithm on input x_i stops in line 4 is at most $(m-k-7)/3 = (m-k-4)/3 - 1$. By a pigeon hole argument, there exists an $i \in [1, m-k-4]$ such that the algorithm reaches the loop in line 6 for the inputs x_i, x_{i+1}, and x_{i+2}. We finish the proof of the claim with the following case distinction:

Case 1: On some $u \in \{x_i, x_{i+1}, x_{i+2}\}$, the algorithm terminates after less than $k+1$ iterations. Choose $v \in \{x_i, x_{i+1}, x_{i+2}\}$ such that the algorithm on input v terminates after $k_0 < k+1$ iterations, and for each $w \in \{x_i, x_{i+1}, x_{i+2}\}$, the algorithm on input w does not terminate after less than k_0 iterations. Let j_0 denote the value of the variable j of the algorithm on input v in the iteration where the algorithm terminates. Hence we have $d_{j_0}(v) \neq 0$.

Next we will show that for all $w \in \{x_i, x_{i+1}, x_{i+2}\}$ we have $d_{j_0+1}(w) = 0$. If $j_0 = k+1$ we have $d_{j_0+1}(w) = d_{k+2}(w) = d(d_{k+1}(w), d_{k+1}(f(w))) = 0$, because for w we reach the loop and do not terminate in line 4 or line 5, which is only possible if $d_{k+1}(w) = d_{k+1}(f(w))$. If $j_0 < k+1$ we have $d_{j_0+1}(w) = 0$, because, by the choice of v, the algorithm on input w does not terminate after less than k_0 iterations. So for all $w \in \{x_i, x_{i+1}, x_{i+2}\}$ we have $d_{j_0+1}(w) = 0$.

This means $d_{j_0}(x_i) = d_{j_0}(x_{i+1}) = d_{j_0}(x_{i+2})$. Together with $d_{j_0}(v) \neq 0$ we obtain $d_{j_0}(x_i) = d_{j_0}(x_{i+1}) = d_{j_0}(x_{i+2}) \neq 0$, hence $d(d_{j_0-1}(x_i), d_{j_0-1}(x_{i+1})) = d(d_{j_0-1}(x_{i+1}), d_{j_0-1}(x_{i+2})) \neq 0$.

By Claim 3.2 it follows that there are $i_1, i_2 \in [i, i+2]$ such that

$$\lfloor d_{j_0-1}(x_{i_1}) / 2^{\mathrm{abs}(d_{j_0}(x_{i_1}))} \rfloor \text{ is even} \iff \lfloor d_{j_0-1}(x_{i_2}) / 2^{\mathrm{abs}(d_{j_0}(x_{i_2}))} \rfloor \text{ is odd.}$$

Hence the algorithm accepts x_{i_1} if and only if it rejects x_{i_2}. This contradicts $x_{i_1}, x_{i_2} \in S$.

Case 2: On each $u \in \{x_i, x_{i+1}, x_{i+2}\}$, the algorithm reaches the $(k+1)$-st iteration. Recall that $k \geq 1$. Since the algorithm does not stop in the k-th iteration we have $d_2(x_i) = d_2(x_{i+1}) = d_2(x_{i+2}) = 0$, which means that $d_1(x_i) = d_1(x_{i+1}) = d_1(x_{i+2}) = d(x_i, x_{i+1})$. Since f is an autoreduction we have $x_i \neq x_{i+1}$, hence $d_1(x_i) = d_1(x_{i+1}) = d_1(x_{i+2}) \neq 0$. By Claim 3.2 it follows that there are $i_1, i_2 \in [i, i+2]$ where

$$\lfloor x_{i_1} / 2^{\mathrm{abs}(d_1(x_{i_1}))} \rfloor \text{ is even} \iff \lfloor x_{i_2} / 2^{\mathrm{abs}(d_1(x_{i_2}))} \rfloor \text{ is odd.}$$

Hence the algorithm accepts x_{i_1} if and only if it rejects x_{i_2}. This contradicts $x_{i_1}, x_{i_2} \in S$.

In each case we obtain a contradiction, so our assumption was wrong, and the claim holds. □

Given the last two claims, we can now finish our proof of item 1 as well. By Claim 3.6, there must exist some iteration with $i \leq m = 6c' \cdot \lceil \log^{(k)}(n)+1 \rceil + k + 7 \leq (12c' + k + 7) \cdot (\log^{(k)}(n)+1)$ where the algorithm for h on input x stops, hence $x \in S \iff f^{(i)}(x) \notin S$. By Claim 3.5 it holds that $|f^{(j)}(x)| \leq p^{(3)}(|x|)$ for all $j \leq i$. Hence also item 1 of the lemma holds. Finally, note that we have shown item 1 and item 3 for different polynomials. Clearly we can choose a single polynomial q for which both items hold. □

Lemma 3.7 *Let f be a polynomially length-bounded function such that $f(x) \neq x$ for all x. For every $k \geq 1$ there exist a set S and a function g with the following properties:*

1. $g(x) \in S \iff f(g(x)) \notin S$
2. $g(x) \in \{x, f(x)\}$
3. *If $s(n) \geq \log(n)$ and $f \in \mathrm{FSPACE}(s)$, then $S \in \mathrm{DSPACE}(s \cdot \log^{(k)})$ and $g \in \mathrm{FSPACE}(s)$.*
4. *If $t(n) \geq n$ and $f \in \mathrm{FTIME}(t)$, then $S \in \mathrm{DTIME}(O(t \circ q))$ and $g \in \mathrm{FTIME}(O(t \circ q))$, where q is some polynomial.*

Proof Let f be a function and p be a polynomial with $f(x) \neq x$ and $|f(x)| \leq p(|x|)$ for all x. By Lemma 3.1 there exist a set S', a polynomial p', and some constant c such that for all x there exists some $i \leq c \cdot (1 + \log^{(k)}(|x|))$ with $x \in S' \iff f^{(i)}(x) \notin S'$ and $|f^{(j)}(x)| \leq p'(|x|)$ for all $j \leq i$. We define S to be the set decided by the algorithm described in Figure 3.

```
On input x:
 1. y := x, i := 0
 2. while (y ∉ S′ or f(y) ∈ S′):
 3.    y := f(y), i := i + 1
 4. accept ⟺ i is even
```

Figure 3: Algorithm for the set S.

Furthermore, we define the function g by

$$g(x) := \begin{cases} f(x) & \text{if } x \in S' \text{ and } f(x) \notin S' \\ x & \text{otherwise} \end{cases}$$

for all x. Note that by this definition, item 2 of the lemma holds for g.

We will next show that item 1 of the lemma holds. Let x be some arbitrary input. Note that after at most $O(\log^{(k)}(|x|))$ iterations of f, the membership to S' changes. Choose the minimal l such that $f^{(l)}(x) \in S'$ and $f^{(l+1)}(x) \notin S'$. Note that $l \in O(\log^{(k)}(|x|))$ and $|f^{(l')}(x)| \leq p'(p'(|x|))$ for all $l' \leq l$. By the choice of l, for all $l' < l$ it holds that $f^{(l')}(x) \notin S'$ or $f^{(l'+1)}(x) \in S'$. This means that the algorithm stops after exactly l iterations. We distinguish the following cases:

Case 1: $l > 0$. Then, for $l' = l - 1$ it holds that $f^{(l')}(f(x)) \in S'$ and $f^{(l'+1)}(f(x)) \notin S'$, and l' is minimal with this property. Hence, on input $f(x)$, the algorithm for S stops after $l - 1$ iterations, and we obtain $g(x) \in S \iff x \in S \iff l$ is even $\iff l - 1$ is odd $\iff f(x) \notin S \iff f(g(x)) \notin S$.

Case 2: $l = 0$. Hence, $g(x) = f(x)$, and $x \in S'$ and $f(x) \notin S'$. So, for the smallest $l' \geq 0$ with $f^{(l')}(f(x)) \in S'$ and $f^{(l'+1)}(f(x)) \notin S'$ it holds that $l' > 0$, and the argumentation of the first case shows that $g(x) \in S \iff f(x) \in S \iff f(f(x)) \notin S \iff f(g(x)) \notin S$.

In each case we obtain $g(x) \in S \iff f(g(x)) \notin S$, so item 1 of the lemma holds for g.

It remains to argue for the running time and the space requirements of the algorithm. We have the following claims.

Claim 3.8 *If $f \in \mathrm{FTIME}(t)$ for some t with $t(n) \geq n$, then $S \in \mathrm{DTIME}(O(t(q(n))))$ and $g \in \mathrm{FTIME}(O(t(q(n))))$, where q is some polynomial.*

Proof Suppose that $t(n) \geq n$ and $f \in \mathrm{FTIME}(t)$. By Lemma 3.1 we have $S' \in \mathrm{DTIME}(O(t(q'(n))))$ for some polynomial q'. Then, $g \in \mathrm{FTIME}(O(t(q''(n))))$ for some polynomial q''. Next, we consider the algorithm for S. There exists a constant c' such that we iterate at most

$$l \leq c \cdot (1 + \log^{(k)}(|x|)) + c \cdot (1 + \log^{(k)}(p'(p'(|x|)))) \leq c' \cdot (1 + \log^{(k)}(|x|))$$

times. In each iteration, we compute f and decide S' on inputs of length at most $p'(p'(|x|))$, which is possible in time $O(t(p'(p'(|x|))))$ and $O(t(q'(p'(p'(|x|)))))$, respectively. Hence there exists a polynomial q'' such that the entire loop can be executed in time $O(c' \cdot (1 + \log^{(k)}(n)) \cdot t(q''(n)))$, and thus there exists a polynomial q such that the entire algorithm can be executed in time $O(t(q(n)))$. □

Claim 3.9 *If $s(n) \geq \log(n)$ and $f \in \mathrm{FSPACE}(s)$, then $S \in \mathrm{DSPACE}(s \cdot \log^{(k)})$ and $g \in \mathrm{FSPACE}(s)$.*

Proof Suppose that $s(n) \geq \log(n)$ and $f \in \mathrm{FSPACE}(s)$. By Lemma 3.1 we have $S' \in \mathrm{DSPACE}(s)$, hence $g \in \mathrm{FSPACE}(s)$ (here we need to recompute each bit of the function f on input x, which is possible since $|f(x)| \leq p(|x|)$ and $s(n) \geq \log(n)$).

Next, consider the algorithm for S. We have already argued that we iterate at most $c' \cdot (1 + \log^{(k)}(|x|))$ times, and that in each iteration, the length of y is bounded by $p'(p'(|x|))$, and hence in each iteration, the single bits of y can be addressed by $O(\log(|x|))$ many bits. So by a blockwise recomputation we obtain $S \in \mathrm{DSPACE}(O(s(n) \cdot c' \cdot (1 + \log^{(k)}(n)))) = \mathrm{DSPACE}(s(n) \cdot \log^{(k)}(n))$. This shows the claim. □

□

4 Weak Mitoticity

We show how autoreducibility of complete sets for general classes implies weak mitoticity. This gives progress towards the general question of whether complete sets are mitotic.

General Approach and Results. Given a many-one autoreduction f for some set A complete for some class $\mathcal{C}$, we apply the results of the previous section to generate a 2-ruling set S for f. Since the complexity of S is only slightly higher than the complexity of f, we obtain $A \cap S \in \mathcal{C}$ and $A \cap \overline{S} \in \mathcal{C}$. Considering some input x, we then find two elements y, z on f's trace on x with the same membership to A as x such that exactly one is contained in S. Hence, y, z are a 2-dtt-reduction from A to $A \cap S$ and $A \cap \overline{S}$. So A is many-one complete for $\mathcal{C}$, and $A \cap S$ and $A \cap \overline{S}$ are 2-dtt complete, which shows weak 2-dtt mitoticity of A.

The approach can be generalized to further reducibility notions, including disjunctive truth-table reductions and truth-table reductions with exactly one query.

Particular Results. Together with previously known autoreducibility results for complete sets, we obtain the following mitoticity results:

1. Every $\leq^{\log}_{\mathrm{m}}$-complete set and every $\leq^{\log}_{1\text{-tt}}$-complete set for NEXP is weakly $\leq^{\log}_{2\text{-dtt}}$-mitotic.
2. Every $\leq^{\log}_{\mathrm{m}}$-complete set for Δ^{p}_k (and for P in particular) is weakly $\leq^{\log}_{2\text{-tt}}$-mitotic.
3. Every $\leq^{\mathrm{p}}_{k\text{-dtt}}$-complete set for PSPACE is weakly $\leq^{\mathrm{p}}_{k(k^2+k+1)\text{-dtt}}$-mitotic.
4. Every $\leq^{\mathrm{p}}_{\mathrm{bdtt}}$-complete set for PSPACE is weakly $\leq^{\mathrm{p}}_{\mathrm{bdtt}}$-mitotic.
5. Every $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-complete set for PSPACE is weakly $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-mitotic.

4.1 Many-One Complete Sets

We first consider logspace many-one autoreducible, complete sets for classes that contain the intersection of P and some space class slightly higher than L. Since the classes will be closed under intersection, the intersection of the complete set with the separator and its complement remain in the same complexity class.

Theorem 4.1 *Let $\mathcal{C} \supseteq (\mathrm{DSPACE}(\log \cdot \log^{(c)}) \cap \mathrm{P})$ for some $c \geq 1$ be some complexity class that is closed under intersection. If A is $\leq^{\log}_{\mathrm{m}}$-complete for $\mathcal{C}$ and $\leq^{\log}_{\mathrm{m}}$-autoreducible, then A is weakly $\leq^{\log}_{\text{2-dtt}}$-mitotic.*

Proof Let $f \in \mathrm{FL}$ be a $\leq^{\log}_{\mathrm{m}}$-autoreduction for A. From Lemma 3.7 we obtain a set $S \in (\mathrm{DSPACE}(\log \cdot \log^{(c)}) \cap \mathrm{P})$ and a function $g \in \mathrm{FL}$ such that

$$g(x) \in S \iff f(g(x)) \notin S \tag{6}$$
$$g(x) \in \{x, f(x)\} \tag{7}$$

for all x. We will show that $A \cap S$ and $A \cap \overline{S}$ are $\leq^{\log}_{\text{2-dtt}}$-complete for $\mathcal{C}$.

Note that $(\mathrm{DSPACE}(\log \cdot \log^{(c)}) \cap \mathrm{P})$ is closed under complementation, hence it holds that $S, \overline{S} \in \mathcal{C}$. Furthermore, since $\mathcal{C}$ is closed under intersection, we obtain $A \cap S \in \mathcal{C}$ and $A \cap \overline{S} \in \mathcal{C}$. So it remains to argue for the $\leq^{\log}_{\text{2-dtt}}$-hardness of $A \cap S$ and $A \cap \overline{S}$ for $\mathcal{C}$. Since A is $\leq^{\log}_{\mathrm{m}}$-hard for $\mathcal{C}$, it remains to show $A \leq^{\log}_{\text{2-dtt}} A \cap S$ and $A \leq^{\log}_{\text{2-dtt}} A \cap \overline{S}$.

Observe that $c_A(x) = c_A(g(x)) = c_A(f(g(x)))$ and $\{g(x), f(g(x))\} \cap S \neq \emptyset$. Let $h(x) := \{g(x), f(g(x))\}$. If $x \in A$, then $h(x) \subseteq A$, hence $h(x) \cap (S \cap A) = (h(x) \cap A) \cap S = h(x) \cap S \neq \emptyset$. If $x \notin A$, then $h(x) \cap (A \cap S) \subseteq h(x) \cap A = \emptyset$. Hence, h shows that $A \leq^{\log}_{\text{2-dtt}} A \cap S$. Analogously, h shows that $A \leq^{\log}_{\text{2-dtt}} A \cap \overline{S}$. We hence obtain that A is $\leq^{\log}_{\mathrm{m}}$-complete and that $A \cap S$ and $A \cap \overline{S}$ are $\leq^{\log}_{\text{2-dtt}}$-complete for $\mathcal{C}$. Therefore, A is weakly $\leq^{\log}_{\text{2-dtt}}$-mitotic. □

Corollary 4.2 *1. Every $\leq^{\log}_{\mathrm{m}}$-complete set for* NEXP *is weakly $\leq^{\log}_{\text{2-dtt}}$-mitotic.*
2. Every $\leq^{\log}_{\text{1-tt}}$-complete set for NEXP *is weakly $\leq^{\log}_{\text{2-dtt}}$-mitotic.*

Proof We start with the first item, so let A be $\leq^{\log}_{\mathrm{m}}$-complete for NEXP. Glaßer et al. [GNR+13] show that A is $\leq^{\log}_{\mathrm{m}}$-autoreducible. $\mathrm{P} \subseteq \mathrm{NEXP}$ and NEXP is closed under intersection, hence we can apply Theorem 4.1 and obtain that A is weakly $\leq^{\log}_{\text{2-dtt}}$-mitotic.

To also show the second item, note that every $\leq^{\log}_{\text{1-tt}}$-complete set for NEXP is also $\leq^{\log}_{\mathrm{m}}$-complete for NEXP. By the previous item we obtain weak $\leq^{\log}_{\text{2-dtt}}$-mitoticity. □

4.2 Truth-Table Complete Sets with One Query

Our approach for many-one autoreductions can be generalized to truth-table autoreductions that ask exactly one query. We can think of such a truth-table autoreduction for some set A as two functions f, f', where f' maps to the set of all unary Boolean functions, such that for all x it holds that $f(x) \neq x$ and $c_A(x) = f'(x)(c_A(f(x)))$. If A is non-trivial, we can modify f' such that it never maps to a constant unary Boolean function. We will further modify the autoreduction such that on each input it either has a long part in its trace that behaves like a many-one autoreduction, or ends up after a few steps in a small cycle. Those cycles can be treated directly, and on the long parts of the trace that behave like a many-one autoreduction we can proceed similar to the many-one case.

Theorem 4.3 *Let $\mathcal{C} \supseteq (\mathrm{DSPACE}(\log \cdot \log^{(c)}) \cap \mathrm{P})$ for some $c \geq 1$ be some complexity class that is closed under intersection. If A is $\leq^{\log}_{1\text{-tt}}$-complete for $\mathcal{C}$ and $\leq^{\log}_{1\text{-tt}}$-autoreducible, then A is weakly $\leq^{\log}_{2\text{-tt}}$-mitotic.*

Proof Since A is $\leq^{\log}_{1\text{-tt}}$-autoreducible, there are functions $f, f' \in \mathrm{FL}$ such that f' maps to the set of all unary Boolean functions, and for all x it holds that $x \neq f(x)$ and $c_A(x) = f'(x)(c_A(f(x)))$. Note that we can assume that A is non-trivial (otherwise the result trivially holds), and hence that further $f'(x) \in \{\mathrm{id}, \mathrm{not}\}$ for all x. Given the autoreduction f, we define a function g as follows. For each x,

$$g(x) := f^{(k)}(x) \quad \text{for the smallest } k \leq 3 \text{ with } x \neq f^{(k)}(x) \text{ and } c_A(x) = c_A(f^{(k)}(x)), \text{ if such a } k \text{ exists, and} \tag{8}$$

$$g(x) := f(x) \quad \text{otherwise.} \tag{9}$$

Observe that $g \in \mathrm{FL}$ (because we can decide $c_A(x) = c_A(f^{(k)}(x))$ for $k \leq 3$ by looking at f and f') and $g(x) \neq x$ for all x. Furthermore, for every x we have the following observation:

$$c_A(x) = c_A(g(x)) \iff g(x) \text{ is defined according to (8)} \tag{10}$$

Claim 4.4 *For every x, at least one of the following holds:*

- $g^{(2)}(x) = g^{(4)}(x)$
- $c_A(g^{(k)}(x)) = c_A(g^{(k+1)}(x)) = c_A(g^{(k+2)}(x))$, *for some $k \leq 2$*

Proof Consider some x, and for each i, denote $x_i := g^{(i)}(x)$. Suppose for x, the claim does not hold, hence $x_2 \neq x_4$ (and hence $x_1 \neq x_3$ and $x_0 \neq x_2$), and

$$\text{for all } k \leq 2 \text{ it holds that } c_A(x_k) = c_A(x_{k+1}) \implies c_A(x_{k+1}) \neq c_A(x_{k+2}). \tag{11}$$

We distinguish the following cases:

Case 1: $c_A(x_k) \neq c_A(x_{k+1})$ and $c_A(x_{k+1}) \neq c_A(x_{k+2})$ for some $k \leq 2$. This means that $c_A(x_k) \neq c_A(g(x_k))$ and $c_A(x_{k+1}) \neq c_A(g(x_{k+1}))$. By (10), this means that $g(x_k)$ and $g(x_{k+1})$ are defined according to (9). Hence we have $g(x_k) = f(x_k)$ and $g(x_{k+1}) = f(x_{k+1})$. This means that $x_{k+2} = g(x_{k+1}) = f(x_{k+1}) = f(g(x_k)) = f(f(x_k)) = f^{(2)}(x_k)$.

We now argue that $x_{k+2} = x_k$. Suppose this is not the case, hence $x_{k+2} \neq x_k$. By our hypothesis, $c_A(x_{k+2}) = c_A(x_k)$. So $g(x_k)$ is defined according to (8), hence $c_A(x_k) = c_A(g(x_k))$. This contradicts $c_A(x_k) \neq c_A(g(x_k))$, so we have $x_{k+2} = x_k$.

Since $k \leq 2$, this contradicts either $x_0 \neq x_2$, $x_1 \neq x_3$, or $x_2 \neq x_4$. Hence this case cannot occur.

Case 2: $c_A(x_k) = c_A(x_{k+1})$ or $c_A(x_{k+1}) = c_A(x_{k+2})$ for all $k \leq 2$. If $c_A(x_0) \neq c_A(x_1)$, then $c_A(x_1) = c_A(x_2)$, hence $c_A(x_2) \neq c_A(x_3)$ by (11). If $c_A(x_0) = c_A(x_1)$, then $c_A(x_1) \neq c_A(x_2)$ by (11), hence $c_A(x_2) = c_A(x_3)$, and hence $c_A(x_3) \neq c_A(x_4)$ by (11). So in either case there exists some $k \leq 1$ such that $c_A(x_k) \neq c_A(x_{k+1})$, $c_A(x_{k+1}) = c_A(x_{k+2})$, and $c_A(x_{k+2}) \neq c_A(x_{k+3})$.

Then, $g(x_{k+1})$ is defined according to (8), and $g(x_k)$ and $g(x_{k+2})$ are defined according to (9). Hence we have $g(x_k) = f(x_k)$ and $g(x_{k+2}) = f(x_{k+2})$.

We first argue that $g(x_{k+1}) = f(x_{k+1})$ holds. So suppose that $g(x_{k+1}) \neq f(x_{k+1})$, we show that this implies a contradiction. Since $g(x_{k+1})$ is defined according to (8), but $g(x_{k+1}) \neq f(x_{k+1})$ holds, we know that $c_A(x_{k+1}) \neq c_A(f(x_{k+1}))$. Recall that $c_A(x_k) \neq c_A(x_{k+1})$, hence we obtain $c_A(x_k) = c_A(f(x_{k+1})) = c_A(f(g(x_k))) = c_A(f(f(x_k))) = c_A(f^{(2)}(x_k))$. We now

argue that both $x_k \neq f^{(2)}(x_k)$ and $x_k = f^{(2)}(x_k)$ lead to a contradiction. If $x_k \neq f^{(2)}(x_k)$, then $g(x_k)$ is defined according to (8), which contradicts $c_A(x_k) \neq c_A(x_{k+1})$. If $x_k = f^{(2)}(x_k)$, then $x_k = x_{k+2}$, which contradicts either $x_0 \neq x_2$ or $x_1 \neq x_3$. So in both cases we obtain a contradiction, hence $g(x_{k+1}) = f(x_{k+1})$. This means that $x_{k+3} = g^{(3)}(x_k) = f^{(3)}(x_k)$.

We now argue that $x_{k+3} = x_k$. Suppose this is not the case, hence $x_{k+3} \neq x_k$. From $c_A(x_k) \neq c_A(x_{k+1}) = c_A(x_{k+2}) \neq c_A(x_{k+3})$ we obtain $c_A(x_k) = c_A(x_{k+3})$. So $g(x_k)$ is defined according to (8). But this contradicts $c_A(x_k) \neq c_A(x_{k+1})$, hence $x_{k+3} = x_k$.

This means that $f^{(2)}(x_{k+2}) = f(f(x_{k+2})) = f(g(x_{k+2})) = f(x_{k+3}) = f(x_k) = x_{k+1}$. Recall that $x_{k+2} = g(x_{k+1}) = f(x_{k+1})$. Since f is an autoreduction, we have $x_{k+1} \neq x_{k+2}$. Furthermore, we have $c_A(x_{k+1}) = c_A(x_{k+2})$. So $g(x_{k+2})$ is defined according to (8). This contradicts $c_A(x_{k+2}) \neq c_A(g(x_{k+2}))$. Hence this case cannot occur.

So in each case we obtain a contradiction. Hence the claim must hold. □

From Lemma 3.7 we obtain a set $S \in (\mathrm{DSPACE}(\log \cdot \log^{(c)}) \cap \mathrm{P})$ and a function $h \in \mathrm{FL}$ such that

$$h(x) \in S \iff g(h(x)) \notin S \tag{12}$$
$$h(x) \in \{x, g(x)\} \tag{13}$$

for all x. Define the set

$$S' := \{x \in S \mid x \neq g^{(2)}(x)\} \cup \{x \mid x = g^{(2)}(x) \text{ and } x < g(x)\}$$

and observe that also $S' \in (\mathrm{DSPACE}(\log \cdot \log^{(c)}) \cap \mathrm{P})$ holds. We will show that $A \cap S'$ and $A \cap \overline{S'}$ are $\leq^{\log}_{\text{2-tt}}$-complete for $\mathcal{C}$.

Note that $(\mathrm{DSPACE}(\log \cdot \log^{(c)}) \cap \mathrm{P})$ is closed under complement, thus $S', \overline{S'} \in \mathcal{C}$. And, since $\mathcal{C}$ is closed under intersection, we obtain $A \cap S' \in \mathcal{C}$ and $A \cap \overline{S'} \in \mathcal{C}$. So it remains to argue for the $\leq^{\log}_{\text{2-tt}}$-hardness of $A \cap S'$ and $A \cap \overline{S'}$ for $\mathcal{C}$. Since A is $\leq^{\log}_{\text{1-tt}}$-hard for $\mathcal{C}$, it remains to show $A \leq^{\log}_{\text{2-tt}} A \cap S'$ and $A \leq^{\log}_{\text{2-tt}} A \cap \overline{S'}$. We show $A \leq^{\log}_{\text{2-tt}} A \cap S'$. Let x be some input. We distinguish the following cases:

Case 1: $g^{(2)}(x) = g^{(4)}(x)$. Recall that $g^{(2)}(x) \neq g^{(3)}(x)$. Let $i \in \{0,1\}$ such that $g^{(2+i)}(x) < g^{(3-i)}(x)$. Then we have $g^{(2+i)}(x) \in S'$. Let $b \in \{0,1\}$ such that $c_A(x) = b \oplus c_A(g^{(2+i)}(x))$. Note that we can determine i and b by a constant number of applications of f and f'. We obtain $c_A(x) = b \oplus c_A(g^{(2+i)}(x)) = b \oplus c_{A \cap S'}(g^{(2+i)}(x))$.

Case 2: $g^{(2)}(x) \neq g^{(4)}(x)$. In this case, by Claim 4.4, there exists some minimal $k \leq 2$ such that $c_A(g^{(k)}(x)) = c_A(g^{(k+1)}(x)) = c_A(g^{(k+2)}(x))$. Let $b \in \{0,1\}$ such that $c_A(x) = b \oplus c_A(g^{(k)}(x))$. Let $y := h(g^{(k)}(x))$. By (13) it holds that $b \oplus c_A(x) = c_A(g^{(k)}(x)) = c_A(y) = c_A(g(y))$. By (12) it holds that $c_{S'}(y) = 1 - c_{S'}(g(y))$. We obtain $c_A(x) = b \oplus (c_{A \cap S'}(y) \vee c_{A \cap S'}(g(y)))$.

Hence, a logspace computation with at most two nonadaptive queries to $A \cap S'$ is sufficient to determine $c_A(x)$. This shows $A \leq^{\log}_{\text{2-tt}} A \cap S'$. The proof for $A \leq^{\log}_{\text{2-tt}} A \cap \overline{S'}$ works analogously. Hence A is weakly $\leq^{\log}_{\text{2-tt}}$-mitotic. □

Corollary 4.5 *Let $\mathcal{C} \supseteq (\mathrm{DSPACE}(\log \cdot \log^{(c)}) \cap \mathrm{P})$ for some $c \geq 1$ be a complexity class closed under intersection and complementation. If A is $\leq^{\log}_{\text{m}}$-complete for $\mathcal{C}$, then A is weakly $\leq^{\log}_{\text{2-tt}}$-mitotic.*

Proof If A is $\leq^{\log}_{\mathrm{m}}$-complete for $\mathcal{C}$, then $\overline{A} \in \mathcal{C}$ by the closure properties, and we have $\overline{A} \leq^{\log}_{\mathrm{m}} A$ via some $f \in \mathrm{FL}$. This means that $c_A(x) = 1 - c_{\overline{A}}(x) = 1 - c_A(f(x))$, which in particular means that $f(x) \neq x$. So, A is $\leq^{\log}_{1\text{-tt}}$-complete for $\mathcal{C}$ and $\leq^{\log}_{1\text{-tt}}$-autoreducible. From Theorem 4.3 we obtain that A is weakly $\leq^{\log}_{2\text{-tt}}$-mitotic. □

Since $\mathrm{P} \subseteq \Delta^{\mathrm{p}}_k$ for all $k \geq 0$, and each Δ^{p}_k is closed under complementation and intersection, we immediately have the following corollary. Note that this includes the $\leq^{\log}_{\mathrm{m}}$-complete sets for P as the special case where $k = 0$.

Corollary 4.6 *For every $k \geq 0$, every $\leq^{\log}_{\mathrm{m}}$-complete set for Δ^{p}_k is weakly $\leq^{\log}_{2\text{-tt}}$-mitotic.*

4.3 Disjunctive Truth-Table Complete Sets for PSPACE

We can further generalize our approach to disjunctive truth-table autoreductions of complete sets for some higher complexity classes. Here, we consider the reduction graph of some disjunctive truth-table autoreduction f. If we grant the separator enough resources, for each input x, it can determine the smallest equivalent $y \in f(x)$ and hence treat f like a many-one reduction. For most higher classes, a diagonalization method can already show (strong) mitoticity results. For PSPACE in the polynomial-time reducibility setting, however, only autoreducibility results are known. Here, our approach as described above shows weak mitoticity.

Lemma 4.7 *Let $A \in \mathrm{PSPACE}$ and $f \in \mathrm{FP}$ be a $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-autoreduction for A such that f never maps to the empty set. Then there exists a set $S \in \mathrm{PSPACE}$ such that for all x there exist $y \in f(x)$ and $z \in f(y)$ with the following properties:*

1. $c_A(x) = c_A(y) = c_A(z)$
2. $\emptyset \subsetneq (\{x, y, z\} \cap S) \subsetneq \{x, y, z\}$

Proof Let $A \in \mathrm{DSPACE}(p)$ for some polynomial p, and $f \in \mathrm{FP}$ be a $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-autoreduction for A that never maps to the empty set. Hence, for every x there exists some $k \geq 1$ such that $f(x) = \langle y_1, \ldots, y_k \rangle$. We consider the function g with

$$g(x) := \begin{cases} y_i & \text{if } f(x) = \langle y_1, \ldots, y_k \rangle \wedge y_i \in A \wedge y_j \notin A \text{ for all } j < i \text{ , and} \\ y_1 & \text{if } f(x) = \langle y_1, \ldots, y_k \rangle \wedge y_j \notin A \text{ for all } j \leq k, \end{cases}$$

for all x. Since $A \in \mathrm{DSPACE}(p)$, there exists a polynomial q such that $g \in \mathrm{FSPACE}(q)$. Furthermore, since g maps to values of f, we have $g(x) \neq x$ for all x, and we can modify q such that $|g(x)| \leq q(|x|)$.

We apply Lemma 3.7 and obtain a set S and a function h with the following properties:

1. $S \in \mathrm{DSPACE}(q \cdot \log^{(c)}) \subseteq \mathrm{PSPACE}$ (where $c \geq 1$ is some constant)
2. $h(x) \in \{x, g(x)\}$
3. $h(x) \in S \iff g(h(x)) \notin S$

Let $y := g(x)$ and $z := g(y)$. Hence $y \in f(x)$ and $z \in f(y)$, and $c_A(x) = c_A(y) = c_A(z)$. Furthermore, $h(x) \in \{x, y\}$, so we either have $x \in S \iff y \notin S$, or $y \in S \iff z \notin S$. □

Theorem 4.8
1. *All $\leq^{\mathrm{p}}_{k\text{-dtt}}$-complete sets for* PSPACE *are weakly $\leq^{\mathrm{p}}_{k(k^2+k+1)\text{-dtt}}$-mitotic.*
2. *All $\leq^{\mathrm{p}}_{\mathrm{bdtt}}$-complete sets for* PSPACE *are weakly $\leq^{\mathrm{p}}_{\mathrm{bdtt}}$-mitotic.*
3. *All $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-complete sets for* PSPACE *are weakly $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-mitotic.*

Proof We show the first statement, the other statements are shown analogously. So let L be $\leq^{\mathrm{p}}_{k\text{-dtt}}$-complete for PSPACE for some $k \geq 1$. Then, L is $\leq^{\mathrm{p}}_{k\text{-dtt}}$-autoreducible [GOP+07]. Let f be some $\leq^{\mathrm{p}}_{k\text{-dtt}}$-autoreduction for L. Note that L is non-trivial, hence we can assume that f never maps to the empty set. We apply Lemma 4.7 and obtain $S \in \mathrm{PSPACE}$ with the specified properties.

We will show that $L \cap S$ is $\leq^{\mathrm{p}}_{k(k^2+k+1)\text{-dtt}}$-complete for PSPACE. Clearly, $L \cap S \in \mathrm{PSPACE}$, so it remains to show the hardness for PSPACE. For arbitrary $A \in \mathrm{PSPACE}$ we already know that $A \leq^{\mathrm{p}}_{k\text{-dtt}} L$, hence it suffices to show $L \leq^{\mathrm{p}}_{(k^2+k+1)\text{-dtt}} L \cap S$.

On input x, return $Q_x := \{x\} \cup f(x) \cup \bigcup_{y \in f(x)} f(y)$, which can be computed in polynomial time. The number of the elements in the output is bounded by (k^2+k+1). To show that Q_x is a reduction as claimed above, choose y, z as in the lemma. We distinguish the following cases:

- If $x \in L$, then $\{x, y, z\} \subseteq L$. Since $\{x, y, z\} \cap S \neq \emptyset$ and $\{x, y, z\} \subseteq Q_x$ we obtain $(L \cap S) \cap Q_x \supseteq (L \cap S) \cap \{x, y, z\} = S \cap \{x, y, z\} \neq \emptyset$.
- If $x \notin L$, then $(L \cap S) \cap Q_x \subseteq L \cap Q_x = \emptyset$.

This shows that $L \cap S$ is $\leq^{\mathrm{p}}_{k(k^2+k+1)\text{-dtt}}$-hard and hence $\leq^{\mathrm{p}}_{k(k^2+k+1)\text{-dtt}}$-complete for PSPACE. For $L \cap \overline{S}$ we can show the completeness for PSPACE analogously. From the completeness results we obtain that L is weakly $\leq^{\mathrm{p}}_{k(k^2+k+1)\text{-dtt}}$-mitotic. □

5 Logspace Autoreducibility for NP

In this section we consider logspace complete sets for NP. In this setting, neither can we apply diagonalization (here, NP is too weak to diagonalize against logspace reductions), nor can we trace entire computation paths in the nondeterministic computation tree (because logspace reductions have too little storage). However, we know that logspace complete sets for NP are redundant in the polynomial-time setting, which gives us access to particular deterministic polynomial-time computations. We will consider the transcripts of those computations to obtain logspace redundancy results.

5.1 Autoreducibility by the Tableau Method

Theorem 5.1 *Let A be $\leq^{\log[k]}_{\mathrm{T}}$-hard for* P. *If A is $\leq^{\mathrm{p}}_{\mathrm{tt}}$-autoreducible, then A is $\leq^{\log[2k+1]}_{\mathrm{T}}$-autoreducible.*

Proof Since A is $\leq^{\mathrm{p}}_{\mathrm{tt}}$-autoreducible, there are functions $f, g \in \mathrm{FP}$ such that for all x there exists some m such that $f(x) = \langle y_1, \ldots, y_m \rangle$, $x \notin \{y_1, \ldots, y_m\}$ and $c_A(x) = g(x, c_A(y_1), \ldots, c_A(y_m))$.

Let M_1 be a polynomial-time Turing transducer that computes f, and let M_2 be a polynomial-time Turing transducer that computes g. We will consider the transcripts of the Turing transducers, which are bit string representations of the sequence of configurations of the transducer on some input, starting with the input itself, and ending on the function value computed. Given a transcript of polynomial size in n, we can verify the consistency of each bit of the transcript in space $\log(n)$ by looking at constantly many previous bits of the transcript.

On input x, let F_x denote the transcript of M_1, and let G_x denote the transcript of M_2. We assume that there are polynomials p and q such that $|F_x| = p(|x|)$ and $|G_x| = q(|x|)$. Let c be some constant such that each bit in F_x and G_x can be verified by reading at most c previous bits in the transcript.

Let $F_x[i]$ denote bit i in F_x, and let $G_x[i]$ denote bit i in G_x. We define the sets $B := \{\langle x, i \rangle \mid F_x[i] = 1\}$ and $C := \{\langle x, i \rangle \mid G_x[i] = 1\}$. Since F_x and G_x are transcripts of polynomial-time

computations, we have $B, C \in \mathrm{P}$. Since A is $\leq_{\mathrm{T}}^{\log[k]}$-hard for $\mathcal{C}$, there exist logspace oracle Turing machines N_1, N_2 with k oracle tapes such that $B = L(N_1^A)$ and $C = L(N_2^A)$. We consider the algorithm described in Figure 4.

```
On input x:
 1. for i := 1 to p(|x|):
 2.   compute U_x[i] := N_1^{A∪{x}}(⟨x, i⟩)
 3.   verify U_x[i] by reading U_x[j_1], U_x[j_2], ..., U_x[j_c] for some j_1, ..., j_c < i
 4.   if the verification fails, reject
 5. // here it holds that U_x = F_x
 6. compute m such that f(x) = ⟨y_1, ..., y_m⟩ and let y := (x, c_A(y_1), ..., c_A(y_m))
 7. for i := 1 to q(|y|):
 8.   compute V_y[i] := N_2^{A∪{x}}(⟨y, i⟩)
 9.   verify V_y[i] by reading V_y[j_1], V_y[j_2], ..., V_y[j_c] for some j_1, ..., j_c < i
10.   if the verification fails, reject
11. // here it holds that V_y = G_y = G_(x,c_A(y_1),...,c_A(y_m))
12. if V_y[q(|y|)] = 1 then accept, otherwise reject
```

Figure 4: Autoreduction for A.

Claim 5.2 *The algorithm correctly decides A.*

Proof Let x be some input to the above algorithm, and let $f(x) = \langle y_1, \ldots, y_m \rangle$ and $y = (x, c_A(y_1), \ldots, c_A(y_m))$, hence $c_A(x) = g(x, c_A(y_1), \ldots, c_A(y_m))$. We distinguish the following cases.

Case 1: $x \in A$. In this case, $A = A \cup \{x\}$, hence in line 2, in each iteration we compute $N_1^A(\langle x, i \rangle) = F_x[i]$. So in iteration i we have $U_x[j] = F_x[j]$ for all $j \leq i$, hence the verification of bit i succeeds. Hence we never reject in line 4. By a similar argumentation, we never reject in line 10. This means that we reach line 12, where $V_y = G_{(x,c_A(y_1),\ldots,c_A(y_m))}$ holds. Since $x \in A$, we have $1 = c_A(x) = g(x, c_A(y_1), \ldots, c_A(y_m)) = G_{(x,c_A(y_1),\ldots,c_A(y_m))}[q(|y|)] = V_y[q(|y|)]$, hence we accept correctly.

Case 2: $x \notin A$. The algorithm either correctly rejects in line 2 or in line 10, or it reaches line 12. In the latter case, we have verified that $U_x = F_x$ and $V_y = G_{(x,c_A(y_1),\ldots,c_A(y_m))}$. Since $x \notin A$, we have $0 = c_A(x) = g(x, c_A(y_1), \ldots, c_A(y_m)) = G_{(x,c_A(y_1),\ldots,c_A(y_m))}[q(|y|)] = V_y[q(|y|)]$, hence we reject correctly. □

Claim 5.3 *On input x, the algorithm can be executed in space $\log(|x|)$ with oracle A and $(2k+1)$ oracle tapes, such that it never queries x.*

Proof Let x be some input of the above algorithm where $n = |x|$, and let $f(x) = \langle y_1, \ldots, y_m \rangle$ and $y = (x, c_A(y_1), \ldots, c_A(y_m))$. We assume that the algorithm reaches line 12, since this argumentation includes the case where we reject earlier as well. We consider the parts of the algorithm separately.

Lines 1 to 5: The variable of the loop in line 1 can be stored in space $O(\log(n))$, so consider some particular iteration i. In line 2 we compute $N_1^{A\cup\{x\}}(\langle x, i\rangle)$, which is possible in space $O(\log(n))$ with oracle A and k oracle tapes by simulation of N_1 without querying x. The computed value $U_x[i]$ is verified in line 3, where for the verification we need the values $U_x[j_1], U_x[j_2], \ldots, U_x[j_c]$ for some $j_1, \ldots, j_c < i$. The values $j_1, \ldots, j_c$ can be computed in space $O(\log(n))$. Since $j_1, \ldots, j_c < i$, the verification of $U_x[j_1], \ldots, U_x[j_c]$ already succeeded in previous iterations, so we can sequentially simulate $N_1^{A\cup\{x\}}$ on $\langle x, j_1\rangle, \ldots, \langle x, j_c\rangle$ to obtain $U_x[j_1], \ldots, U_x[j_c]$ in logspace, again with k oracle tapes and without querying x. In particular, since c is a constant, we can store the values $U_x[j_1], \ldots, U_x[j_c]$ temporarily for the verification on the working tape, and we are not required to store the entire bit string U_x. With the values $U_x[j_1], \ldots, U_x[j_c]$ we proceed to verify $U_x[i]$, which again works in space $O(\log(n))$.

Hence the entire loop in line 1 can be executed in space $O(\log(n))$ with oracle A and k oracle tapes, such that we never query x, and after we exit the loop it holds that $U_x = F_x$.

Note that we do not store the bit string U_x.

Line 6: Having verified that $U_x = F_x$ holds, we now have access to each single bit in F_x, say bit i, by simulating $N_1^{A\cup\{x\}}(\langle x, i\rangle)$, which takes space $O(\log(n))$ and occupies k oracle tapes. Recall that the function value $f(x)$ is encoded in the last bits of F_x. So by sequentially simulating $N_1^{A\cup\{x\}}$ on $\langle x, 1\rangle, \ldots, \langle x, p(|x|)\rangle$, we also have access to each single bit of $f(x) = \langle y_1, \ldots, y_m\rangle$, and hence to each bit of y_j for each j. So in logspace we can compute m with k oracle tapes where we never query x.

Note that the value of m can be polynomially in n, hence again we cannot store $y = (x, c_A(y_1), \ldots, c_A(y_m))$ directly on a working tape. As argued above, in space $O(\log(n))$ we can compute each bit of y_j with k oracle tapes such that we never query x. We sequentially compute each bit of y_j and copy it on the $(k+1)$-st oracle tape. Since f is an autoreduction, $y_j \neq x$. So after y_j is written to the oracle tape, we can query y_j and obtain $c_A(y_j)$.

Hence each bit of y can be computed in space $O(\log(n))$ with $(k+1)$ oracle tapes and without querying x.

Lines 7 to 11: The variable of the loop in line 7 can be stored in space $O(\log(n))$, so consider some particular iteration i. In line 8 we compute $N_2^{A\cup\{x\}}(\langle y, i\rangle)$. Note that we have not stored y on the working tape, but instead in space $O(\log(n))$ we have access to each bit of y, which occupies $(k+1)$ oracle tapes. Hence, by recomputing the bits of y whenever necessary, we can compute $N_2^{A\cup\{x\}}(\langle y, i\rangle)$ in space $O(\log(n))$ with oracle A and $(2k+1)$ oracle tapes by simulation of N_2 without querying x. The thus computed value $V_y[i]$ is verified in line 9, where for the verification we need the values $V_y[j_1], V_y[j_2], \ldots, V_y[j_c]$ for some $j_1, \ldots, j_c < i$. The values $j_1, \ldots, j_c$ can be computed in space $O(\log(n))$. Since $j_1, \ldots, j_c < i$, the verification of $V_y[j_1], \ldots, V_y[j_c]$ already succeeded in previous iterations, so we can sequentially simulate $N_2^{A\cup\{x\}}$ on $\langle y, j_1\rangle, \ldots, \langle y, j_c\rangle$ to obtain $V_y[j_1], \ldots, V_y[j_c]$ in logspace, again with $(2k+1)$ oracle tapes and without querying x. In particular, since c is a constant, we can store the values $V_y[j_1], \ldots, V_y[j_c]$ temporarily for the verification on the working tape, and we are not required to store the entire bit string V_y. With the values $V_y[j_1], \ldots, V_y[j_c]$ we proceed to verify $V_y[i]$, which again works in space $O(\log(n))$.

Hence the entire loop in line 7 works in space $O(\log(n))$ with oracle A and $(2k+1)$ oracle tapes, such that we never query x, and after the loop it holds that $V_y = G_y$.

Again note that we do not store the bit string V_y.

Line 12: It remains to compute bit $q(|y|)$ in V_y, which again is possible in space $O(\log(n))$, with $(2k+1)$ oracle tapes and without querying x.

This means that we can execute the entire algorithm in space $O(\log(n))$ and hence in space $\log(n)$ with oracle A and $(2k+1)$ oracle tapes, such that on input x we never query x. □

From Claim 5.2 and Claim 5.3 it follows that the set A is $\leq_{\mathrm{T}}^{\log[2k+1]}$-autoreducible. □

Corollary 5.4 *Let A be $\leq_{\mathrm{T}}^{\log}$-hard for* P*. If A is $\leq_{\mathrm{tt}}^{\mathrm{p}}$-autoreducible, then A is $\leq_{\mathrm{T}}^{\log}$-autoreducible.*

Proof Let B be $\leq_{\mathrm{m}}^{\log}$-complete for P. Since A is $\leq_{\mathrm{T}}^{\log}$-hard for P, there exists some k such that $B \leq_{\mathrm{T}}^{\log[k]} A$. Hence, A is $\leq_{\mathrm{T}}^{\log[k]}$-hard for P. By Theorem 5.1, A is $\leq_{\mathrm{T}}^{\log[2k+1]}$-autoreducible, hence A is $\leq_{\mathrm{T}}^{\log}$-autoreducible. □

Theorem 5.5 ([GOP+07]) *Let $\mathcal{C}$ be one of the following classes:*

- PSPACE
- *the levels $\Sigma_k^{\mathrm{p}}, \Pi_k^{\mathrm{p}}, \Delta_k^{\mathrm{p}}$ of the polynomial-time hierarchy*
- 1NP
- *the levels of the Boolean hierarchy over* NP
- *the levels of the* MODPH *hierarchy*

Let r be one of the following reductions: $\leq_{\mathrm{m}}^{\mathrm{p}}$, $\leq_{1\text{-tt}}^{\mathrm{p}}$, $\leq_{\mathrm{dtt}}^{\mathrm{p}}$, and $\leq_{k\text{-dtt}}^{\mathrm{p}}$ for $k \geq 2$. Then every nontrivial set that is r-complete for $\mathcal{C}$ is r-autoreducible.

Note that each of the classes mentioned in Theorem 5.5 contains P, so here we can apply Corollary 5.4. While for PSPACE and the Δ_k^{p}-levels of the polynomial hierarchy, autoreducibility and mitoticity results are already known, we obtain new autoreducibility results for the Σ_k^{p} and Π_k^{p} levels of the polynomial-time hierarchy, including NP and coNP.

Corollary 5.6 *Let $\mathcal{C}$ be one of the following classes:*

- *the levels Σ_k^{p} and Π_k^{p} of the polynomial-time hierarchy*
- 1NP
- *the levels of the Boolean hierarchy over* NP
- *the levels of the* MODPH *hierarchy*

Let r be one of the following reductions: $\leq_{\mathrm{m}}^{\log}$, $\leq_{1\text{-tt}}^{\log}$, $\leq_{\mathrm{dtt}}^{\log}$, and $\leq_{k\text{-dtt}}^{\log}$ for $k \geq 2$. Then every nontrivial r-complete set for $\mathcal{C}$ is $\leq_{\mathrm{T}}^{\log}$-autoreducible.

Proof Let A be nontrivial and r-complete for $\mathcal{C}$. So, A is $\leq_{\mathrm{T}}^{\log}$-hard for $\mathcal{C}$. This in particular means that A is $\leq_{\mathrm{T}}^{\log}$-hard for P.

If A is $\leq_{\mathrm{m}}^{\log}$-complete or $\leq_{k\text{-dtt}}^{\log}$-complete for $\mathcal{C}$, then A is $\leq_{\mathrm{dtt}}^{\log}$-complete for $\mathcal{C}$. So, A is either $\leq_{1\text{-tt}}^{\log}$-complete or $\leq_{\mathrm{dtt}}^{\log}$-complete for $\mathcal{C}$. If A is $\leq_{1\text{-tt}}^{\log}$-complete for $\mathcal{C}$, then A is $\leq_{1\text{-tt}}^{\mathrm{p}}$-complete for $\mathcal{C}$, and if A is $\leq_{\mathrm{dtt}}^{\log}$-complete for $\mathcal{C}$, then A is $\leq_{\mathrm{dtt}}^{\mathrm{p}}$-complete for $\mathcal{C}$. Hence, A is either $\leq_{1\text{-tt}}^{\mathrm{p}}$-complete or $\leq_{\mathrm{dtt}}^{\mathrm{p}}$-complete for $\mathcal{C}$. By Theorem 5.5, A is either $\leq_{1\text{-tt}}^{\mathrm{p}}$-autoreducible or $\leq_{\mathrm{dtt}}^{\mathrm{p}}$-autoreducible. This means that A is $\leq_{\mathrm{tt}}^{\mathrm{p}}$-autoreducible.

We apply Corollary 5.4 and obtain that A is $\leq_{\mathrm{T}}^{\log}$-autoreducible. Note that A is actually $\leq_{\mathrm{T}}^{\log[1]}$-hard for $\mathcal{C}$, so from Theorem 5.1 we obtain that the Turing autoreduction uses at most three oracle tapes. □

Note that Corollary 5.6 includes the classes NP and coNP as special cases.

6 Summary

We summarize results obtained in this paper and outline future research and open problems.

Autoreducibility for NP and all other classes of the PH. We have shown that all $\leq^{\log}_{\mathrm{m}}$-complete sets for NP and all other classes of the polynomial hierarchy are $\leq^{\log}_{\mathrm{T}}$-autoreducible. Our proof builds on several known results on polynomial-time autoreducibility and mitoticity. It seems to be difficult to obtain a short self-contained proof, because on the one hand, the classes of the polynomial hierarchy are too weak to simulate arbitrary logspace reductions, and hence diagonalization techniques do not apply here, yet the classes are complex enough such that logspace reductions cannot verify their computations (for instance, in logspace, we cannot simulate an accepting NP computation path).

Observe that the obtained logspace Turing autoreduction uses at most three oracle tapes. Can we reduce the number of oracle tapes to two or even one? In the latter case, all $\leq^{\log}_{\mathrm{m}}$-complete sets for NP are $\leq^{\log}_{\mathrm{tt}}$-autoreducible.

Mitoticity for P, the Δ-levels of the PH, and NEXP. We have further obtained that all $\leq^{\log}_{\mathrm{m}}$-complete sets for P and the levels Δ^{p}_k of the polynomial hierarchy ($k \geq 2$) are weakly $\leq^{\log}_{\text{2-tt}}$-mitotic, and all $\leq^{\log}_{\text{1-tt}}$-complete sets for NEXP are weakly $\leq^{\log}_{\text{2-dtt}}$-mitotic.

Recall that all $\leq^{\log}_{\mathrm{m}}$-complete sets for PSPACE and EXP are $\leq^{\log}_{\mathrm{m}}$-mitotic. We would like to know whether the same holds for NEXP. Can we at least show $\leq^{\log}_{\text{2-dtt}}$-mitoticity or (weak) $\leq^{\log}_{\mathrm{m}}$-mitoticity for NEXP? Furthermore, can we generalize our results to further completeness notions, for instance to $\leq^{\log}_{\mathrm{dtt}}$-complete or $\leq^{\log}_{\mathrm{ctt}}$-complete sets?

Mitoticity for PSPACE. For PSPACE we have shown that all $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-complete sets are weakly $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-mitotic. Again, PSPACE is too weak to simulate arbitrary polynomial-time reductions, so standard diagonalization techniques fail to show mitoticity. Instead, we shift complexity from the reduction to the separator to obtain weak mitoticity.

It remains an open question whether all $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-complete sets for PSPACE are $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-mitotic, i.e., if one can find a separator set in P. Furthermore, it remains open if a similar result holds for $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-reductions.

References

[AS84] K. Ambos-Spies. P-mitotic sets. In E. Börger, G. Hasenjäger, and D. Roding, editors, *Logic and Machines*, volume 171 of *Lecture Notes in Computer Science*, pages 1–23. Springer Verlag, 1984.

[Ber77] L. Berman. *Polynomial Reducibilities and Complete Sets*. PhD thesis, Cornell University, Ithaca, NY, 1977.

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

[BFvMT00] H. Buhrman, L. Fortnow, D. van Melkebeek, and L. Torenvliet. Separating complexity classes using autoreducibility. *SIAM Journal on Computing*, 29(5):1497–1520, 2000.

[BHT98] H. Buhrman, A. Hoene, and L. Torenvliet. Splittings, robustness, and structure of complete sets. *SIAM Journal on Computing*, 27(3):637–653, 1998.

[BT05] H. Buhrman and L. Torenvliet. A Post's program for complexity theory. *Bulletin of the EATCS*, 85:41–51, 2005.

[Buh93] H. Buhrman. *Resource Bounded Reductions*. PhD thesis, University of Amsterdam, 1993.

[CV86] R. Cole and U. Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986.

[GH92] K. Ganesan and S. Homer. Complete problems and strong polynomial reducibilities. *SIAM Journal on Computing*, 21(4):733–742, 1992.

[Gla10] C. Glaßer. Space-efficient informational redundancy. *Journal of Computer and System Sciences*, 76(8):792–811, 2010.

[GNR+13] C. Glaßer, D. T. Nguyen, C. Reitwießner, A. L. Selman, and M. Witek. Autoreducibility of complete sets for log-space and polynomial-time reductions. In *Proceedings 40th International Colloquium on Automata, Languages and Programming (ICALP)*, Lecture Notes in Computer Science, pages 473–484. Springer Verlag, 2013.

[GOP+07] C. Glaßer, M. Ogihara, A. Pavan, A. L. Selman, and L. Zhang. Autoreducibility, mitoticity, and immunity. *Journal of Computer and System Sciences*, 73(5):735–754, 2007.

[GPSZ08] C. Glaßer, A. Pavan, A. L. Selman, and L. Zhang. Splitting NP-complete sets. *SIAM Journal on Computing*, 37(5):1517–1535, 2008.

[HJ97] L. A. Hemaspaandra and Z. Jiang. Logspace reducibility: Models and equivalences. *International Journal of Foundations of Computer Science*, 08(01):95–108, 1997.

[HKR93] S. Homer, S. A. Kurtz, and J. S. Royer. On 1-truth-table-hard languages. *Theoretical Computer Science*, 115(2):383–389, 1993.

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

[Lad75] R. E. Ladner. On the structure of polynomial-time reducibility. *Journal of the ACM*, 22(1):155–171, 1975.

[LL76] R. E. Ladner and N. A. Lynch. Relativization of questions about log space computability. *Mathematical Systems Theory*, 10:19–32, 1976.

[Lyn78] N. A. Lynch. Log space machines with multiple oracle tapes. *Theoretical Computer Science*, 6:25–39, 1978.

[NS14] D. T. Nguyen and A. L. Selman. Non-autoreducible sets for NEXP. In *Proceedings 31st Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science. Springer Verlag, 2014. To appear.

[Tra70] B. Trakhtenbrot. On autoreducibility. *Dokl. Akad. Nauk SSSR*, 192(6):1224–1227, 1970. Translation in Soviet Math. Dokl. 11(3): 814–817, 1970.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de