Weak Mitoticity of Bounded Disjunctive and Conjunctive Truth-table Autoreducible Sets [*]

Liyu Zhang [†] Mahmoud Quweider Hansheng Lei
Fitra Khan

{liyu.zhang, mahmoud.quweider, hansheng.lei, fitra.khan}@utrgv.edu

Department of Computer Science,
University of Texas Rio Grande Valley,
One West University Boulevard, Brownsville, TX, 78520, USA

September 8, 2019

Abstract

Glaßer et al. (SIAMJCOMP 2008 and TCS 2009)[1] proved existence of two sparse sets A and B in EXP, where A is 3-tt (truth-table) polynomial-time autoreducible but not weakly polynomial-time Turing mitotic and B is polynomial-time 2-tt autoreducible but not weakly polynomial-time 2-tt mitotic. We unify and strengthen both of those results by showing that there is a sparse set in EXP that is polynomial-time 2-tt autoreducible but not even weakly polynomial-time Turing mitotic. All these results indicate that polynomial-time autoreducibilities in general do not imply polynomial-time mitoticity at all with the only exceptions of the many-one and 1-tt reductions. On the other hand, however, we proved that every autoreducible set for the polynomial-time bounded disjunctive or conjunctive tt reductions is weakly mitotic for the polynomial-time tt reduction that makes logarithmically many queries only. This shows that autoreducible sets for reductions making more than one query could still be mitotic in some way if they possess certain special properties.

keywords: computational complexity; polynomial-time autoreducibility; weak polynomial-time mitoticity; bounded polynomial-time truth-table autoreducible sets.

1 Introduction

Let r be a reduction between two languages as defined in computational complexity such as the common *many-one* and *Turing* reductions. We say that a language L is r-*autoreducible* if L is

[*]A preliminary version of this paper appears in the proceedings of COCOON'18 (24th International Computing and Combinatorics Conference, 2018)

[†]Research supported in part by NSF CCF grant 1218093.

[1]The two papers have slightly different sets of authors

reducible to itself via the reduction r where the reduction does not query on the same string as the input. In case that r is the many-one reduction, we require that r outputs a string different from the input in order to be an autoreduction. Researchers started investigating on autoreducibility as early as 1970's [12] although much of the work done then was in the recursive setting. Ambos-Spies [1] translated the notion of autoreducibility to the polynomial-time setting, and Yao [13] considered autoreducibility in the probabilistic polynomial-time setting, which he called *coherence*.

More recently polynomial-time autoreducibilities, which correspond to polynomial-time reductions, gained attention due to its candidacy as a structural property that can be used in the "*Post's program for complexity theory*" [3] that aims at finding a structural/computational property that complete sets of two complexity classes don't share, hereby separating the two complexity classes. Autoreducibility is believed to be possibly one of such properties that will lead to new separation results in the future [2]. We refer the reader to Glaßer et al. [7] and Glaßer et al. [6] for recent surveys along this line of research.

In this paper we continue to study the relation between the two seemingly different notions, autoreducibility and mitoticity. Glaßer et al. [9] proved that among polynomial-time reductions, autoreducibility coincides with polynomial-time mitoticity for the many-one and 1-tt reductions, but not for the 3-tt reduction or any reduction weaker than 3-tt. In a subsequent paper Glaßer et al. [4] further proved that 2-tt autoreducibility does not coincide with 2-tt mitoticity. However, the set they construct is weakly 5-tt mitotic. So the technical question remained open whether one can construct a language that is 2-tt autoreducible but not weakly Turing-mitotic. We solve this problem in the positive way. More precisely, we proved that there exists a sparse set in EXP that is 2-tt autoreducible but not even weakly Turing-mitotic. This result unifies and strengthens both of the previous results.

In attempting to strengthen our results further we asked the question whether one can even construct a language that is r-autoreducible but not weakly Turing-mitotic for any reduction r that is weaker than the 1-tt reduction but stronger than 2-tt reduction such as 2-dtt and 2-ctt reductions. We proved that any language that is k-dtt or k-ctt autoreducible is also weakly $k^{O(2^c \log^{(c-1)} n)}$-tt mitotic for any integers $k, c \geq 2$. Glaßer et al. [9] and Glaßer et al. [5] showed that k-dtt and/or k-ctt complete set for many common complexity classes including NP, PH, PSPACE and NEXP are k-dtt and/or k-ctt autoreducible, respectively. In light of that we have the interesting corollary that k-dtt and/or k-ctt complete sets of those complexity classes are weakly tt-mitotic.

The rest of the paper is organized as follows. Section 2 provides definitions and notations needed to present our results. Section 3 provides proofs for our first main result that there is a sparse set in EXP that is 2-tt autoreducible but not weakly Turing-mitotic. Section 4 provides proofs for our second main result that any k-dtt or k-ctt autoreducible sets are weakly $k^{O(2^c) \log^{(c-1)} n}$-tt mitotic for all integers $k, c \geq 2$.

2 Definitions and Notations

We assume familiarity with basic notions in complexity theory and particularly, common complexity classes such as P, NP, PH, PSPACE and EXP, and polynomial-time reductions including many-one ($\leq_m^p$), truth-table ($\leq_{tt}^p$) and Turing reductions ($\leq_T^p$) [11, 10]. Without loss of generality, we use the alphabet $\Sigma = \{0, 1\}$ and all sets we referred to in this paper are either languages over Σ or sets of *integers*. Let $\mathbb{N}$ denote the set of natural numbers and $\mathbb{N}^+$ denote $\mathbb{N}\backslash\{0\}$. We use a pairing function $\langle\cdot,\cdot\rangle$ that satisfies $\langle x, y\rangle > x + y$. For every string/integer x, we use $|x|/abs(x)$ to denote the length/absolute value of x. For every function f and every $i \in \mathbb{N}$,, we define

$$f^{(i)}(x) = \begin{cases} x & \text{if } i = 0, \text{ and} \\ f^{(i-1)}(x) & \text{otherwise.} \end{cases}$$

Throughout the paper, we use the two terms *Turing machines* and *algorithms* interchangeably. All reductions used in this paper are *polynomial-time computable* unless otherwise specified. A language L is *complete* for a complexity class $\mathcal{C}$ for a reduction r if every language in $\mathcal{C}$ is reducible to L via r. For any algorithm or Turing machine $\mathcal{A}$, we use $\mathcal{A}(x)$ to denote both the execution and output of $\mathcal{A}$ on input x, i.e., "$\mathcal{A}(x)$ accepts" has the same meaning as "$\mathcal{A}(x) = accept$". We use $\mathcal{A}^B(x)$ or $\mathcal{A}^g(x)$ to denote the same for algorithm/Turing machine $\mathcal{A}$ that has oracle access to a set B or a function g, respectively. Also $L(\mathcal{A})$ ($L(\mathcal{A}^B)$ or $L(\mathcal{A}^g)$) denotes the language accepted by $\mathcal{A}$ ($\mathcal{A}^B$ or $\mathcal{A}^g$).

We provide detailed definitions for the most relevant reductions considered in this paper below.

Definition 2.1. *Define a language A to be* polynomial-time truth-table (tt) reducible *($\leq^p_{tt}$) to a language B, if there exists a polynomial-time algorithm $\mathcal{A}$ that accepts A with oracle access to B. In addition, there exists a polynomial-time computable function g that on input x outputs all queries $\mathcal{A}(x)$ makes to B.*

Truth-table reductions are also called *nonadaptive Turing reductions* in the sense that they are the same as the general Turing reductions except that all queries the reductions make can be computed from the input in polynomial time without knowing the answer to any query.

Definition 2.2. *For any positive integer k, define a language A to be* polynomial-time k-tt reducible *($\leq^p_{k\text{-}tt}$) to a language B, if there exists a polynomial-time truth-table reduction r from A to B that makes at most k queries on every input x. If in addition the k queries q_0, q_1, $\cdots$, q_{k-1} that r makes are such that $x \in A$ if and only if some (every) $q_i \in B$, then r is called a* k-disjunctive (k-conjunctive) truth-table reduction *and A is said to be* k-disjunctive (k-conjunctive) truth-table reducible *($\leq^p_{k\text{-}dtt}/\leq^p_{k\text{-}ctt}$) to B.*

Now we define autoreducible and mitotic languages formally.

Definition 2.3. *Given any reduction r, a language is* autoreducible for r *or* r-autoreducible*, if the language is reducible to itself via r that does not query on the input.*

Definition 2.4. *Given any reduction r, a language L is* weakly mitotic for r *or* weakly r-mitotic*, if there exists another language S, where L, $S \cap L$ and $\overline{S} \cap L$ are all equivalent under the reduction r, i.e., $L \equiv_r S \cap L \equiv_r \overline{S} \cap L$. If in addition $S \in \mathrm{P}$, then we say that L is* mitotic for r *or* r-mitotic.

The proof of our second result uses *log derivative sequences* based on *log-distance functions*. We define both concepts below.

Let *sgn* denote the common sign function defined on integers, i.e., for every $z \in \mathbb{Z}$, $sgn(z) = 1$ if $z \geq 0$ and $sgn(z) = -1$ otherwise.

Definition 2.5. *For every pair of integers or strings x and y, we define the following* log-distance *function, logD, as follows.*

$$logD(x,y) = \begin{cases} sgn(y-x)\lfloor \log |y-x| \rfloor & \text{if } x \neq y \text{ and } \infty \notin \{x,y\}, \\ \infty & \text{otherwise.} \end{cases}$$

In case where x and y are strings, $y-x$ is defined to be their lexicographical difference.

The above function is the same as the "distance function" defined by Glaßer et al. [9], except that we define $logD(x,y) = \infty$ instead of 0 when $x = y$, or either x or y is ∞.

Definition 2.6. *Let $X = \{x_j\}_{j\geq 0}$ be a sequence of strings or integers, where x_j denotes the j-th element in X. Define the i-*th log derivative sequence *of X, written $X^{(i)}$, as follows:*

- $X^{(0)} = X$, *and*
- *For $i \geq 1$, $X^{(i)} = \{x_j^{(i)}\}$, where $x_j^{(i)} = logD(x_j^{(i-1)}, x_{j+1}^{(i-1)})$.*

In case X is a finite sequence $\{x_j\}_{s\leq j\leq t}$, where $s,t \in \mathbb{N}$ and $s \leq t$, then $X^{(i)} = \{x_j^{(i)}\}_{s\leq j\leq t-i}$ for every $i \in [0, t-s]$. For every $i \geq 2$, we say that $X^{(i)}$ is a higher-order *log derivative sequence of X.*

3 Non-mitoticity of 2-tt Autoreducible sets

Our first main result is that there exists a sparse set in EXP that is 2-tt autoreducible but not weakly mitotic even for the polynomial-time Turing reduction, the most general polynomial-time reduction.

Overall the proof of our first main result follows the approach of the proof by Glaßer et al. [4] that there exists a sparse set in EXP that is 2-tt autoreducible but not 2-tt mitotic. The proof is in general a *diagonalization* against all possible partitions of a constructed language L into L_1 and L_2, as well as all possible polynomial-time oracle Turing machines M_i and M_j, where $L \leq^p_{2\text{-}tt} L_1$ via M_i and $L \leq^p_{2\text{-}tt} L_2$ via M_j. The construction of L proceeds in stages, where in each stage, only polynomially many strings of a particular length are added to L. The gaps between lengths of strings added to L in different stages are made super-exponential so that strings added to L in later stages won't affect the computations of considered Turing machines on strings added to L in previous stages. In light of the fact that the set constructed as described above is actually weakly 5-tt mitotic [4], it was assumed that a straightforward adaption of the above construction won't be sufficient for proving the stronger result that there exists a sparse set in EXP that is 2-tt autoreduction but not weakly Turing mitotic.

However, something overlooked here is that the aforementioned proof actually proved a stronger statement than stated. The proof actually showed that there is a set L, where for every partition $\{L_1, L_2\}$ of L, either $L \not\leq^p_{2\text{-}tt} L_1$ or $L \not\leq^p_{2\text{-}tt} L_2$, while we only need to show $L \not\leq^p_{2\text{-}tt} L_1$, $L_1 \not\leq^p_{2\text{-}tt} L_2$ or $L_2 \not\leq^p_{2\text{-}tt} L$ in order to prove that L is not 2-tt mitotic . The latter is clearly a weaker statement. In light of this observation, we adapt the previous proof by doing diagonalization against *three* oracle

Turing machines M_i, M_j and M_k instead of two in each stage of constructing the language L. It turns out that this is critical for the proof to go through.

We now state our first main result in detail below.

Theorem 3.1. *There exists $L \in \mathrm{SPARSE} \cap \mathrm{EXP}$ such that*

- *L is 2-tt-autoreducible, but*
- *L is not weakly Turing-mitotic.*

The proof requires the following lemma by Glaßer et al. [9]:

Lemma 3.2 ([9]). *Let $l, m \geq 0$ and let $k \geq (l+2)^{2^m}$. If $Q_1, \dots, Q_k$ are sets of cardinality $\leq l$ and if $n_1, \dots, n_k$ are pairwise different numbers, then there exist pairwise different indices $i_1, \dots, i_m$ such that for all $s, t \in [1, m]$,*

$$s \neq t \;\Rightarrow\; n_{i_s} \notin Q_{i_t}.$$

Now comes the proof of Theorem 3.1.

Proof. Define a tower function by $t(0) = 4$ and

$$t(n+1) = 2^{2^{2^{2^{t(n)}}}}.$$

For any string s, let $W(s) \overset{df}{=} \{s00, s01, s10\}$. We will define L such that it satisfies the following:

(i) If $w \in L$, then there exists n such that $|w| = t(n)$.

(ii) For all n, all $s \in \Sigma^{t(n)-2}$, it holds that $W(s) \cap L$ contains exactly one element.

It is easy to see that such an L is 2-tt-autoreducible: On input w, determine n such that $|w| = t(n)$. If such n does not exist, then reject. Otherwise, accept if and only if $L \cap (W(s) - \{w\}) = \emptyset$, where $w \in W(s)$. This is clearly a 2-tt-autoreduction.

We turn to the construction of L. Let $M_1, M_2, \dots$ be an enumeration of deterministic, polynomial-time-bounded oracle Turing machines such that for all i, the running time of M_i is $n^i + i$. We construct L stagewise such that in stage n we determine which of the strings of length $t(n)$ belong to L. In other words, at stage n we define a set $S_n \subseteq \Sigma^{t(n)}$, and finally we define L to be the union of all S_n's.

We start by defining $S_0 = \emptyset$. Suppose we are at stage $n > 0$. Let $m = t(n)$ and determine i, j and k such that $n = \langle i, j, k \rangle$. If such i, j and k do not exist, then let $S_n = \emptyset$ and go to stage $n+1$. Otherwise, i, j and k exist. In particular, $i + j + k < n < \log\log m$. Let $O \overset{df}{=} S_0 \cup \dots \cup S_{n-1}$ be the part of L that has been constructed so far. Let $O_1, O_2, \dots, O_l$ be the list of all subsets of O (lexicographically ordered according to their characteristic sequences). Since $O \subseteq \Sigma^{\leq t(n-1)}$ we obtain $\|O\| \leq 2^{t(n-1)+1}$. Therefore,

$$l \leq 2^{2^{t(n-1)+1}} \leq 2^{2^{2^{t(n-1)}}} = \log\log t(n) = \log\log m. \tag{1}$$

We give some intuition for the claim below. If L is weakly Turing-mitotic, then there exists a partition $L = L_1 \cup L_2$ such that $L \leq_T^p L_1 \leq_T^p L_2 \leq_T^p L$, via some machines M_i, M_j, and M_k, in that order. Let $e \in \{i, j, k\}$. Note that $O \cap L_1$ must appear (say as O_u) in our list of subsets of O. The following claim makes sure that we can find a list of strings $s_1, \ldots, s_l$ of length $m-2$ such that for all $u \in [1, l]$ it holds that if the partition of L is such that $O \cap L_1 = O_u$, then none of M_e's on input of a string from $W(s_u)$ queries the oracle for strings from $W(s_v)$ if $u \neq v$. Also, we will construct L such that

$$L \cap \Sigma^{t(n)} \subseteq W(s_1) \cup W(s_2) \cdots W(s_l).$$

Hence, if M_e on input of a string from $W(s_u)$ queries a string of length m that does not belong to $W(s_u)$, then it always gets a no answer. So the following is the only information about the partition of L that can be exploited by M_e on an input string in $W(s_u)$:

- the partition of $O = \Sigma^{<t(n)} \cap L$
- the partition of $W(s_u) \cap L$

In particular, M_e cannot exploit information about the partition of $W(s_v) \cap L$ for $v \neq u$. This independence of M_e's behavior on inputs in $W(s_u)$ from that on inputs in $W(s_v)$, where $s_u \neq s_v$, makes our diagonalization possible.

Claim 3.3. *For sufficiently large m, there exist pairwise different strings $s_1, \ldots, s_l \in \Sigma^{m-2}$ such that for all $u, v \in [1, l]$, $u \neq v$, all $x \in W(s_u)$, and all $V' \subseteq W(s_u)$, none of $M_i^{O_u \cup V'}$, $M_j^{(O-O_u)\cup V'}$ and $M_k^{O \cup V'}$ queries the oracle for strings in $W(s_v)$ on input x.*

Proof. For $s \in \Sigma^{m-2}$, let

$$Q_s \overset{df}{=} \{s' \in \Sigma^{m-2} \mid \text{There exist } x \in W(s),\ y \in W(s'),\ u \in [1,l], \text{ and } V' \subseteq W(s), \text{ where } y \text{ is queried by } M_i^{O_u \cup V'}, M_j^{(O-O_u)\cup V'} \text{ or } M_k^{O\cup V'} \text{ on input } x.\}$$

Observe for every $s \in \Sigma^{m-2}$ that

$$\begin{aligned}
\|Q_s\| &\leq 3 \cdot l \cdot 2^3 \cdot ((m^i + i) + (m^j + j) + (m^k + k)) \text{ (since } \|W(s)\| = 3) \\
&\leq 24 \cdot \log\log m \cdot (m^{i+j+k} + (i + j + k)) \text{ (By Equation (1))} \\
&\leq 24 \cdot \log\log m \cdot (m^{\log\log m} + \log\log m) \\
&\leq 48 \cdot \log\log m \cdot m^{\log\log m} \\
&\leq 2^{\log^2 m} - 2 \text{ (for sufficiently large } m) \qquad (2)
\end{aligned}$$

We identify numbers in $[1, 2^{m-2}]$ with strings in Σ^{m-2}. Considered in this way, each Q_s is a subset of $[1, 2^{m-2}]$. By Inequality (2), $Q_1, Q_2, \ldots, Q_{2^{m-2}}$ are sets of cardinality at most $2^{\log^2 m} - 2$. Clearly, $1, 2, \ldots, 2^{m-2}$ are pairwise different numbers. By Equation (1), it follows that

$$2^{m-2} \geq (2^{\log^3 m}) \geq (2^{\log^2 m})^{2^l}.$$

Therefore, we can apply Lemma 3.2 with $m' = l$, $l' = 2^{\log^2 m} - 2$, and $k' = 2^{m-2}$. We obtain strings $s_1, \ldots, s_l$ in Σ^{m-2} such that for all $u, v \in [1, l]$,

$$u \neq v \ \Rightarrow \ s_v \notin Q_{s_u}. \tag{3}$$

Now assume there exist $u, v \in [1, l]$, $u \neq v$, $x \in W(s_u)$, and $V' \subseteq W(s_u)$, where some $q \in W(s_v)$ is queried by $M_i^{O_u \cup V'}(x)$, $M_j^{(O-O_u)\cup V'}$, or $M_k^{O\cup V'}$ on input x. Then $s_v \in Q_{s_u}$. This contradicts Statement (3) and finishes the proof of Claim 3.3. □

Let $s_1, \ldots, s_l \in \Sigma^{m-2}$ be the strings assured by Claim 3.3. The following claim allows us define an appropriate set $V_u \subseteq W(s_u)$ for every $u \in [1,\ l]$ and three given oracle Turing reductions M_i, M_j and M_k so that $O \cup V_u$ cannot be weakly Turing mitotic via M_i, M_j and M_k.

Claim 3.4. *For every $u \in [1,\ l]$ and $x \in W(s_u)$, define*

- $S_x = L(M_k^{O\cup\{x\}}) \cap W(s_u)$ *and*
- $T_x = L(M_j^{(O-O_u)\cup S_x}) \cap W(s_u)$.

Then for every $u \in [1,\ l]$, there exists $x \in W(s_u)$, where at least one of the following must hold:

$$S_x \not\subseteq \{x\} \tag{4}$$
$$T_x \neq \{x\} - S_x \tag{5}$$
$$L(M_i^{O_u \cup T_x}) \cap W(s_u) \neq \{x\} \tag{6}$$

Proof. (Of Claim 3.4) Assume that the claim is false for some $u \in [1,\ l]$. Then all the following hold for every $x \in W(s_u)$:

$$S_x \subseteq \{x\} \tag{7}$$
$$T_x = \{x\} - S_x \tag{8}$$
$$L(M_i^{O_u \cup T_x}) \cap W(s_u) = \{x\} \tag{9}$$

It follows that

$$\forall x \in W(s_u)\ (S_x \cap T_x = \emptyset) \ \wedge \ (S_x \cup T_x = \{x\}) \tag{10}$$
$$\forall\ x, x' \in W(s_u)\ (x \neq x') \implies (S_x \neq S_{x'} \ \wedge \ T_x \neq T_{x'}) \tag{11}$$

Statement (11) holds because otherwise

$$T_x = L(M_j^{(O-O_u)\cup S_x}) \cap W(s_u) = L(M_j^{(O-O_u)\cup S_{x'}}) \cap W(s_u) = T_{x'}, \text{ and}$$

$$\{x\} = L(M_i^{O_u \cup T_x}) \cap W(s_u) = L(M_i^{O_u \cup T_{x'}}) \cap W(s_u) = \{x'\}.$$

Note that Statement (11) implies that there is at most one element $x \in W(s_u)$, where $S_x = \emptyset$, and at most one element $x' \in W(s_u)$, where $T_{x'} = \emptyset$. On the other hand, however, Statement (10) implies that for every $x \in W(s_u)$ either $S_x = \emptyset$ or $T_x = \emptyset$. This is a contradiction since $W(s_u)$ contains three different elements.

$\square$

Now for every $u \in [1, l]$, let $x \in W(s_u)$ be the string assured by Claim 3.4 and define a set $V_u = \{x\}$. Then we define $S_n \overset{df}{=} \bigcup_{u \in [1,l]} V_u$ and $L = \bigcup_n S_n$.

Note that by the construction, $S_n \subseteq \Sigma^{t(n)}$ which shows (i). Observe that the construction also ensures (ii). We argue for $L \in \text{EXP}$: Since $l \leq \log\log m$, there are no more than $2^{m \log\log m}$ possibilities to choose the strings $s_1, \ldots, s_l$. For each such possibility we have to simulate $O(l^2)$ computations $M_i(x)$, $M_j(x)$ and $M_k(x)$ to determine if they meet the requirements by Claim 3.3. This can be done in exponential time in m. For the definition of each V_u we have to simulate a constant number of computations $M_i(x)$, $M_j(x)$ and $M_k(x)$. This shows that L is printable in exponential time. Hence $L \in \text{EXP}$. From the construction it follows that $\|L \cap \Sigma^m\| \leq l \leq \log\log m$ since $\|V_u\| = 1$ as defined for every $u \in [1,\ l]$. This shows $L \in \text{SPARSE}$. It remains to show that L is not weakly Turing mitotic.

Assume that L is weakly Turing mitotic. So L can be partitioned into $L = L_1 \cup L_2$, where $L_1 \cap L_2 = \emptyset$, and there exist machines M_i, M_j and M_k, where

(iii) $L \leq_T^p L_1$ via M_i,

(iv) $L_1 \leq_T^p L_2$ via M_j, and

(v) $L_2 \leq_T^p L$ via M_k.

Let $n = \langle i, j, k \rangle$, $m = t(n)$, and $O = S_0 \cup \cdots \cup S_{n-1}$, i.e., $O = L \cap \Sigma^{<t(n)}$. Let $O_1, O_2, \ldots, O_l$ be the list of all subsets of O (again lexicographically ordered according to their characteristic sequences). Let $s_1, \ldots, s_l$ and $V_1, \ldots, V_l$ be as in the definition of S_n. Choose $u \in [1, l]$ such that $L_1 \cap \Sigma^{<t(n)} = O_u$ and hence $L_2 \cap \Sigma^{<t(n)} = O - O_u$. Then a set $V_u \subseteq \Sigma^{t(n)}$ was choosen according to Claim 3.4 and added to L during Stage n.

Note that s_u was chosen according to Claim 3.3. Considering also that for all $e \in \{i, j, k\}$ and $x \in W(s_u)$, M_e on input x runs in time

$$|x|^e + e = m^e + e < m^m + m < t(n+1),$$

it follows that for all $x \in W(s_u)$, $M_i^{L_1}$, $M_j^{L_2}$ and M_k^{L} behave the same on input x as $M_i^{O_u \cup (L_1 \cap V_u)}$, $M_j^{(O - O_u) \cup (L_2 \cap V_u)}$ and $M_k^{O \cup (L \cap V_u)}$, in that order. In particular, all the following hold:

$$L(M_k^{O \cup V_u}) \cap W(s_u) = L_2 \cap V_u \subseteq V_u \tag{12}$$

$$L(M_j^{(O - O_u) \cup (L_2 \cap V_u)}) \cap W(s_u) = V_u - (L_2 \cap V_u) \tag{13}$$

$$L(M_i^{O_u \cup (V_u - L_2)}) \cap W(s_u) = L \cap V_u \tag{14}$$

By Claim 3.4 $V_u = \{x\}$, where $x \in W(s_u)$ and at least one of the Statements (4) - (6) is true. This is, however, a contradiction since Statements (4) - (6) are exactly the negations of Statements (12) - (14), when $V_u = \{x\}$, $S_x = L(M_k^{O \cup V_u}) \cap W(s_u)$ and $T_x = L(M_j^{(O-O_u) \cup S_x}) \cap W(s_u)$.

We have shown that L cannot be partitioned into L_1 and L_2, where $L \cap W(s_u) = V_u$ and (iii)-(v) all hold. Since we defined V_u for every possible partition of L, L cannot be mitotic through the reductions M_i, M_j and M_k in the way as described by (iii)-(v). This shows that L is not weakly Turing-mitotic.

□

4 Weak Mitoticity of ctt- and dtt-Autoreducible sets

Theorem 3.1 indicates that 2-tt autoreducibility does not imply weak mitoticity for the Turing reduction, the most general polynomial-time reduction. This shows that in general autoreducibility does not even imply the weakest form of mitoticity in the polynomial-time setting among reductions making more than one query, despite that autoreducibility and mitoticity are equivalent for the many one and 1-tt reductions. A further question that is natural to ask is whether autoreducibility implies any form of mitoticity at all for reductions that lie between the 2-tt reduction and 1-tt reduction, or reductions with special properties that are incomparable to 2-tt and/or 1-tt reductions, such as honest and positive reductions.

Here we consider *bounded disjunctive* and *conjunctive* truth-table reductions. We prove that if a language is k-dtt or k-ctt autoreducible for some integer $k \geq 2$, then the language is weakly truth-table mitotic. In addition, the reduction can be made to query on at most $k^{O(2^c \log^{(c-1)} n)}$ strings for every integer $c \geq 2$. Our proof adapts and generalizes in a significant way the proof strategy used by Glaßer et al. [9], where they showed that every nontrivial language is many-one or 1-tt autoreducible if and only if the language is many-one or 1-tt mitotic, respectively. We review their proof strategy below at a higher level and then describe the changes needed in order to establish the weak mitoticity of any k-dtt or k-ctt autoreducible sets.

Let L be a nontrivial many-one ($\leq_m^p$) autoreducible set, where there exists a polynomial-time computable function f such that $f(x) \neq x$ and $f(x) \in L$ if and only if $x \in L$ for every $x \in \Sigma^*$. It is sufficient to prove that there exists a polynomial-time decidable set S and a function f', where for every $x \in \Sigma^*$,

(i) $f'(x) \in L$ if and only if $x \in L$, and

(ii) $f'(x) \in S$ if and only if $x \in \overline{S}$.

The idea of finding a function f' that satisfies conditions (i) and (ii) as stated above is to define $f'(x) = f^{(i)}(x)$ for an appropriate $i \leq p(|x|)$, where p is a polynomial. Since $f(x)$ is an autoreduction, it is obvious that f' defined in this way satisfies condition (i). Now we need to construct a set S where for each x we can find a correct i so that Condition (ii) also holds, i.e., $f'(x) = f^{(i)}(x) \in S$ if and only if $x \in \overline{S}$.

To construct such set S we consider the sequence x, $f(x)$, $f(f(x))$, $\ldots$, called *trajectory* of x in Glaßer et al. [9]. Note that every string on the trajectory of x has the same membership in L as x and also that every two consecutive strings on the trajectory are unequal to each other. We first partition Σ^* into a sequence of *segments*, each of which contains all strings of some consecutive lengths. In addition, those segments are assigned to S and $\overline{S}$ in an alternate way, i.e., the i-th segment is assigned to S if and only if the $(i+1)$-st is assigned to $\overline{S}$. Then we look for changes of monotonicity in the trajectory of x and assign x to S or $\overline{S}$ accordingly. For instance, assign x to S if $x < f(x)$ and $f(x) > f(f(x))$ and to $\overline{S}$ if $x > f(x)$ and $f(x) < f(f(x))$. Clearly if we can find more than one change in monotonicity along the trajectory of x within polynomially many strings, then we will find a string y where $y \notin S$ if and only if $x \in S$. Otherwise, we can find a strictly monotonic sub-trajectory within the trajectory of x. If within that sub-trajectory, f increases or decreases fast enough, i.e., leading to a change in length, then we can find strings on the trajectory of x that are polynomially many strings from x but belong to different segments and hence have different memberships of S by x.

The most difficult case arises when the trajectory of x is a strictly monotonic but does not increase or decrease fast enough so that trajectory can reach a neighboring segment from the segment containing x within polynomially many strings away from x. Glaßer et al. [9] dealt with this case essentially by dividing each segment into smaller segments of increasing sizes based on a *log distance* function applied on strings on the trajectory of x. This way the trajectory will contain strings in neighboring segments of the same size depending on how fast the autoreduction function increases or decreases. Then we can find a y in a neighboring segment from x on the trajectory, where $y \in S$ if and only $x \in \overline{S}$.

The above strategy obviously does not apply to reductions making more than one queries as it is. We, however, found a way to adapt the strategy to apply it on bounded dtt or ctt reductions and prove the weak tt mitoticity of bounded dtt or ctt autoreducible sets.

Consider a k-dtt autoreducible set L for some integer $k \geq 2$. Then there exists a polynomial-time computable function f, where for every $x \in \Sigma^*$, $f(x) = \langle y_0, y_1, \cdots, y_{k-1}\rangle$ such that

(i) $y_j \neq x$ for every $j \in [0, k-1]$, and

(ii) $f(x) \in L$ if and only if $y_j \in L$ for some $j \in [0, k-1]$.

Now define a function g where $g(x)$ is the *lexicographically least* string in $f(x)$ that has the same membership of L as x. Then it is clear that $g(x) \neq x$ and $g(x) \in L$ if and only if $x \in L$. We can apply Glaßer et al.'s construction [9] as described above on g and establish the tt mitoticity of any k-dtt autoreducible set. The problem with this approach is, however, that the function g might not be polynomial-time computable. We circumvented this problem by considering all *possible* values of $g(x)$ for every x, i.e., any of the k values in $f(x)$, This means that we will look for a $y \neq x$ where $y \in S$ if and only if $x \notin S$, along all *possible* trajectories from x. Hence, we no longer can afford traversing along a trajectory from x for polynomially many strings before we can find the desired y since there could be exponentially many possible trajectories. Instead, we need another way to construct the set S so that there exists a y on the g-trajectory that is at most $O(\log n)$ strings away from x, where $x \in S$ if and only if $y \notin \overline{S}$.

We solve this problem by considering *higher-order log derivatives*, defined in Section 2, of the sequence consisting of strings on the trajectory of x, i.e., $G(x) = \{x_j = g^{(j)}(x)\}_{j\geq 0}$. The log-distance function used by Glaßer et al. [9] can be viewed as the *first-order log derivative* of the sequence X. We will attempt to find changes of monotonicity in X, the 1st-order log derivative of X, the 2nd-order log derivative of X and so on until the c-th order log derivative for some integer $c \geq 2$, in that order, and then assign x to either S or $\overline{S}$ accordingly. If we don't find enough changes of monotonicity among all those high-order log derivative sequences of X, then we will show that the function g increases fast enough already so that for some $j \in [1, O(log|x|)$, $g^{(j)}(x)$ belongs to the next segment after the one containing x. Hence, $g^{(j)}(x)$ will be assigned to $\overline{S}$ if and only if x is assigned to S. We now provide the detailed proof for our main theorem.

We first need the following lemma that was essentially proved in Glaßer et al. [9]. We provide the proof of the lemma here for completeness.

Lemma 4.1 ([9]). *Let $\{x_j\}_{0\leq j\leq 2}$ be a strictly monotonic sequence of integers, where there exists some $d \in \mathbb{Z}$ such that $logD(x_j, x_{j+1}) = d$ for $0 \leq j \leq 1$. Then the set $X = \{\lfloor \frac{x_j}{2^{abs(d)+1}} \rfloor \mid 0 \leq j \leq 2\}$ contains at least one even number and one odd number.*

Proof. 4.1 We only prove the lemma for strictly increasing sequences. The proof for strictly decreasing sequence is the same by letting $d' = -d$ and $x'_j = x_{3-j}$ for $0 \leq j \leq 2$.

Let $\{x_j\}_{0\leq j\leq 2}$ be a strictly increasing sequence of integers. For each $j \in \{0, 1\}$, since $logD(x_j, x_{j+1}) = d$ and $x_{j+1} > x_j$, we have $\lfloor \log(x_{j+1} - x_j) \rfloor = d \geq 0$ and hence,

$$\begin{array}{rc} & 2^d \leq x_{j+1} - x_j < 2^{d+1}, \text{ for } j \in \{0,1\} \\ \Longrightarrow & \frac{1}{2} \leq \frac{x_{j+1}}{2^{d+1}} - \frac{x_j}{2^{d+1}} < 1, \text{ for } j \in \{0,1\} \\ \Longrightarrow & 1 \leq \frac{x_2}{2^{d+1}} - \frac{x_0}{2^{d+1}} < 2 \\ \Longrightarrow & \lfloor \frac{x_2}{2^{d+1}} \rfloor = \lfloor \frac{x_0}{2^{d+1}} \rfloor + 1 \end{array}$$

The last equation above shows that one of the values $\lfloor \frac{x_2}{2^{d+1}} \rfloor$ and $\lfloor \frac{x_0}{2^{d+1}} \rfloor$ is even and the other is odd. This finishes the proof of Lemma 4.1.

□

Theorem 4.2. *Let g be a polynomially-bounded function and $g(x) \neq x$ for every $x \in \Sigma^*$. Then for every positive integer $c \geq 2$, there is a polynomial-time algorithm $\mathcal{S}_c$ with oracle access to function g, a polynomial r, and a constant $d > 0$, where for every $x \in \Sigma^*$, an integer $j_x \in [0, \lceil d2^c \log^{(c-1)}(|x|) \rceil]$ exists such that*

(i) for each $j \in [0,\ j_x]$, $|g^{(j)}(x)| \leq r(|x|)$,

(ii) for each $j \in [0,\ j_x - 1]$, $\mathcal{S}_c^g$ accepts x if and only if $\mathcal{S}_c^g$ accepts $g^{(j)}(x)$, and

(iii) $\mathcal{S}_c^g$ accepts x if and only if $\mathcal{S}_c^g$ rejects $g^{(j_x)}(x)$.

Proof. Let g be an $(n^l + l)$-bounded function for some $l \in \mathbb{N}^+$ as given in the premise. Let t be a tower function defined by $t(0) = 0$ and $t(i+1) = t(i)^l + l$ for $i \in \mathbb{N}$. Define the inverse tower function as $t^{-1}(n) = \min\{i \mid t(i) \geq n\}$. Note that t^{-1} is polynomial-time computable. Now consider the algorithm $\mathcal{S}_c^g$ given below (Algorithm 1).

```
Input  : An arbitrary string $x \in \{0,1\}^*$, where $|x| = n$
Output: ACCEPT or REJECT
if $t^{-1}(|x|) < t^{-1}(|g(x)|)$ then ACCEPT iff $t^{-1}(|x|)$ is odd ;

// Compute the log derivatives of $G(x) = \{g^{(j)}(x)\}_{j \geq 0}$ at;
// $x_0 = x$, $x_1 = g(x)$ and $x_2 = g(g(x))$;
// by filling in the table $T$ along the diagonal lines;
$T[0,0] \leftarrow x$, $T[0,1] \leftarrow g(x)$, $T[1,0] \leftarrow logD(x, g(x))$;
for $i \leftarrow 0$ to $c$ do

    // Compute the values on the $(i+2)$-nd diagonal line of the table $T$;
    $T[0, i+2] \leftarrow g(T[0, i+1])$;
    if $t^{-1}(|T[0, i+1]|) < t^{-1}(|T[0, i+2]|)$ then
        ACCEPT iff $t^{-1}(|T[0, i+1]|)$ is even
    end
    for $j \leftarrow i+1$ to $0$ do
        $T[i+2-j, j] \leftarrow logD(T[i+1-j, j], T[i+1-j, j+1])$
    end

    // Accept or reject $x$ based on the computed log derivatives;
    // Here $T[i,0] = x_0^{(i)}$, $T[i,1] = x_1^{(i)}$, $T[i,2] = x_2^{(i)}$;
    $u \leftarrow T[i,0]$, $v \leftarrow T[i,1]$, $w \leftarrow T[i,2]$;
    if $u = v = w$ then ACCEPT iff $\lfloor \frac{T[i-1,0]}{2^{abs(u)+1}} \rfloor$ is odd;
    if $(u < v$ and $v \geq w)$ or $(u = v > w)$ then ACCEPT;
    if $(u > v$ and $v \leq w)$ or $(u = v < w)$ then REJECT;
end

ACCEPT iff $t^{-1}(|x|)$ is even
```

Algorithm 1: The Splitting Algorithm $\mathcal{S}_c^g$ based on log-derivative sequences.

Let $m = \lceil d 2^c \log^{(c-1)} |x| \rceil$, where d is some constant to be determined later. We first observe that Algorithm $\mathcal{S}_c^g$ queries on strings $g^{(j)}(x)$ for $1 \leq j \leq c+2$ only, each of which is polynomially bounded since g is a polynomially bounded function. Hence, $\mathcal{S}_c^g$ runs in polynomial time assuming the value of $g^{(j)}(u)$ for any u queried on can be obtained instantly.

We now turn to the proof for conditions (i) - (iii) of Theorem 4.2. Let x be an arbitrary input string and let X denote the sequence $\{x_j = g^{(j)}(x)\}_{0 \leq j \leq m}$. The algorithm $\mathcal{S}_c^g$ uses a matrix T to compute and store the log derivatives of the sequence X. More precisely, for each $i, j \geq 0$, $T[i,j]$, when and if computed, will contain the value of $x_j^{(i)}$, the i-th order log derivative at x_j of X. The algorithm $\mathcal{S}_c^g$ will fill T diagonally by computing all entries $T[i,j]$, where $i+j=0$, $i+j=1$, and so on, until $i+j=c+2$ if needed. It suffices to prove that there exist integers j_1 and j_2, where $\{j_1, j_2\} \subseteq [0,m]$ and $\mathcal{S}_c^g$ accepts $g^{(j_1)}(x)$ if and only if $\mathcal{S}_c^g$ rejects $g^{(j_2)}(x)$.

Now we consider the following cases.

Case 1: There exists $j_1 \in [1, m-1]$, where $t^{-1}(|x_{j_1}|) < t^{-1}(|x_{j_1+1}|)$. Let j_1 be the smallest such number. Then $\mathcal{S}_c^g$ accepts $x_{j_1} = x_{j_1}^{(0)}$ at Line 1 if and only if $t^{-1}(|x_{j_1}|)$ is odd.

Subcase 1(*a*): $t^{-1}(|x_{j_1-1}|) \geq t^{-1}(|x_{j_1}|)$. In this subcase $\mathcal{S}_c^g$ accepts $x_{j_1-1} = x_{j_1-1}^{(0)}$ at Line 12 if and only if $t^{-1}(|x_{j_1}|)$ is even. It follows that $\mathcal{S}_c^g$ accepts x_{j_1-1} if and only if $\mathcal{S}_c^g$ rejects x_{j_1}.

Subcase 1(*b*): $t^{-1}(|x_{j_1-1}|) < t^{-1}(|x_{j_1}|)$. In this subcase, $\mathcal{S}_c^g$ accepts $x_{j_1-1} = x_{j_1-1}^{(0)}$ at Line 1 if and only if $t^{-1}(|x_{j_1-1}|)$ is odd. Note that for each $j \in [1,m]$, it holds that

$$|x_j^{(0)}| = |x_j| \leq |x_{j-1}|^l + l = |x_{j-1}^{(0)}|^l + l$$

since $x_j = g(x_{j-1})$. This implies that $t^{-1}(|x_j|) \leq t^{-1}(|x_{j-1}|) + 1$ for each $j \in [1,m]$. Hence, it follows from the hypothesis of this subcase that $t^{-1}(|x_{j_1-1}| = t^{-1}(|x_{j_1}|) - 1$. Then we derive again in this case that $\mathcal{S}_c^g$ accepts x_{j_1-1} if and only if $\mathcal{S}_c^g$ rejects x_{j_1}.

Let $j_2 = j_1 - 1$. Then we have shown in both subcases (a) and (b) of Case 1 that $\mathcal{S}_c^g$ accepts x_{j_2} if and only if $\mathcal{S}_c^g$ rejects x_{j_1}, where $\{j_1, j_2\} \subseteq [0,m]$.

If Case 1 does not hold, then

$$\forall j \in [1, m-1], t^{-1}(|x_j|) \geq t^{-1}(|x_{j+1}|), \tag{15}$$

which implies

$$\forall j \in [0, m], t^{-1}(|x_j|) \leq t^{-1}(|x_1|) \leq t^{-1}(|x|) + 1. \tag{16}$$

We assume that statements (15) and (16) are both true for all the subsequent cases.

Case 2.*i*: For each $i \in [0,\ c]$, we consider Case 2.i in the increasing order of i, which consists of the following subcases $2.i(a) - 2.i(d)$.

If $i = 0$, let Z_0 be $X^{(0)} \backslash \{x\} = X \backslash \{x\}$, which is a consecutive subsequence of $X^{(0)} = X$ with start index $s_0 = 1$ and ending index $t_0 = m$, respectively. Otherwise, Z_i is a consecutive subsequence of $X^{(i)}$ constructed in Case $2.(i-1)(c)$ or $2.(i-1)(d)$ if applicable, with start index s_i and ending index t_i. Note the following statement:

Statement 4.3. *If Cases 2.i needs to be considered, then S_c^g does not accept or reject any string x_j, where $s_i \leq j \leq t_i - 2$, before the i-th iteration of the outer loop.*

Statement (4.3) is true for $i = 0$ in light of Case 1: We will consider Case 2.0 only if $\mathcal{S}_c^g$ does not accept any x_j for $1 \leq j \leq m-2$ at Line 1. We will see that Statement 4.3 holds true through all cases 2.i until the smallest i where $\mathcal{S}_c^g$ makes output during the i-th iteration of the outer loop on x_j for some $j \in [s_i, t_i - 2]$. In addition, if the execution of $\mathcal{S}_c^g$ reaches Line 20 during the i-th iteration on some input $x_j = g^{(j)}(x)$, then none of the elements $u = x_j^{(i)}$, $v = x_{j+1}^{(i)}$, and $w = x_{j+2}^{(i)}$ is ∞, for otherwise two elements among $\{x_k^{(i-1)} \mid j \le k \le j+3\}$ must equal each other since $x_j^{(i)} = logD(x_j^{(i-1)},\ x_{j+1}^{(i-1)})$ for every $i \ge 1$ and $j \ge 0$. That will make $\mathcal{S}_c^g$ halt in the $(i-1)$-st iteration of the outer loop already, at lines 21 - 23.

Subcase 2.i(a): Z_i contains 5 consecutive equal elements $\{x_j^{(i)}\}_{a \le j \le a+4}$, where $a \in [s_i,\ t_i - 4]$. Then for $a \le j \le a+2$, $\mathcal{S}_c^g$ accepts x_j at Line 21 if and only if $\lfloor \frac{x_j^{(i-1)}}{2^{abs(d)+1}} \rfloor$ is odd, where $d = x_a^{(i)} = logD(x_a^{(i-1)}, x_{a+1}^{(i-1)})$.

Define

$$E_i = \left\{ \lfloor \frac{x_j^{(i-1)}}{2^{abs(d)+1}} \rfloor \right\}_{a \le j \le a+2}.$$

Note that $logD(x_{j+1}^{(i-1)},\ x_j^{(i-1)}) = d$ for each $j \in [a, a+2]$. Then by Lemma 4.1, E_i contains at least one even number and one odd number. Therefore, $\mathcal{S}_c^g$ accepts at least one string x_{j_1} and rejects at least one string x_{j_2}, where $j_1, j_2 \in [a, a+2]$.

Now we assume that there don't exist 5 consecutive equal strings in Z_i.

Subcase 2.i(b): The sequence Z_i contains two elements $x_{j_1}^{(i)}$ and $x_{j_2}^{(i)}$, where $\{j_1, j_2\} \subseteq [s_i,\ t_i - 2]$ and

- $x_{j_1}^{(i)} < x_{j_1+1}^{(i)}$ and $x_{j_1+1}^{(i)} \ge x_{j_1+2}^{(i)}$, or $x_{j_1}^{(i)} = x_{j_1+1}^{(i)} > x_{j_1+2}^{(i)}$, and
- $x_{j_2}^{(i)} > x_{j_2+1}^{(i)}$ and $x_{j_2+1}^{(i)} \le x_{j_2+2}^{(i)}$, or $x_{j_2}^{(i)} = x_{j_2+1}^{(i)} < x_{j_2+2}^{(i)}$.

In this subcase Algorithm $\mathcal{S}_c^g$ accepts x_{j_1} at Line 22 and rejects x_{j_2} at Line 23.

Subcase 2.i(c): There does not exist $j_1 \in [s_i, t_i - 2]$ as required by Subcase $2.i(b)$, then both of the following hold for each $j \in [s_i, t_i - 2]$:

- If $x_j^{(i)} < x_{j+1}^{(i)}$, then $x_{j+2}^{(i)} < x_{j+1}^{(i)}$.
- If $x_j^{(i)} = x_{j+1}^{(i)}$, then $x_{j+2}^{(i)} \le x_{j+1}^{(i)}$.

This shows that Z_i is of the following form, where $s_i \le s_i' \le t_i' \le t_i$:

$$x_{s_i}^{(i)} > x_{s_i+1}^{(i)} > \cdots > x_{s_i'}^{(i)} = x_{s_i'+1}^{(i)} \cdots = x_{t_i'}^{(i)} < x_{t_i'+1}^{(i)} < \cdots < x_{t_i}^{(i)} \tag{17}$$

Hence, both $Y_{i_1} = \{x_{s_i}^{(i)}, x_{s_i+1}^{(i)}, \cdots, x_{s_i'}^{(i)}\}$ and $Y_{i_2} = \{x_{t_i'}^{(i)}, x_{t_i'+1}^{(i)}, \cdots, x_{t_i}^{(i)}\}$ are strictly monotonic consecutive subsequences of Z_i. We set $Y_i = Y_{i_1}$ if $\|Y_{i_1}\| \ge \|Y_{i_2}\|$, and $Y_i = Y_{i_2}$ otherwise. Then we define $Z_{i+1} = Y_i^{(1)}$, the log derivative sequence of Y_i, and proceed to Case 2.$(i+1)$.

Subcase 2.i(d): There does not exist $j_2 \in [0, m_i - 2]$ as required by Subcase 2.$i(b)$. This subcase is symmetric to Subcase 2.$i(c)$. we define Y_i and Z_{i+1} similarly and proceed to Case 2.$(i+1)$,

Assume that subcases 2.i(c) or 2.i(d) applies. If $i = 0$, then $s'_i = t'_i$ since $x_j \neq x_{j+1}$ for every $j \in [1, m-1]$. Hence,

$$\|Y_0\| \geq \lceil \|Z_0\|/2 \rceil = \lceil m/2 \rceil. \tag{18}$$

Otherwise, $t'_i - s'_i \leq 4$ due to Subcase 2.i(a), Hence,

$$\|Y_i\| \geq \lceil (\|Z_i\| - 4)/2 \rceil = \lceil \|Z_i\|/2 \rceil - 2 = \lceil (\|Y_{i-1}\| - 1)/2 \rceil - 2 \geq \lceil \|Y_{i-1}\|/2 \rceil - 3. \tag{19}$$

Summary of Case 2.i: In Case 2.i, we either find $\{j_1, j_2\} \subseteq [0, m-2]$, where $\mathcal{S}_c^g$ accepts x_{j_1} if and only if $\mathcal{S}_c^g$ rejects x_{j_2} (subcases 2.i(a) and 2.i(b)) or we obtain a strictly monotonic and consecutive subsequence Y_i of $X^{(i)}$ and if $i > 0$, of $Y_{i-1}^{(1)}$ as well, where $\|Y_i\|$ satisfies inequalities (18) and (19).

Assume now that none of the subcases 2.i(a) and 2.i(b) apply for all $0 \leq i \leq c$. Then we arrive at a set of sequences $\{Y_i\}_{0 \leq i \leq c}$ with the properties stated above.

We observe that the length of each string in $X^{(0)} = X$ is at most $n^l + l$ due to Equation (16). Hence, every element in Y_1 has an absolute value no more than $\log(2 \cdot 2^{(n^l+l+1)}) = O(n^l)$. This in turn implies that every element in Y_2 has an absolute value no more than $O(\log n)$ using the same argument. Continuing applying this argument on Y_3, Y_4,... through Y_c, we derive that every element in Y_c for $c \geq 2$ should have an absolute value no more than $O(\log^{(c-1)} n)$. Let $c_1 > 0$ be a constant such that every element in Y_c has an absolute value smaller than $c_1 \log^{(c-1)} n$.

By using a straightforward induction proof based on inequalities (18) and (19) we obtain that $\|Y_i\| \geq \lceil \frac{m}{2^{i+1}} - 6 \rceil$ for $0 \leq i \leq c$. It follows that

$$\|Y_c\| \geq \lceil \frac{m}{2^{c+1}} - 6 \rceil \geq \frac{m}{2^{c+2}} \geq \frac{d}{4} \log^{(c-1)} n.$$

Now let $d = 8c_1$. Then the maximal absolute value of elements in Y_c is no less than $\|Y_c\|/2 \geq c_1 \log^{(c-1)} n)$ since $\|Y_c\|$ is a strictly monotonic sequence of integers. This is a contradiction to our previous argument that every element in Y_c should have an absolute value smaller than $c_1 \log^{(c-1)} n$.

Hence, either subcase 2.$i(a)$ or 2.$i(b)$ must hold for some $i \in [0,\ c]$ if Case 1 does not hold. This will ensure that $\mathcal{S}_c^g$ accepts $x_{j_1} = g^{(j_1)}(x)$ if and only if $\mathcal{S}_c^g$ rejects $x_{j_2} = g^{(j_2)}(x)$ for some $\{j_1, j_2\} \subseteq [0, m]$. This proves (ii) and (iii) of Theorem 4.2

Regarding Condition (i) of the theorem, we observe that if Case 1 applies then $j_x \leq j_1$, where j_1 is the smallest number $j \in [1, m-1]$ such that $t^{-1}(|x_j|) < t^{-1}(|x_{j+1}|)$. This implies that $t^{-1}(|x_j|) \leq t^{-1}(|x_1|)$, for each $j \in [1, j_1]$. Hence, it follows that $|x_j| \leq |x_1|^l + l$ for each $j \in [1, j_x]$. Note that $|x_1| \leq |x|^l + l$ since $x_1 = g(x_0) = g(x)$. So for each $j \in [0, j_x]$, $|g^{(j)}(x)| \leq (|x|^l + l)^l + l \leq 2|x|^{2l}$. Therefore, Condition (i) holds for $r(n) = 2n^{2l}$.

Now assume that Case 1 does not apply. Then by Equation (16), for each $j \in [0, m]$, it holds that $t^{-1}(|x_j|) \leq t^{-1}(|x_1|) \leq t^{-1}(|x|) + 1$, or equivalently, $|g^{(j)}(x)| = |x_j| \leq (|x|^l + l)^l + l \leq 2|x|^{2l}$. So we again show that Condition (i) holds for $r(n) = 2n^{2l}$.

This finishes the proof of Theorem 4.2.

□

With Theorem 4.2 we can now establish the rest of our main results.

Theorem 4.4. *For every $k \in \mathbb{N}^+$ and positive integer $c \geq 2$, if a non-trivial language L is $\leq^p_{k\text{-}dtt}$-autoreducible, then L is weakly $\leq^p_{k^{O(2^c \log^{(c-1)} n)}\text{-}tt}$-mitotic.*

Proof. We assume that $k \geq 2$ since it is already known that a non-trivial language is $\leq^p_{1\text{-}tt}$ autoreducible if and only if it is $\leq^p_{1\text{-}tt}$ mitotic [9].

Let L be a non-trivial and $\leq^p_{k\text{-}dtt}$-autoreducible language. Then there exists a polynomial-time computable function f, where $f(w) = \langle u_0, u_1, \ldots, u_{k-1}\rangle$ for each $x \in \Sigma^*$ such that

- for each $i \in [0, k-1]$, $x \neq u_i$, and
- $x \in L$ if and only if $\exists i \in [0, k-1]$, $u_i \in L$.

When there is no confusion we also use $f(x)$ to denote the set $\{u_0, u_1, \ldots, u_{k-1}\}$.

Let *min* (W) denote the mininal element of a set W of numbers or the *lexicographically least* string if W is a set of strings.

Now define

$$g(x) = \begin{cases} \mathit{min}\ (f(x)) & \text{if } x \notin L, \\ \mathit{min}\ (f(x) \cap L) & \text{if } x \in L. \end{cases}$$

Assume that f is computable in time $n^l + l$ for some $l \in \mathbb{N}^+$. Then g is a polynomially bounded function. In particular, it holds for every $x \in \Sigma^{=n}$ that $|g(x)| \leq n^l + l$. It is also clear that $g(x) \neq x$ for every $x \in \Sigma^*$. Hence we can apply Theorem 4.2 on function g and any positive integer $c \geq 2$ to obtain an algorithm $\mathcal{S}^g_c$, a polynomial r, and a constant $d > 0$ that satisfy all the conditions as stated in Theorem 4.2.

Now let $S = L(\mathcal{S}^g_c)$. We use $QT(f, x, h)$ to denote the *query tree* of the dtt reduction f rooted at x with height h, where the root x has one child node for each string in $f(x)$, each child node y of x has a child node of its own for each string in $f(y)$, and so on. It is clear that $x \in L$ if and only if $QT(f, x, h)$ contains a path of length at least 1 from the root to a leaf consisting only of nodes corresponding to strings in L.

Now we consider Algorithm 2, where $c \geq 2$ is an integer.

```
    Input  : An arbitrary string $x \in \Sigma^*$, where $|x| = n$
    Output: ACCEPT or REJECT
 1  $m \leftarrow \lceil d2^c \log^{(c-1)} n \rceil$;
 2  if $x \in S \cap L$ then ACCEPT;
 3  for each path $p$ from the root to a leaf in $QT(f, w, m)$ do
 4      Let $p = \langle x_0 = x, x_1, \cdots, x_m \rangle$;
 5      for $j \leftarrow 1$ to $m$ do
 6          if $|x_j| > r(|x|)$ then exit the $j$ loop;
 7          if $x_j \in S \cap L$ then ACCEPT;
 8      end
 9  end
10  REJECT
```
Algorithm 2: The bounded dtt reduction from L to $S \cap L$

We first observe that the number of nodes in $QT(f, x, m)$, where $|x| = n$ and $m = \lceil d2^c \log^{(c-1)} n \rceil$, is at most $k^{m+1} = k^{O(2^c \log^{(c-1)} n)}$. Also, Algorithm 2 on input x only queries on nodes in $Q(f, x, m)$ corresponding to strings of length at most $r(|x|)$ due to line 6. Hence, Algorithm 2 makes $k^{O(2^c \log^{(c-1)} n)}$ queries to $S \cap L$ and runs in polynomial time. Note also that all queries made by Algorithm 2 can be computed using f and x only.

Consider $x \in \Sigma^*$, where $|x| = n$. If $x \notin L$, then $x \notin S \cap L$. In addition, none of the strings in $QT(f, w, m)$ belongs to L since otherwise $x \in L$ due to that f is a $\leq^p_{k\text{-}dtt}$-autoreduction on L. Hence, Algorithm 2 rejects x at line 10.

Now assume that $x \in L$. If $x \in S \cap L$, then Algorithm 2 accepts x at Line 2. Otherwise, $x \notin S$. Note that $\{g^{(j)}(x)\}_{0 \leq j \leq m}$ forms a path in $QT(f, x, m)$ and each $g^{(j)}(x) \in L$. By Theorem 4.2, there exists $j_x \in [1, m]$ such that $g^{(j_x)}(x) \in S$ and for each $j \in [0, j_x - 1]$, $|g^{(j)}(x)| \leq r(|x|)$ and $g^{(j)} \notin S$. Hence, Algorithm 2 accepts x in Line 7 when and if $g^{(j_x)}(x)$ is examined in the j loop. This shows that $L \leq^p_{k^{O(2^c \log^{(c-1)} n)}\text{-}dtt} S \cap L$.

We now show $S \cap L \leq^p_{k^{O(c)}\text{-}tt} L$ using Algorithm 3.

```
    Input  : An arbitrary string $x \in \{0,1\}^*$, where $|x| = n$
    Output: ACCEPT or REJECT
 1  if $x \notin L$ then REJECT ;
 2  Compute all strings $u \in QT(f, x, c+2)$ and determine whether $u \in L$ by querying $L$;
 3  Run $\mathcal{S}_c^g$ on $x$ and use the information obtained in Line 2 to determine the value of $g(u)$
      when and if needed;
 4  ACCEPT iff $\mathcal{S}_c^g$ accepts $x$
```
Algorithm 3: The bounded tt reduction from $S \cap L$ to L

Clearly Algorithm 3 can correctly simulate Algorithm $\mathcal{S}_c^g$ on x and decides $S \cap L$ if the value of each $g^{(j)}(x)$ for $1 \leq j \leq c+2$ is available. Note that all those values appear in $QT(f, x, c+2)$. Hence, Algorithm 3 obtains values of all $g^{(j)}(x)$ for $1 \leq j \leq c+2$ by querying L on all strings in $QT(f,x,c+2)$ at Line 2. Since there are at most k^{c+3} strings in $QT(f,x,c+2)$ and g is a polynomially bounded function, Algorithm 3 makes no more than $k^{c+3} = k^{O(c)}$ queries to L and runs in polynomial time considering that k and c are constants. In addition, all queries made by Algorithm 3 can be computed with f and x. This shows that $S \cap L \leq^p_{k^{O(c)}\text{-}tt} L$ and hence $L \equiv^p_{k^{O(2^c \log^{(c-1)} n)}\text{-}tt} S \cap L$.

Similarly we can show $L \equiv^p_{k^{O(2^c \log^{(c-1)} n)}\text{-}tt} \overline{S} \cap L$ by using Algorithm $\overline{\mathcal{S}}^g_c$ instead, which is the same as $\mathcal{S}^g_c$ except the output is swapped, i.e., $\overline{\mathcal{S}}^g_c$ accepts if and only if $\mathcal{S}^g_c$ rejects on any input.

It remains to show that $S \cap L \equiv^p_{k^{O(2^c \log^{(c-1)} n)}\text{-}tt} \overline{S} \cap L$. We observe that changing the 'ACCEPT' output to 'REJECT' in Line 2 of Algorithm 2 gives a $\leq^p_{k^{O(2^c \log^{(c-1)} n)}\text{-}dtt}$ reduction from $\overline{S} \cap L$ to $S \cap L$. By symmetry the same reduction shows that $S \cap L \leq^p_{k^{O(2^c \log^{(c-1)} n)}\text{-}dtt} \overline{S} \cap L$. This finishes the proof of Theorem 4.4.

□

Using a similar argument we prove the same result as Theorem 4.4 for k-ctt autoreducible sets:

Theorem 4.5. *For every $k \in \mathbb{N}^+$ and integer $c \geq 2$, if a non-trivial language L is $\leq^p_{k\text{-}ctt}$-autoreducible, then L is weakly $\leq^p_{k^{O(2^c \log^{(c-1)} n)}\text{-}tt}$-mitotic.*

Proof. The proof is similar to that of Theorem 4.4. We again assume that $k \geq 2$ for the same reason. Let L be a non-trivial and $\leq^p_{k\text{-}ctt}$-autoreducible language. Then there exists a polynomial-time computable function f', where $f'(x) = \langle u_0, u_1, \ldots, u_{k-1} \rangle$ for each $x \in \Sigma^*$ such that

- for each $i \in [0, k-1]$, $x \neq u_i$, and
- $x \in L$ if and only if $\forall i \in [0, k-1]$, $u_i \in L$.

We need to use the following function g' instead of the function g used in the proof of Theorem 4.4:

$$g'(x) = \begin{cases} min\ (f'(x)) & \text{if } x \in L \\ min\ (f'(x) \cap \overline{L}) & \text{if } x \notin L \end{cases}$$

Then we can apply Theorem 4.2 on g' and any positive integer $c' \geq 2$ to obtain an algorithm $\mathcal{S}^{g'}_{c'}$, a polynomial r', and a constant $d' > 0$ that satisfy all the conditions as stated in Theorem 4.2.

In the same way as we used algorithm $\mathcal{S}^g_c$, polynomial r and constant d in the proof of Theorem 4.4, we use Algorithm $\mathcal{S}^{g'}_{c'}$, polynomial r' and constant d' to construct bounded tt reductions among L, $S' \cap L$, and $\overline{S'} \cap L$, where $S' = L(\mathcal{S}^{g'}_c)$. Those reductions will show that L is weakly $\leq^p_{k^{O(2^c \log^{(c-1)} n)}\text{-}tt}$-mitotic. We give the reductions between L and $S' \cap L$ below and omit the details of the rest of the proof.

□

```
Input  : An arbitrary string $x \in \Sigma^*$, where $|x| = n$
Output: ACCEPT or REJECT
1 $m \leftarrow \lceil d' 2^{c'} \log^{(c'-1)} n \rceil$
2 for each path $p'$ from the root to a leaf in $QT(f', x, m)$ do
3     Let $p = \langle x_0 = x, x_1, \cdots, x_m \rangle$;
4     for $j \leftarrow 0$ to $m$ do
5         if $|x_j| > r'(|x|)$ then REJECT;
6         if $x_j \in S' \cap L$ then exit the $j$ loop;
7     end
8 end
9 ACCEPT
```

Algorithm 4: The bounded ctt reduction from L to $S' \cap L$

```
Input  : An arbitrary string $x \in \{0,1\}^*$, where $|x| = n$
Output: ACCEPT or REJECT
1 if $x \notin L$ then REJECT ;
2 Compute all strings $u \in QT(f, x, m)$ with $|u| \leq r^{(2)}(n)$, and determine whether $u \in L$ by
    querying $L$;
3 Run $\mathcal{S}_c^{g'}$ on $x$ and use the information obtained in Line 2 to determine the value of $g(u)$
    when and if needed.;
4 ACCEPT iff $\mathcal{S}_c^{g'}$ accepts $x$
```

Algorithm 5: The bounded tt reduction from $S' \cap L$ to L

In light that the k-dtt complete sets of many common complexity classes have been proven to be k-dtt autoreducible for $k \geq 2$ [8, 5] we have the following corollary providing a better understanding of (weak) mitoticity of complete sets in complexity theory.

Corollary 4.6. *Let $k, c \geq 2$ be both integers. Then every k-dtt complete set for the following classes is weakly $k^{O(2^c \log^{(c-1)} n)}$-tt mitotic:*

- PSPACE,
- *the levels* Σ_i^P, Π_i^P *and* Π_i^P *of the polynomial-time hierarchy for* $i \geq 2$
- 1NP,
- *the levels of the Boolean hierarchy over* NP,
- *the levels of the* MODPH *hierarchy, and*

In addition, every k-dtt or k-ctt complete set for NEXP *is weakly $k^{O(2^c \log^{(c-1)} n)}$-tt mitotic.*

Proof. Glaßer et al. [8] showed that all k-dtt complete sets of the complexity classes listed above except NEXP are k-dtt autoreducible. In addition, Glaßer et al. [5] recently showed that all k-dtt and k-ctt complete sets for NEXP are k-dtt and k-ctt autoreducible, respectively. The corollary follows immediately by applying theorems 4.4 and 4.5.

□

References

[1] K. Ambos-Spies. On the structure of the polynomial time degrees of recursive sets. Habilitationsschrift, Zur Erlangung der Venia Legendi Für das Fach Informatik an der Abteilung Informatik der Universität Dortmund, September 1984.

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

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

[4] C. Glaßer, A. Selman, S. Travers, and L. Zhang. Non-mitotic sets. *Theoretical Computer Science*, 410(21-23):2011–2033, 2009.

[5] C. Glaßer, D. Nguyen, C. Reitwießer, A. Selman, and M. Witek. Autoreducibility of complete sets for log-space and polynomialtime reductions. In *Proceedings 40th International Colloquiumon on Automata, Languages and Programming*, number 7965 in Lecture Notes in Computer Science, pages 473–484. Springer Verlag, 2013.

[6] C. Glaßer, D. Nguyen, A. Selman, and M. Witeck. Introduction to autoreducibility and mitoticity. In A. Day et al., editor, *Computability and Complexity, Essays Dedicated to Rodney G. Downey on the Occasion of His 60th Birthday*, volume 10010 of *Lecture Notes in Computer Science*, pages 56–78. Springer, 2017.

[7] C. Glaßer, M. Ogihara, A. Pavan, A. Selman, and L. Zhang. Autoreducibility and mitoticity. *ACM SIGACT News*, 40(3):60–76, 2009.

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

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

[10] L. Hemaspaandra and M. Ogihara. *The Complexity Theory Companion.* Springer, 2002.

[11] S. Homer and A. Selman. *Computability and Complexity Theory.* Texts in Computer Science. Springer, New York, 2e edition, December 2011.

[12] B. Trakhtenbrot. On autoreducibility. *Dokl. Akad. Nauk SSSR*, 192(6):1224–1227, 1970. Translation in Soviet Math. Dokl. 11(3): 814C817, 1970.

[13] A. Yao. Coherent functions and program checkers. In *Proceedings of the 22nd Annual Symposium on Theory of Computing*, pages 89–94, 1990.