Weak Mitoticity of Bounded Disjunctive and Conjunctive Truth-Table Autoreducible Sets

Liyu Zhang(✉), Mahmoud Quweider, Hansheng Lei, and Fitra Khan

Department of Computer Science, University of Texas Rio Grande Valley, One West University Boulevard, Brownsville, TX 78520, USA
{liyu.zhang,mahmoud.quweider,hansheng.lei,fitra.khan}@utrgv.edu

Abstract. Glaßer et al. (SIAMJCOMP 2008 and TCS 2009 (The two papers have slightly different sets of authors)) 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.

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 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*"

L. Zhang—Research supported in part by NSF CCF grant 1218093.

© Springer International Publishing AG, part of Springer Nature 2018
L. Wang and D. Zhu (Eds.): COCOON 2018, LNCS 10976, pp. 192–204, 2018.
https://doi.org/10.1007/978-3-319-94776-1_17

liyu.zhang@utrgv.edu

[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 dtt- and/or ctt-mitotic, respectively.

We give definitions and notations needed to present our results in Sect. 2 below. We then describe our main results in more details in Sect. 3. Due to space limit we have to omit the proofs for Lemma 1, Theorems 4, and 5 in this proceeding paper. Those proofs will be available upon request and in the journal version of the paper.

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 ($\leq_T^p$) [10,11]. 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

liyu.zhang@utrgv.edu

every function f, we use $f^{(i)}(x)$ to denote $\underbrace{f(f(\cdots f}_{i}(x)))$ for every $i \in \mathbb{N}$, where $f^{(0)}(x) = x$.

Throughout the paper, we use the two terms *Turing machines* and *algorithms* interchangeably. Following Glaßer et al. [9], we define a non-trivial set to be a set L where both $L|$ and $\overline{L}$ contain at least two distinct elements. This allows us present our results in a simple and concise way. 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. 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 1. *Define a language A to be* polynomial-time truth-table 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. *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* disjunctive/conjunctive truth-table reduction *and A is said to be* disjunctive/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 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 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 $\overline{S}$ 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.

liyu.zhang@utrgv.edu

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

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

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 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 of X.*

3 Results

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 a stronger result that there

liyu.zhang@utrgv.edu

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 shows 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 $L \not\leq^p_{2\text{-}tt} L_2$. In order to prove that L is not 2-tt mitotic 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 $L_2 \not\leq^p_{2\text{-}tt} L$. The latter is clearly a weaker statement. In light of this observation, we adapt the previous proof by considering *three* oracle Turing machines M_i, M_j and M_k instead for the purpose of diagonalization 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 1. *There exists* $L \in \mathrm{SPARSE} \cap \mathrm{EXP}$ *such that*

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

Theorem 1 indicates that 2-tt autoreducibility does not imply weak mitoticity even 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 ($\leq^p_m$) 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

liyu.zhang@utrgv.edu

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, $f(x)$, $f(f(x))$, ..., 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 $x < f(x) > f(f(x))$ and to $\overline{S}$ if $x > f(x) < f(f(x))$. 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

liyu.zhang@utrgv.edu

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 Sect. 2, of the sequence consisting of strings on the trajectory of x, i.e., $X = \{x_j = g^{(j)}\}_{j \geq 1}$. 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].

Lemma 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.*

Theorem 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, and a polynomial r, where for every $x \in \Sigma^*$ an integer $j_x \in [1, \lceil 2^c \log^{(c-1)} |x| \rceil]$ 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).

Let $m = \lceil 2^c \log^{(c-1)} |x| \rceil$. 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 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

liyu.zhang@utrgv.edu

Input : An arbitrary string $w \in \{0,1\}^*$, where $|w| = n$
Output : ACCEPT or REJECT

1 $m \leftarrow k\lfloor \log n \rfloor$;
2 $D[0,0] \leftarrow x$, $D[0,1] \leftarrow g(x)$;
3 **if** $t^{-1}(|D[0,0]|) < t^{-1}(|D[0,1]|)$ **then**
4 | ACCEPT iff $t^{-1}(|D[0,0]|)$ is odd
5 **end**
6 $D[1,0] \leftarrow logD(D[0,0], D[0,1])$;
7
8 // Compute the log derivatives of X at $x_0 = x$, $x_1 = g(x)$ and $x_2 = g(g(x))$;
9 **for** $i \leftarrow 0$ **to** k **do**
10 |
11 | // Compute the i-th order log derivatives;
12 | **for** $j \leftarrow i+2$ **to** 0 **do**
13 | | **if** $j = i+2$ **then**
14 | | | $D[0,j] \leftarrow g(D[0,j-1])$;
15 | | | **if** $t^{-1}(D[0,j-1])) < t^{-1}(|D[0,j]|)$ **then**
16 | | | | ACCEPT iff $t^{-1}(|D[0,j-1]|)$ is even
17 | | | **end**
18 | | **end**
19 | | **else**
20 | | | $D[i+2-j,j] \leftarrow logD(D[i+1-j,j+1], D[i+1-j,j])$
21 | | **end**
22 | **end**
23 |
24 | // Accept or reject x based on the computed log derivatives;
25 | // Here $D[i,0] = x_0^{(i)}$, $D[i,1] = x_1^{(i)}$, $D[i,2] = x_2^{(i)}$;
26 | $u \leftarrow D[i,0]$, $v \leftarrow D[i,1]$, $w \leftarrow D[i,2]$;
27 | **if** $u = v = w$ **then**
28 | | ACCEPT iff $\lfloor \frac{D[i-1,0]}{2^{abs(u)}} \rfloor$ is odd
29 | **end**
30 | **if** $u < v \geq w$ *or* $u = v > w$ **then** ACCEPT;
31 | **if** $u > v \leq w$ *or* $u = v < w$ **then** REJECT;
32 **end**
33
34 ACCEPT iff $isEvenStage$;

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

algorithm $\mathcal{S}_c^g$ uses an array D to compute and store the log derivatives of the sequence X. More precisely, every time the execution of the algorithm $\mathcal{S}_c^g$ reaches Line 26, $D[p,q]$ will contain the value of $x_q^{(p)}$, the p-th order log derivative of X at x_q, for each $p \in [0,i]$ and $q \in [0,i+2]$, where $p+q \leq i+2$. In particular, $D[i,0]$, $D[i,1]$ and $D[i,2]$ will store the values of $x_0^{(i)}$, $x_1^{(i)}$ and $x_2^{(i)}$, in that order.

liyu.zhang@utrgv.edu

We consider the following cases in that order so that the proof for Case e assumes that none of the cases $1, 2, \cdots, e-1$ holds. We also assume that $t^{-1}(|x|)$ is even.

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 4 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 16 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 4 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

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

$$\forall j \in [0, m], t^{-1}(|x_j|) \leq t^{-1}(|x_1|) \leq t^{-1}(|x|) + 1 \tag{2}$$

We assume that statements (1) and (2) is 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 $2.i(a-d)$.

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 3. *If Cases* $2.i$ *needs to be considered, then* S_c *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 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 $x_j = x_j^{(0)}$ for $1 \leq j \leq m-2$ at Line 4. We will see that Statement 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 $x_j = x_j^{(0)}$ for some $j \in [s_i, t_i - 2]$. In addition, if the execution of $\mathcal{S}_c^g$ reaches Line 26 during the i-th

liyu.zhang@utrgv.edu

iteration, then none of the elements $u = x_0^{(i)}$, $v = x_1^{(i)}$, and $w = x_2^{(i)}$ is ∞, for otherwise two elements among $\{x_j^{(i-1)} \mid 0 \le j \le 3\}$ must equal each other since $x_j^{(i)} = logD(x_{j+1}^{(i-1)},\ x_j^{(i-1)})$ for every $j \in [0, m_i - i]$. That will make $\mathcal{S}_c^g$ halt in the $(i-1)$-st iteration of the outer loop already, at lines 29–31.

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 29 if and only if $\lfloor \frac{x_j^{(i-1)}}{2^{abs(d)+1}} \rfloor$ is odd.

Define

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

Note that $x_j^{(i)} = logD(x_{j+1}^{(i-1)},\ x_j^{(i-1)})$ for each $j \in [a, a+2]$. Then by Lemma 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

- $x_{j_1}^{(i)} < x_{j_1+1}^{(i)} \ge x_{j_1+2}^{(i)}$ or $x_{j_1}^{(i)} = x_{j_1+1}^{(i)} < x_{j_1+2}^{(i)}$, and
- $x_{j_2}^{(i)} > x_{j_2+1}^{(i)} \le x_{j_2+2}^{(i)}$ 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 30 and rejects x_{j_2} at Line 31.

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 string $j \in [s_i, t_i - 2]$:

- 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 $x_{j+2}^{(i)} \ge x_{j+1}^{(i)}$.

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

$$x_{s_i}^{(i)} < x_{s_i+1}^{(i)} < \cdots < x_{s_i'}^{(i)} = x_{s_i'+1}^{(i)} \cdots = x_{t_i'}^{(i)} < x_{t_i'+1}^{(i)} < \cdots < x_{t_i}^{(i)} \tag{3}$$

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. Note that Y_i is a consecutive subsequence of $Y_{i-1}^{(1)}$, the log derivative sequence of Y_{i-1}.

If $i = 0$, Subcase $2.i$(a) does not apply since $x_j \ne x_{j-1}$ for every $j \in [1, m]$. Consequently $|Y_0| \ge m_0 = \lceil m/2 \rceil$. Otherwise, $t_i' - s_i' \le 4$ due to Subcase $2.i$(a) and Eq. (3). Assume $|Y_{i-1}| \ge m_{i-1}$. Then $|Z_i| = |Y_{i-1}| - 1$. Hence, $|Y_i| \ge m_i = \lceil ((m_{i-1} - 1) - 3)/2 \rceil = \lceil m_{i-1}/2 \rceil - 2$.

liyu.zhang@utrgv.edu

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 again obtain a strictly monotonic consecutive subsequence Y_i of length at least m_i within Z_i. We let Y_i be that subsequence with starting index s_i and ending index t_i. Again, Y_i is a subsequence of the log derivative sequence of Y_{i-1}.

For both subcases 2.i(c) and 2.i(d) we define $Z_{i+1} = Y_i^{(1)}$ and proceed to Case 2.$(i + 1)$. Clearly Z_{i+1} is a consecutive subsequence of $X^{(i+1)}$ since Y_i is a consecutive subsequence of $X^{(i)}$. Also, the start and ending indices of Z_{i+1} are $s_{i+1} = s_i$ and $t_{i+1} = s'_i - 1$, or $s_{i+1} = t'_i$ and $t_{i+1} = t_i - 1$, respectively, depending on how Y_i is formed in Case 2.i(c) or Case 2.i(d).

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 both $X^{(i)}$ and Y_{i-1}, where $|Y_i| \geq m_i$.

If 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, where

- Y_0 is a strictly monotonic and consecutive subsequence of $X^{(0)}$ with $|Y_0| \geq m_0 = \lceil m/2 \rceil$, and
- for each $i \in [1,\ c]$, Y_i is a strictly monotonic and consecutive subsequence of both $X^{(i)}$ and $Y_{i-1}^{(1)}$ with $|Y_i| \geq m_i = \lceil m_{i-1}/2 \rceil - 2$

Note that the length of each string in $X^{(0)} = X$ is at most $n^l + l$ due to Eq. 2. Hence, every element in Y_1 has an absolute value no more than $\log(2 \cdot 2^{(n^l+l+1)}) = O(n^l)$ for sufficiently large n. 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 obtain that every element in Y_c for $c \geq 2$ should have an absolute value no more than $O(\log^{(c-1)} n)$.

However, a simple induction proof shows that $m_c = \Omega(\log^{(c)} n)$. The maximal absolute value of elements in Y_c is no less than $m_c/2 = \Omega(\log^{(c)} n)$, since Y_c is a strictly monotonic sequence of integers of length at least m_c. This is a contradiction to the argument above that every element in Y_c for $c \geq 2$ should have an absolute value no more than $O(\log^{(c-1)} n)$. 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 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 $j \in [0, j_x]$, $|g^{(j)}(x)| \leq |x|^l + l$.

Now assume that Case 1 does not apply. Then by Eq. (2), 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, $|g^{(j)}(x)| = |x_j| \leq (|x|^l + l)^l + l \leq 2|x|^{2l}$. Therefore, Condition (i) holds for $r(n) = 2n^{2l}$.

This finishes the proof of Theorem 2. □

With Theorem 2 we can now establish the rest of our main results.

liyu.zhang@utrgv.edu

Theorem 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 $\{u_0, u_1, \ldots, u_{k-1}\}$. For a set of strings W, let $lex - min\ (W)$ denote the *lexicographically least* string in W.

Now define

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

It's clear that function g is polynomial bounded and for every $x \in \Sigma^*$, $g(x) \neq x$. Hence we can apply Theorem 2 on function g and any positive integer $c \geq 2$ to obtain an algorithm $\mathcal{S}^g_c$ and a polynomial r that satisfies all the conditions as stated in Theorem 2. Let $S = L(\mathcal{S}^g_c)$. Then one can argue that S can be used to show that

$$S \cap L \equiv^p_{k^{O(c)}\text{-}tt} L \equiv^p_{k^{O(2^c \log^{(c-1)} n)}\text{-}tt} \overline{S} \cap L.$$

□

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

Theorem 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.*

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

Corollary 1. *For every integer $k \geq 2$ and $c \geq 2$, every k-dtt complete set for the following classes is weakly $k^{O(2^c \log^{(c-1)} n)}$-dtt mitotic:*

- PSPACE,
- *the levels* $\Sigma^{\mathrm{P}}_{\mathrm{h}}$, $\Pi^{\mathrm{P}}_{\mathrm{h}}$ *and* $\Pi^{\mathrm{P}}_{\mathrm{h}}$ *of the polynomial-time hierarchy for* $h \geq 2$
- 1NP,
- *the levels of the Boolean hierarchy over* NP,
- *the levels of the* MODPH *hierarchy, and*
- NEXP.

liyu.zhang@utrgv.edu

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 any k-dtt complete set for NEXP is k-dtt autoreducible. The corollary follows immediately by applying Theorem 4.

□

References

1. Ambos-Spies, K.: 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. Buhrman, H., Fortnow, L., van Melkebeek, D., Torenvliet, L.: Using autoreducibility to separate complexity classes. SIAM J. Comput. **29**(5), 1497–1520 (2000)
3. Buhrman, H., Torenvliet, L.: A Post's program for complexity theory. Bull. EATCS **85**, 41–51 (2005)
4. Glaßer, C., Selman, A., Travers, S., Zhang, L.: Non-mitotic sets. Theoret. Comput. Sci. **410**(21–23), 2011–2033 (2009)
5. Glaßer, C., Nguyen, D.T., Reitwießner, C., Selman, A.L., Witek, M.: Autoreducibility of complete sets for log-space and polynomial-time reductions. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013. LNCS, vol. 7965, pp. 473–484. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39206-1_40
6. Glaßer, C., Nguyen, D.T., Selman, A.L., Witek, M.: Introduction to autoreducibility and mitoticity. In: Day, A., Fellows, M., Greenberg, N., Khoussainov, B., Melnikov, A., Rosamond, F. (eds.) Computability and Complexity. LNCS, vol. 10010, pp. 56–78. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-50062-1_5
7. Glaßer, C., Ogihara, M., Pavan, A., Selman, A., Zhang, L.: Autoreducibility and mitoticity. ACM SIGACT News **40**(3), 60–76 (2009)
8. Glaßer, C., Ogihara, M., Pavan, A., Selman, A.L., Zhang, L.: Autoreducibility, mitoticity, and immunity. J. Comput. Syst. Sci. **73**, 735–754 (2007)
9. Glaßer, C., Pavan, A., Selman, A., Zhang, L.: Splitting NP-complete sets. SIAM J. Comput. **37**(5), 1517–1535 (2008)
10. Hemaspaandra, L., Ogihara, M.: The Complexity Theory Companion. Springer, Heidelberg (2002). https://doi.org/10.1007/978-3-662-04880-1
11. Homer, S., Selman, A.: Computability and Complexity Theory. Texts in Computer Science, 2nd edn. Springer, New York (2011). https://doi.org/10.1007/978-1-4614-0682-2
12. Trakhtenbrot, B.: On autoreducibility. Dokl. Akad. Nauk SSSR **192**(6), 1224–1227 (1970). Transl. Soviet Math. Dokl. **11**(3), 814–817 (1790)
13. Yao, A.: Coherent functions and program checkers. In: Proceedings of the 22nd Annual Symposium on Theory of Computing, pp. 89–94 (1990)

liyu.zhang@utrgv.edu