Inf. Process. Lett. 186 (2024) 106472

Contents lists available at ScienceDirect

Information Processing Letters

journal homepage: www.elsevier.com/locate/ipl

# Splitting NP-complete sets infinitely

Liyu Zhang *, Mahmoud Quweider, Fitra Khan, Hansheng Lei

*Department of Informatics and Engineering Systems, University of Texas Rio Grande Valley, One West University Boulevard, Brownsville, TX, 78520, USA*

ARTICLE INFO

*Keywords:*
Computational complexity
NP-complete sets
Effectively infinite splittability
Polynomial-time autoreducibility
Polynomial-time mitoticity

ABSTRACT

Glaßer et al. (SIAMJCOMP 2009 and TCS 2009) proved that NP-complete languages are polynomial-time mitotic for the many-one reduction, meaning that each NP-complete language $L$ can be split into two NP-complete languages $L \cap S$ and $L \cap \overline{S}$, where $S$ is a language in P. It follows that every NP-complete language can be partitioned into an arbitrary *finite* number of NP-complete languages. We strengthen and generalize this result by showing that every NP-complete language can be partitioned into *infinitely* many NP-complete languages. Furthermore those NP-complete languages resulting from such partitioning can be *effectively presented*.

## 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 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. We say that a language $L$ is $r$-*mitotic* if $L$ can be partitioned by a polynomial-time decidable set $S$ into to two subsets, both of which are equivalent to $L$ for the reduction $r$, i.e., $L \equiv_r^p L \cap S \equiv_r^p L \cap \overline{S}$. Autoreducibility and mitoticity are related but different computational properties of languages. They coincide for the many-one and 1-tt reduction [8] but not for the 2-tt or any weaker reduction [4] among polynomial-time reductions. Complete sets of many complexity classes are known to be autoreducible, mitotic, or both under various reductions while several important problems remain open [1,6,5].

Polynomial-time autoreducibilities and mitoticities gained attention due to their candidacies as structural properties 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 [1]. We refer the reader to Glaßer et al. [6] and Glaßer et al. [5] for surveys along this line of research.

In this paper we strengthen one of the main results by Glaßer et al. [8] that every NP-complete set is many-one-mitotic, i.e., can be split into two NP-complete subsets by a polynomial-time decidable set. Though the result by Glaßer et al. does imply that every NP-complete sets can be split by a polynomial-time decidable set into $k$ NP-complete subsets for every $k \geq 1$, it does not provide a uniform way to construct nondeterministic Turing machines that accepts those NP-complete subsets. Our results provide such uniform constructions that can be generalized to an infinite number of subsets.

The key observations used by our proof are that the *left set* of a nondeterministic polynomial-time Turing machine $N$ is structurally similar to that of the same machine with a deterministic polynomial-time decider $D$ attached to the end of every computation path. This similarity can be utilized to construct an m-autoreduction of the language $L(N) \cap L(D)$ in a uniform way (see Theorem 3.7 and Theorem 3.2), which in turn can be used to show the m-mitoticity of $L(N) \cap L(D)$. Our constructions are also recursive in nature. The m-autoreductions we constructed for NP-complete subsets of the given NP-complete language were used recursively to construct m-autoreductions for finer-grained NP-complete subsets of the same NP-complete language. We consider our result a significant extension of Glaßer et al. [8] rather than a straightforward adaption.

We provide needed definitions and notations in Section 2 and state our results and proofs in Section 3.

## 2. Definitions and notations

We assume familiarity with basic notions in complexity theory and particularly, common complexity classes P and NP, and polynomial-time reductions [10,9]. Without loss of generality, we use the alphabet

* Corresponding author.
*E-mail addresses:* liyu.zhang@utrgv.edu (L. Zhang), mahmoud.quweider@utrgv.edu (M. Quweider), fitra_khan@yahoo.com (F. Khan), hansheng.lei@utrgv.edu (H. Lei).

https://doi.org/10.1016/j.ipl.2024.106472
Received 20 June 2023; Received in revised form 6 January 2024; Accepted 9 January 2024
Available online 22 January 2024
0020-0190/© 2024 Elsevier B.V. All rights reserved.

$\Sigma = \{0,1\}$ and all sets we referred to in this paper are either languages over $\Sigma$ or sets consisting of *integers*. Call a set $L$ non-trivial if both $L$ and $\overline{L}$ have at least two elements. Let $\mathbb{N}$ denote the set of natural numbers and $\mathbb{N}^+$ denote $\mathbb{N}\backslash\{0\}$. For every $n \in \mathbb{N}$, we use $\Sigma^{=n}$ to represent strings over $\Sigma$ of length $n$. For every string/integer $x$, we use $|x|/abs(x)$ to denote the length/absolute value of $x$. We use a pairing function $\langle\cdot,\cdot\rangle$, where for every strings/integers $x$ and $y$, $\langle x,y\rangle = xy10^{|x|}$ and hence, $|\langle x,y\rangle| = 2|x| + |y| + 1$. For 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. All reductions used in this paper are *polynomial-time computable* unless otherwise specified. A language $L$ is *complete* for a complexity class $C$ for a reduction $r$ if every language in $C$ 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 provide the detailed definition for the most relevant reduction, many-one reduction, considered in this paper.

**Definition 2.1.** Define a language $A$ to be *polynomial-time many-one reducible* ($\leq_m^p$) to a language $B$, if there exists a polynomial-time computable function $f : \Sigma^* \to \Sigma^* \cup \{ACC, REJ\}$[1] that on every input string $x$ outputs $y \in B$ or $ACC$ if $x \in A$, and outputs $y \notin B$ or $REJ$ if $x \notin A$. Here $f$ is called a *many-one reduction* ($\leq_m^p$-reduction), or simply $m$-reduction from $A$ to $B$.

Now we define autoreducible and mitotic languages formally.

**Definition 2.2.** Given any type of reduction $r$, a language is *autoreducible for $r$* or *$r$-autoreducible*, if the language is reducible to itself via a reduction of type $r$ that does not query on the input. The $r$-reduction used here is called an *$r$-autoreduction*.

**Definition 2.3.** Given any type of reduction $r$, a language $L$ is *mitotic for $r$* or *$r$-mitotic*, if there exists another language $S$ where $S$ is polynomial-time decidable, and $L$, $S \cap L$ and $\overline{S} \cap L$ are all equivalent under the reduction $r$, i.e., $L \equiv_r S \cap L \equiv_r \overline{S} \cap L$.

Note that the many-one type of reduction we defined above is slightly weaker than the commonly used many-one reduction, denoted by $\leq_{mo}^p$, that always outputs a string $y \in \Sigma$. The two types of many-one reductions are equivalent for non-trivial sets in terms of reducibility. In particular, for any non-trivial sets $A$ and $B$, $A \leq_m^p B$ if and only if $A \leq_{mo}^p B$. Therefore, most results about the $\leq_{mo}^p$ reduction in complexity theory and from Glaßer et al. [8] hold for the $\leq_m^p$ reduction as well on non-trivial languages. The following are two examples.

**Corollary 2.1.** *A non-trivial language is $\leq_m^p$-complete for* NP *if and only if the language is* NP*-complete, i.e., $\leq_{mo}^p$-complete for* NP.

**Corollary 2.2.** *Every $\leq_m^p$-complete language for* NP *is $\leq_m^p$-mitotic.*

However, the traditional $\leq_{mo}^p$ reduction has the property of always outputting a string, which the reduction $\leq_m^p$ does not have. For this reason, our main results were only shown for the $\leq_m^p$ reduction. Adapting our proofs for the traditional $\leq_{mo}^p$ reduction would require an effective procedure that outputs a string accepted, and a string rejected by a given nondeterministic Turing machine. Our current paper does not include such as a procedure.

We will also need the following notions for the proofs.

[1] Here we assume that $ACC$ and $REJ$ represent two special symbols not appearing in $\Sigma$.

**Definition 2.4** *([8]).* For any pair of strings $x$ and $y$, define their *log-distance* as

$$logD(x,y) = sign(y-x)\lfloor \log_2 |y-x| \rfloor,$$

where $sign$ is the regular sign function and $|y-x|$ denotes the *lexicographical distance* between $x$ and $y$.

**Definition 2.5.** Let $N$ be a nondeterministic Turing machine in which each accepting path is of length exactly $p(n)$ for some polynomial $p$. A *left set* of $N$ is defined to be

$$left(N) = \{\langle x,y\rangle \mid x,y \in \Sigma^{=p(|x|)}, \exists z \\ (z \in \Sigma^{=p(|x|)},\ y \leq z, \text{and } N \text{ accepts } x \text{ along } z.)\}$$

The set $left(N)$ defined above is called a left set of $N$ since it contains all $\langle x,y\rangle$, where $x \in L(N)$, and $y$ is a computation of $N$ on $x$ that is to the left (smaller than or is equal to) of an accepting computation path. Left sets were introduced by Ogiwara and Watanabe [11] and originally defined on languages in NP. We found it more convenient to define them directly on nondeterministic polynomial-time Turing machines.

## 3. Results

Glaßer et al. [8] proved that every non-trivial autoreducible set $L$ is mitotic for the standard many-one type of reductions. The core part of the proof is to construct from a given m-autoreduction $f$ for $L$ a polynomial-time computable function $s : \Sigma^* \to \{0,1\}$ and another m-autoreduction $g$ for $L$, where $s(x) = 0 \Leftrightarrow s(g(x)) = 1$ for every string $x \in \Sigma^*$. Then the language $L$ can be *split* by $s$ into $L_0 = L \cap \{x \mid s(x) = 0\}$ and $L_1 = L \cap \{x \mid s(x) = 1\}$ while the function $g$ always *flips* the value $s(x)$ on every input $x \in \Sigma^*$, providing the needed mapping to make $L$ many-one reducible to both $L_0$ and $L_1$.

In this paper we will apply the above constructions of those "splitting" and "flipping" functions in a more structured way in order to split the given m-autoreducible set repeatedly and uniformly.

For convenience of references, we define splitting and flipping functions adjusted for our definition of many-one reductions below. We then describe how they can be constructed from given many-one autoreductions.

**Definition 3.1.** A function $s : \Sigma^* \to \{0,1,ACC,REJ\}$ is a *splitting* function for a language $L$ if for every $x \in \Sigma^*$,

- $x \in L \Longrightarrow s(x) \in \{0,1,ACC\}$, and
- $x \notin L \Longrightarrow s(x) \in \{0,1,REJ\}$.

A function $g : \Sigma^* \to \Sigma^* \cup \{ACC,REJ\}$ is a *flipping* function for a splitting function $s$ with respect to a language $L$ if $g$ is an m-autoreduction for $L$ and for every $x \in \Sigma^*$,

- $s(x) \in \{0,1\} \Longrightarrow s(g(x)) = 1 - s(x)$, and
- $s(x) \in \{ACC,REJ\} \Longrightarrow g(x) = s(x)$.

Splitting and flipping functions are closely related to m-mitoticity of languages as illustrated by the following proposition:

**Proposition 3.1.** *Assume that $s$ is a splitting function for $L$ and that $g$ is a flipping function for $s$ with respect to $L$. If both $s$ and $g$ are polynomial-time computable, then $L$ is m-mitotic.*

**Proof.** Define $L_0 = \{x \in L \mid s(x) \in \{0, ACC\}\}$ and $L_1 = L - L_0$. Define the following function:

$$f_0(x) = \begin{cases} REJ & \text{if } s(x) \in \{1, REJ\}, \\ ACC & \text{if } s(x) = ACC, \text{ and} \\ g(x) & \text{if } s(x) = 0. \end{cases}$$

Assume that $x \in L_0$. Then $s(x) \in \{0, ACC\}$. If $s(x) = ACC$, then $f_0(x) = ACC$. If $s(x) = 0$, then $f_0(x) = g(x) \in L_1$ since $s(g(x)) = 1 - s(x) = 1$. Assume that $x \notin L_0$. Then $s(x) \in \{1, REJ\}$. Hence, $f(x) = REJ$. Clearly $f_0$ is polynomial-time computable since $s$ and $g$ are polynomial-time computable. This shows that $f_0$ is an m-reduction from $L_0$ to $L_1$.

Similarly, we can show the following functions $f_1$ and $f$ are m-reductions from $L_1$ to $L$ and from $L$ to $L_0$, respectively. Therefore, $L_0$, $L_1$ and $L$ are m-equivalent. It follows that $L$ is m-mitotic.

$$f_1(x) = \begin{cases} REJ & \text{if } s(x) \in \{0, REJ, ACC\}, \text{ and} \\ x & \text{if } s(x) = 1. \end{cases}$$

$$f(x) = \begin{cases} s(x) & \text{if } s(x) \in \{ACC, REJ\}, \\ x & \text{if } s(x) = 0. \\ g(x) & \text{if } s(x) = 1. \end{cases} \quad \square$$

The following theorem is due to Glaßer et al. [7] though it is stated in a slightly more general way so that it can be applied to construct an m-autoreduction for either of the subsets resulting from splitting an NP-complete set. The key idea is to avoid querying on any string in a polynomial-time decidable set that $x$ belongs to instead of just avoiding querying on $x$.

**Theorem 3.2** *([7]). Let $N$ be a nondeterministic polynomial-time Turing machine, where each computation path is of length exactly $p(n)$ for some polynomial $p$. Then $left(N) \leq_m^p L(N)$ implies that $L(N)$ is m-autoreducible. In particular, the function defined in Algorithm 1 is an m-autoreduction for $L(N)$, where $h$ is an m-reduction from $left(N)$ to $L(N)$. Assume that $h$ runs in time $r(n)$ for some polynomial $r$. Then Algorithm 1 runs in $O(p(|x|)r(2|x| + p(|x|) + 1))$ time with output length at most $r(2|x| + p(|x|) + 1)$.*

**Input :** An arbitrary string $x \in \Sigma^*$, where $|x| = n$
**Output:** ACC, REJ, or a string $y \neq x$, where $y \in L(N) \iff x \in L(N)$

1 $m \longleftarrow p(n)$;
2 $y_0 \longleftarrow h(\langle x, 0^m \rangle)$;
3 **if** $y_0 \in \{ACC, REJ\}$ *or* $y_0 \neq x$ **then return** $y_0$ ;
4 **if** $N$ *accepts* $x$ *along* $1^m$ **then return** $ACC$ ;
5 $y_1 \longleftarrow h(\langle x, 1^m \rangle)$;
6 **if** $y_1 = x$ **then return** $REJ$;
7 // Here $y_0 = h(\langle x, 0^m \rangle) = x$.
8 // Here $y_1 = h(\langle x, 1^m \rangle) \in (\Sigma^* - \{x\}) \cup \{REJ\}$
9 Use a binary search to find a path $u$ of $N$ on $x$, where
10 $y_0 = h(\langle x, u \rangle) = x$, and
11 $y_1 = h(\langle x, u+1 \rangle) \in (\Sigma^* - \{x\}) \cup \{REJ\}$ ;
12 **if** *Any path $v$ is found, where $h(\langle x, v \rangle) = ACC$, when executing lines 9 - 11* **then return** $ACC$;
13 **if** $N$ *accepts* $x$ *along* $u$ **then return** $ACC$;
14 **else return** $y_1$ ;

**Algorithm 1:** The m-autoreduction for $L(N)$, given an m-reduction $h$ from $left(N)$ to $L(N)$.

**Proof.** We refer readers to Glaßer et al. [7] for the proof that Algorithm 1 is an m-autoreduction for $L(N)$. Here we just analyze the running time for Algorithm 1 and give a polynomial bound on the output lengths so that it can be used in the constructions later in this paper.

Since $N$ has no more than $2^{p(|x|)}$ computation paths, the binary search in lines 9 - 11 of Algorithm 1 requires at most $p(|x|)$ iterations. Each of those iterations requires computing $h(\langle x, u \rangle)$ for some strings $u$ of length $p(|x|)$. Since $h$ is computable in time $r(|x|)$, the binary search can be completed in time $O(p(|x|)r(|\langle x, y \rangle|)) = O(p(|x|)r(2|x| + p(|x|) + 1))$. The running time for the rest of Algorithm 1 is clearly dominated by the binary search.

The output length of Algorithm 1 is at most

$$|h(\langle x, u \rangle)| \leq r(|\langle x, y \rangle|) = r(2|x| + p(|x|) + 1). \quad \square$$

The following theorem summarizes the most relevant results from Glaßer et al. [8] that we need to establish the main results in this paper. The results were adjusted for our definition of many-one reductions. We also included specific bounds on the running times and output lengths for relevant functions, which required some more detailed analysis than the original proofs.

**Theorem 3.3.** *Let $f$ be an m-autoreduction for a language $L$. Assume that $f$ runs in time $p(n)$ for some increasing and positive polynomial $p$, where $p(n) \geq n$ for every $n \in \mathbb{N}$. Define a towering function $t(n)$ by $t(0) = 2$ and $t(i+1) = p(t(i))$. Define further that $t^{-1}(n) = \min\{i \in \mathbb{N} \mid t(i) \geq n\}$. Then the following hold:*

1. *The function $s$ defined in Algorithm 2 is a splitting function for $L$, computable in time $O(p(p(n)) \log n)$*
2. *The function $g$ defined in Algorithm 3 is a flipping function for the splitting function $s$ given in Statement 1 with respect to $L$, $g$ is computable in time $O(p^{(4)}(n))$, and $|g(x)| \leq p(p(|x|))$ for every $x \in \Sigma^*$.*

**Input :** An arbitrary string $x \in \Sigma^*$, where $|x| = n$
**Output:** a string $w \in \{0, 1, ACC, REJ\}$

1 $y \leftarrow f(x)$;
2 **if** $y \in \{ACC, REJ\}$ **then return** $y$;
3 $seg_x = t^{-1}(|x|)$;
4 $seg_y = t^{-1}(|y|)$;
5 **if** $seg_x < seg_y$ **then return** $seg_x \mod 2$;
6 $z \leftarrow f(y)$;
7 **if** $z \in \{ACC, REJ\}$ **then return** $z$;
8 **if** $seg_y < t^{-1}(|z|)$ **then return** $1 - (seg_y \mod 2)$;
9 **if** $x < y$ *and* $y > z$ **then return** 0;
10 **if** $x > y$ *and* $y < z$ **then return** 1;
11 $d_x = log D(x, y)$ (see definition in Section 2);
12 $d_y = log D(y, z)$;
13 **if** $d_x < d_y$ **then return** 0;
14 **if** $d_x > d_y$ **then return** 1;
15 **if** $\lfloor \frac{y}{2^{abs(d_x)+1}} \rfloor$ *is even* **then return** 0;
16 **else return** 1;

**Algorithm 2:** A splitting function $s$ for a language $L$ given an m-autoreduction $f$ for $L$.

**Input :** An arbitrary string $x \in \Sigma^*$, where $|x| = n$
**Output:** a string $w \in \{0, 1, ACC, REJ\}$

1 $z \leftarrow x$;
2 **while** $f(z) \in \Sigma^*$ *and* $|f(z)| > |z|$ *and* $s(x) = s(z)$ **do**
3 $\quad z \leftarrow f(z)$
4 **end**
5 **if** $f(z) \notin \Sigma^*$ **then return** $f(z)$;
6 **else return** $z$;

**Algorithm 3:** A flipping function $g$ for a splitting function $s$ with respect to a language $L$, for which $f$ is an m-autoreduction.

**Proof.** Much of the proof was given in Glaßer et al. [8]. We provide the proof adjusted for our definition of many-one reductions for completeness. We also present slightly different algorithms from those used in the original paper for computing the targeted splitting and flipping functions. Our algorithms are special cases of a general algorithm we

[12] used earlier as an attempt to generalize the results by Glaßer et al. [8] and study the relations between mitoticity and autoreducibility for reductions weaker than the many-one and 1-tt reductions.

We first note that $t^{-1}(n)$ can be computed in polynomial time. This can be achieved by performing a binary search for $i$ between 1 and $n$, where $t(i) \geq n$ and $t(i-1) < n$. This gives an $O(\log^2 n)$-time algorithm, assuming that $n$ is represented in binary form. We also have the following observation about the function $t$.

**Proposition 3.4.** *For every* $x, y \in \Sigma^*$, $|y| \leq p(|x|) \Longrightarrow t^{-1}(|y|) \leq t^{-1}(|x|)+1$.

**Proof.** Let $t^{-1}(|x|) = k$. Then by the definition of $t^{-1}$, $t(k) \geq |x| > t(k-1)$. This yields $|y| \leq p(|x|) \leq p(t(k)) = t(k+1)$. Hence, $t^{-1}(|y|) \leq k+1 = t^{-1}(|x|)+1$. □

Now consider the functions $s$ and $g$ defined in Algorithm 2 and Algorithm 3, respectively. Clearly $s$ is a splitting function for $L$ since the only case $s$ outputs $ACC$ or $REJ$ is when $f(x) \in \{ACC, REJ\}$ or $f(f(x)) \in \{ACC, REJ\}$. In both cases, $s$ makes the output correctly due to that $f$ is an m-autoreduction for $L$. The fact that $g(x)$ is a flipping function for $s$ with respect to $L$ follows easily from its definition considering again that $f$ is an m-autoreduction.

We now analyze the running times of functions $s$ and $g$. We already know that $t^{-1}(n)$ can be computed in $O(\log^2 n)$ time. We further observe the following about the running time of function $s$, where $n = |x|$:

- $|y| \leq p(n)$ and $|z| \leq p(p(n))$ since $y = f(x)$ and $z = f(y)$.
- $log\,D(x, y)$ and $log\,D(y, z)$ can be computed in $O(p(p(n)) \log n)$ time since the function $log\,D$ can be computed in $O(m \log m)$ time on inputs of length $m$.
- The expression in Line 15 can be computed in $O(|y| + |d_x|) = O(p(n))$ time.

Therefore, the total running time of the function $s$ is bounded by $O(p(p(n)) \log n)$.

The following lemma is critical in showing that $g$ runs in polynomial time.

**Lemma 3.5.** *For every* $x \in \Sigma^*$, *there exists* $i \in [0, 4p(|x|)+2]$, *where* $|f^{(j)}(x)| \leq p(p(|x|))$ *for every* $j \in [0, i]$ *and one of the following holds:*

1. $f^{(i)}(x) \in \{ACC, REJ\}$ *and*
$$s(f^{(i-1)}(x)) = s(f^{(i-2)}(x)) = \cdots = s(f(x)) = s(x) \notin \{ACC, REJ\}.$$
2. $f^{(i)}(x) \notin \{ACC, REJ\}$ *and*
$$s(f^{(i)}(x)) \neq s(f^{(i-1)}(x)) = s(f^{(i-2)}(x)) = \cdots = s(f(x)) = s(x).$$

**Proof.** Let $x$ be a string in $\Sigma^*$ and $|x| = n$. Let $m = 4p(n)+2$. Assume that there exists $i \in [0, m]$, where $f^{(i)}(x) \in \{ACC, REJ\}$ and $f^{(j)}(x) \notin \{ACC, REJ\}$ for every $j \in [0, i-1]$. If for every $j \in [0, i-1]$, $s(f^{(j)}(x)) = s(x)$. Then Statement 1 in Lemma 3.5 holds. Otherwise, an $i' \in [0, i-1]$ must exist, where

$$s(f^{(i')}(x)) \neq s(f^{(i'-1)}(x)) = s(f^{(i'-2)}(x)) = \cdots = s(f(x)) = s(x).$$

This means that Statement 2 in Lemma 3.5 holds.

Now assume that $f^{(i)}(x) \notin \{ACC, REJ\}$ for every $i \in [0, m]$. We show that there must exist $i \in [0, m]$, where Statement 2 in Lemma 3.5 holds. Let $x_j = f^{(j)}(x)$ for every $j \in [0, m]$. It suffices to show that there exist $i_1, i_2 \in [0, m]$, where $s(x_{i_1}) \neq s(x_{i_2})$.

We consider the following cases in that order so that the proof for Case $e$ assumes that none of the cases 1, 2, ⋯, $e-1$ holds.

**Case 1**: There exists $i_1 \in [1, m-1]$, where $t^{-1}(|x_{i_1}|) < t^{-1}(|x_{i_1+1}|)$. Let $i_1$ be the smallest such number. Then $s$ on $x_{i_1}$ outputs $t^{-1}(|x_{i_1}|) \mod 2$ in line 5. On $x_{i_1-1}$, $s$ either outputs $t^{-1}(|x_{i_1-1}|) \mod 2$ in line 5 if $t^{-1}(|x_{i_1-1}|) < t^{-1}(|x_{i_1}|)$, or outputs $1 - (t^{-1}(|x_{i_1}|) \mod 2)$ in line 8 if $t^{-1}(|x_{i_1-1}|) \geq t^{-1}(|x_{i_1}|)$. By Proposition 3.4, $t^{-1}(|x_{i_1-1}|) = t^{-1}(|x_{i_1}|) - 1$ if $t^{-1}(|x_{i_1-1}|) < t^{-1}(|x_{i_1}|)$. Hence, in both cases, $s(x_{i_1-1}) \neq s(x_{i_1})$.

For the remaining cases we assume now that $t^{-1}(|x_j|) \geq t^{-1}(|x_{j+1}|)$ for every $j \in [1, m-1]$. This implies that

$$\forall j \in [1, m]\ |x_j| \leq p(|x_1|) \leq p(p(|x|)) \tag{1}$$

**Case 2**: There exist $i_1, i_2 \in [1, m-1]$, where $x_{i_1-1} < x_{i_1}$ and $x_{i_1} > x_{i_1+1}$, and $x_{i_2-1} > x_{i_2}$ and $x_{i_2} < x_{i_2+1}$. In this case $s(x_{i_1}) = 0$ and $s(x_{i_2}) = 1$. Hence, $s(x_{i_1}) \neq s(x_{i_2})$.

Note that if Case 2 does not hold, then the sequence $\{x_j\}_{1 \leq j \leq m}$ is either monotonic or made up of one increasing and one deceasing subsequences. In both cases, there exists a monotonic subsequence of $\{x_j\}_{1 \leq j \leq m}$ of length at least $m/2$. We assume for the remaining cases that for some $s_3, t_3 \in [1, m]$, $t_3 - s_3 + 1 \geq m/2$ and $\{x_j\}_{s_3 \leq j \leq t_3}$ is a monotonic subsequence of $\{x_j\}_{1 \leq j \leq m}$.

**Case 3**: There exist $i_1, i_2 \in [s_3, t_3 - 2]$, where $d_{x_{i_1}} < d_{x_{i_1+1}}$ and $d_{x_{i_2}} > d_{x_{i_2+1}}$. In this case, $s$ on input $x_{i_1}$ outputs 0 in line 13, and on input $x_{i_2}$ outputs 1 in line 14.

If none of the cases 1-3 hold, then $\{d_{x_j}\}_{s_3 \leq j \leq t_3-1}$ must be a monotonic subsequence or made up of one increasing and one deceasing subsequences. By an argument similar to that after Case 2 we can assume for the remaining cases that for some $s_4, t_4 \in [s_3, t_3 - 1]$, $t_4 - s_4 + 1 \geq (t_3 - 1 - s_3 + 1)/2 = (t_3 - s_3)/2 \geq m/4 - 1/2$ and $\{d_{x_j}\}_{s_4 \leq j \leq t_4}$ is a monotonic subsequence of $\{d_{x_j}\}_{s_3 \leq j \leq t_3-1}$.

**Case 4**: There exists $i_1 \in [s_4, t_4 - 1]$, where $d_{x_{i_1}} = d_{x_{i_1+1}}$. Then by Lemma 3.6 below, $\{s(x_{i_1}), s(x_{i_1+1}), s(x_{i_1+2})\} = \{0, 1\}$.

**Lemma 3.6** *([8]). Let* $x$, $y$, *and* $z$ *be strings over* $\Sigma$, *where* $x < y < z$ *or* $x > y > z$, *and* $log\,D(x, y) = log\,D(y, z) = d$ *for some* $d \in \mathbb{Z}$. *Then the set* $\{\lfloor \frac{w}{2^{abs(d)+1}} \rfloor \mid w \in \{x, y, z\}\}$ *contains at least one even number and one odd number.*

Assume now that none of the cases 1-4 holds. Then $\{d_{x_j}\}_{s_4 \leq j \leq t_4}$ is an increasing or decreasing sequence due to cases 3 and 4. Note that $\{x_j\}_{s_4 \leq j \leq t_4+1}$ is a subsequence of the monotonic sequence $\{x_j\}_{s_3 \leq j \leq t_3}$. Hence, $\{d_{x_j}\}_{s_4 \leq j \leq t4}$ is a monotonic sequence consisting of integers of the same sign. Therefore,

$$\begin{aligned}
|x_{t_4+1} - x_{s_4}| &= |x_{t_4+1} - x_{t_4}| + |x_{t_4} - x_{t_4-1}| + \cdots + |x_{s_4+1} - x_{s_4}| \\
&\geq 2^{abs(d_{t_4+1})} + 2^{abs(d_{t_4})} + \cdots + 2^{abs(d_{s_4})} \\
&\geq 2^{t_4-s_4+1} + 2^{t_4-s_4} + \cdots + 2^1 + 2^0 \\
&\geq 2^{t_4-s_4+2} - 1 \\
&\geq 2^{m/4-1/2+1} \\
&\geq 2^{p(n)+1}
\end{aligned}$$

This implies that either $x_{s_4}$ or $x_{t_4}$ must be of length greater than $p(n)$ as the maximal lexicographical distance between two strings of length $p(n)$ or less is $2^{p(n)} - 1$. That is a contradiction to Statement (1). Therefore, we have argued in all cases that there exists $i \in [0, m]$, where either Statement 1 or Statement 2 is true.

Finally, let $i$ be the smallest index satisfying conditions for Case 1. Then for every $j \in [1, i-1]$, $t^{-1}(|x_j|) \geq t^{-1}(|x_{j+1}|)$ and hence, for every $j \in [2, i]$, $t^{-1}(|x_j|) \leq t^{-1}(|x_1|)$. This implies that $|x_j| \leq p(|x_1|) \leq p(p(|x|))$ for every $j \in [1, i]$ since $|x_1| \leq p(|x|)$. The same holds if Case 1 does not hold due to Inequality (1). This shows that $\forall j \in [0, i]\ |x_j| \leq p(p(|x|))$ and finishes the proof of Lemma 3.5. □

By Lemma 3.5, the while loop in Algorithm 3 runs for no more than $O(p(n))$ iterations on an input of size $n$. Each iteration requires $O(p(|z|))$

time to compute $f(z)$ for some $z$ of length at most $p(p(n)$. Hence, the running time of $g$ is bounded by $O(p^{(4)}(n))$.

For every input $x \in \Sigma^*$, let $i$ be the minimal number guaranteed to exist by Lemma 3.5. Then $g(x) = f^{(i)}(x)$ and $|g(x)| \leq p(p(|x|))$ by Lemma 3.5.

This finishes the proof of Theorem 3.3. □

The following theorem describes the critical step that splits an NP-complete set in a uniform way into two subsets, both being equivalent to the original set for the many-one reduction.

**Theorem 3.7.** *Let $N$ be a nondeterministic polynomial-time Turing machine, where $left(N) \leq_m^p L(N)$ via a function $h$. Assume that $s$ is a splitting function for $L(N)$ and $g$ is a flipping function for $s$ with respect to $L(N)$. Then the following hold for the nondeterministic polynomial-time Turing machines $N_0$ and $N_1$, defined in Algorithm 4, and the functions $h_0$ and $h_1$, defined in Algorithm 5:*

- $L(N_0) = \{x \in L(N) \mid s(x) \in \{0, ACC\}\}$,
- $L(N_1) = \{x \in L(N) \mid s(x) = 1\}$, *and*
- $left(N_0) \leq_m^p L(N_0)$ *via* $h_0$, *and* $left(N_1) \leq_m^p L(N_1)$ *via* $h_1$.
- $L(N) \equiv_m^p L(N_0) \equiv_m^p L(N_1)$

*In addition, assume that $N$ runs in time $p(n)$ and functions $h$, $s$, and $g$ are computable in time $q_1(n), q_2(n)$, and $q_3(n)$, in that order, where $p$, $q_1$, $q_2$, and $q_3$ are all polynomials. Then both $N_0$ and $N_1$ run in $O(p(n) + q_2(n))$ time and both $h_0$ and $h_1$ are computable in $O(q_1(n) + q_2(n) + q_3(n))$ time.*

**Input** : An arbitrary string $x \in \Sigma^*$, where $|x| = n$
**Output:** $ACC$ or $REJ$

1 **if** $N$ *rejects* $x$ **then return** $REJ$ ;
2 **if** $s(x) \in \{REJ, 1-b\}$ **then return** $REJ$ ;
3 // The line below should be included for $b = 1$ only
4 // **if** $s(x) = ACC$ **then return** $REJ$
5 **return** $ACC$

**Algorithm 4:** A nondeterministic polynomial-time TM $N_b$, where $b \in \{0,1\}$, $L(N_0) = \{x \in L(N) \mid s(x) \in \{0, ACC\}\}$, and $L(N_1) = \{x \in L(N) \mid s(x) = 1\}$, given a nondeterministic polynomial-time TM $N$ and a splitting function $s$ for $L(N)$.

**Input** : $\langle x, y\rangle$, where $x, y \in \Sigma^*$ and $|x| = n$
**Output:** $ACC$, $REJ$, or a string $z$, where
$\langle x, y\rangle \in left(N_b) \iff z \in L(N_b) \cup \{ACC\}$

1 **if** $s(x) \in \{REJ, 1-b\}$ **then return** $REJ$ ;
2 $z \longleftarrow h(\langle x, y\rangle)$;
3 **if** $z \in \{ACC, REJ\}$ **then return** $z$;
4 **if** $s(z) \in \{ACC, REJ, s(x)\}$ **then return** $z$;
5 **else return** $g(z)$;

**Algorithm 5:** An m-reduction $h_b$ from $left(N_b)$ to $L(N_b)$, where $b \in \{0,1\}$, $L(N_0) = \{x \in L(N) \mid s(x) \in \{0, ACC\}\}$, and $L(N_1) = \{x \in L(N) \mid s(x) = 1\}$, given a nondeterministic polynomial-time TM $N$, a splitting function $s$ for $L(N)$, a flipping function $g$ for $s$ with respect to $L(N)$, and an m-reduction $h$ from $left(N)$ to $L(N)$.

**Proof.** Let $N$, $h$, $s$ and $g$ be given as in the premise of the theorem. Given the definitions of nondeterministic Turing machine $N_b$ and functions $h_b$, where for $b \in \{0,1\}$, as given by Algorithm 4 and Algorithm 5, respectively, the statements about their running times in the theorem clearly hold.

It is also clear that $L(N_0) = \{x \in L(N) \mid s(x) \in \{0, ACC\}\}$ and $L(N_1) = \{x \in L(N) \mid s(x) = 1\}$, and that $L(N) \equiv_m^p L(N_0) \equiv_m^p L(N_1)$ due to existence of the flipping function $g$ for the splitting function $s$ with respect to $L$. We now show that $left(N_b) \leq_m^p L(N_b)$ for $b \in \{0,1\}$.

Note that $N_0$ and $N_1$ both have the same nondeterministic choices as $N$ and for every $x, y \in \Sigma^*$,

- $y$ is an accepting path of $N_0$ if and only if $y$ is an accepting path of $N$ on $x$ and $s(x) \in \{0, ACC\}$, and
- $y$ is an accepting path of $N_1$ if and only if $y$ is an accepting path of $N$ on $x$ and $s(x) = 1$.

It follows that

$$left(N_0) = \{\langle x, y\rangle \in left(N) \mid s(x) \in \{0, ACC\}\} \tag{2}$$

$$left(N_1) = \{\langle x, y\rangle \in left(N) \mid s(x) = 1\} \tag{3}$$

By hypothesis, $h$ is an m-reduction from $left(N)$ to $L(N)$. Consider the function $h_0$ as defined in Algorithm 5.

Assume that $\langle x, y\rangle \in left(N_0)$. Then $\langle x, y\rangle \in left(N)$ and $s(x) \in \{0, ACC\}$ by Equation (2). Assume $s(x) = 0$. Then $h_0$ reaches Line 2. Note that in this case $z = h(\langle x, y\rangle \in L(N) \cup \{ACC\}$ since $h$ is an m-reduction from $left(N)$ to $L(N)$. Hence, $h_0$ returns $ACC$ in Line 3 if $z = ACC$. Now assume $z \in L(N)$. Then $h_0$ returns $z \in L(N_0)$ if $s(z) \in \{ACC, 0\}$ in Line 4 or $g(z) \in L(N_0)$ if $s(z) = 1$. The latter is due to that $g$ is a flipping function for $s$ with respect to $L(N)$. Therefore, in all cases if $\langle x, y\rangle \in left(N_0)$, then $h_0(\langle x, y\rangle) \in L(N_0) \cup \{ACC\}$.

Now Assume that $\langle x, y\rangle \notin left(N_0)$. Then $\langle x, y\rangle \notin left(N)$ or $s(x) \in \{1, REJ\}$ by Equation (2). If $s(x) \in \{1, REJ\}$, then $h_0$ on input $\langle x, y\rangle$ returns $REJ$ in Line 1. Now assume $s(x) \in \{0, ACC\}$. Then $h_0$ reaches Line 2. Note that in this case $z = h(\langle x, y\rangle \in \overline{L(N)} \cup \{REJ\}$ since $h$ is an m-reduction from $left(N)$ to $\overline{L(N)}$. Hence, $h_0$ returns $REJ$ in Line 3 if $z = REJ$. Now assume $z \in \overline{L(N)}$. Then $h_0$ returns $z \notin L(N_0)$ if $s(z) \in \{REJ, 0\}$ in Line 4 or $g(z) \notin L(N_0)$ if $s(z) = 1$. The latter is due to that $g$ is a flipping function for $s$ with respect to $L_N$. Therefore, in all cases if $\langle x, y\rangle \notin left(N_0)$, then $h_0(\langle x, y\rangle) \notin L(N_0) \cup \{ACC\}$.

This shows that $h_0$ is an m-reduction from $left(N_0)$ to $L(N_0)$. A similar argument would show that $h_1$, also as defined in Algorithm 5, is an m-reduction from $left(N_1)$ to $L(N_1)$.

This finishes the proof of Theorem 3.7. □

Now we have all the necessary tools to establish our main result:

**Theorem 3.8.** *Let $N$ be a nondeterministic polynomial-time Turing machine. Assume that $left(N) \leq_m^p L(N)$ via some polynomial-time computable function $h$. Then there exist a set of nondeterministic polynomial-time Turing machines $\{N_w\}_{w\in\Sigma^*}$ and sets of polynomial-time computable functions $\{h_w\}_{w\in\Sigma^*}$, $\{f_w\}_{w\in\Sigma^*}$, $\{s_w\}_{w\in\Sigma^*}$, and $\{g_w\}_{w\in\Sigma^*}$, where $N_\lambda = N$ and all the following hold for every $w \in \Sigma^*$:*

1. $L(N_w) \equiv_m^p L(N)$, *where*
   a. $L(N_w) = L(N)$ *if* $w = \lambda$,
   b. $L(N_w) = \{x \in L(N_{w'}) \mid s_{w'}(x) \in \{0, ACC\}\}$, *if* $w = w'0$ *for some* $w' \in \Sigma^*$, *and*
   c. $L(N_w) = \{x \in L(N_{w'}) \mid s_{w'}(x) = 1\}$, *if* $w = w'1$ *for some* $w' \in \Sigma^*$.
2. $h_w$ *is an m-reduction from* $left(N_w)$ *to* $L(N_w)$.
3. $f_w$ *is an m-autoreduction for* $L(N_w)$.
4. $s_w$ *is a splitting function for* $L(N_w)$.
5. $g_w$ *is a flipping function for* $s_w$ *with respect to* $L(N_w)$.
6. *The encodings of* $N_w$ *and all the functions of* $h_w$, $f_w$, $s_w$, *and* $g_w$ *can be computed from* $w$.

**Proof.** We prove the theorem by induction on $w$. Let $N$ be a nondeterministic polynomial-time Turing machine NP-complete language, where there is an m-reduction $h$ from $left(N)$ to $L(N)$.

**Base Case**: $w = \lambda$. We let $N_\lambda$ be an nondeterministic polynomial-time Turing machine where $L(N_\lambda) = L(N)$ and every computation path

of $N$ on input $x$ is of length exactly $p(|x|)$. It holds trivially that $L(N_\lambda) \equiv_m^p L(N)$. It should also be clear that $h$ can be modified in a straightforward way to a function $h_\lambda$ so that $left(N_\lambda) \leq_m^p L(N_\lambda)$ via $h_\lambda$. Hence, Statement 1a and 2 hold. Statement 1b or 1c does not apply. Statement 3 holds by applying Theorem 3.2 with $h = h_\lambda$ to obtain the m-autoreduction $f_\lambda$ in the way as described in Algorithm 1. Statements 4 and 5 hold by Theorem 3.3, where function $s_\lambda$ is defined using $f_\lambda$, and the function $g_\lambda$ is defined using $f_\lambda$ and $s_\lambda$ in the ways as described in Algorithm 2 and 3, respectively.

The encoding of $N_\lambda$ can be obviously computed from $\lambda$ given the polynomial $p(|x|)$ fixed previously for $N_\lambda$. Then a program can use the encodings of $N_\lambda$ and the m-reduction $h_\lambda$ from $left(N_\lambda)$ to $L(N_\lambda)$ to generate the encoding of function $f_\lambda$ according to Algorithm 1. Subsequently, the encoding of $f_\lambda$ can be used to generate the encodings of functions $s_\lambda$ and $g_\lambda$ according to Algorithms 2 and 3, respectively.

Here we need to show how we find a polynomial $q$ that bounds the output lengths of $f_\lambda$ in order to generate the part of the encoding of $s_\lambda$ that computes $t^{-1}$. Assume that function $h_\lambda$ is computable in time $r(n)$ for some polynomial $r$. Then output lengths of $h_\lambda$ are bounded by $r(n)$ on inputs of length $n$ and that of $f_\lambda$ is bounded by $q(n) = r(2n + p(n) + 1)$ according to Theorem 3.2. Then we can define and compute the tower function $t$ used by Algorithm 2 in terms of the polynomial $q$. Hence, the part of encoding of $s_\lambda$ that computes $t^{-1}$ can be generated using $q$. Therefore, the encodings of $N_\lambda$, $h_\lambda$, $f_\lambda$, $s_\lambda$ and $g_\lambda$ can all be computed from $w = \lambda$. This shows Statement 6 holds for the base case.

**Induction Step**: Assume that $N_w$, $h_w$, $f_w$, $s_w$ and $g_w$ have been properly defined that satisfy statements 1 - 6. Then we apply Theorem 3.7 with $N = N_w$, $h = h_w$, and $s = s_w$ to define nondeterministic Turing machines $N_{w0}$ and $N_{w1}$, and $m$-reductions $h_{w0}$ and $h_{w1}$ that satisfy statements 1 and 2, respectively. Then similar to the base case we

- apply Theorem 3.2 with $N = N_{w0}$ and $N = N_{w1}$ to define $f_{w0}$ and $f_{w1}$, respectively, that satisfy Statement 3, and
- apply Theorem 3.3 with $f_{w0}$ and $f_{w1}$ to define $s_{w0}$ and $s_{w1}$, respectively, that satisfy Statement 4, and $g_{w0}$ and $g_{w1}$, respectively, that satisfy Statement 5.

In addition it is clear that all those functions defined above, $f_{wb}$, $s_{wb}$, and $g_{wb}$, where $b \in \{0,1\}$, are polynomial-time computable given that $N_w$ is a polynomial-time nondeterministic machine, and $h_w$ and $s_w$ are polynomial-time computable functions by Theorems 3.7, 3.2, and 3.3. Therefore, statements 1 - 5 hold for $w0$ and $w1$.

To compute the encoding of $N_{wb}$, $h_{wb}$, $f_{wb}$, $s_{wb}$, and $g_{wb}$ for $b \in \{0,1\}$, we just need to follow Algorithms 1 - 5 with $N = N_w$, $h = h_w$, $f = f_w$, $s = s_w$, and $g = g_w$.

Here we need to show how to compute the part of the encoding of $s_{w0}$ that computes the towering function $t = t_{w0}$ used by $s_{w0}$. We observe that we just need to find a polynomial that bounds the output lengths of $f_{w0}$. Let $q_w$ and $r_w$ be two polynomials that bound the output lengths of $f_w$ and $h_w$, respectively. Note that $q_\lambda = q$ and $r_\lambda = r$ as given in the base case. Note also that the length of every computation path of $N_w$ is the same as that of $N_\lambda$, i.e., $p(|x|)$ on input $x$. We then observe for every $x \in \Sigma^*$ that there exists some $y \in \Sigma^*$, where $|y| = p(|x|)$ and

$$
\begin{aligned}
|f_{w0}(x)| &\leq |h_{w0}(\langle x, y\rangle)| \text{ by Algorithm 1} \\
&\leq max(|h_w(\langle x, y\rangle)|, |g_w(h_w(\langle x, y\rangle))|) \text{ by Algorithm 5} \\
&\leq q_w(q_w(r_w(|\langle x, y\rangle|))) \text{ by Theorem 3.3} \\
&\leq q_w(q_w(r_w(2|x| + p(|x|) + 1)))
\end{aligned}
$$

Therefore, if we use two polynomials $q_w(n)$ and $r_w(n)$ to bound the output lengths of $f_w$ and $h_w$ in order to compute $s_w$, then we can use $q_{w0} = q_w(q_w(r_w(2n + p(n) + 1)))$ and $r_{w0} = q_w(q_w(r_w(n)))$ to bound the output lengths of $f_{w0}$ and $h_{w0}$, respectively, in order to compute $s_{w0}$. It follows that the encodings of the functions $s_{w0}$ and $g_{w0}$, and the Turing machine $N_{w0}$ can all be computed from $w0$. The same argument shows the same statement for $s_{w1}$, $g_{w1}$, and $N_{w1}$ holds. This implies that Statement 6 holds for $w0$ and $w1$, and completes (the induction step of) the proof of Theorem 3.8. □

With Theorem 3.8 now we see that an NP-complete language can be partitioned by a computable function into an infinite set of NP-complete languages. Similar results were shown for NEXP by Buhrman, Hoene and Torenvliet [2].

**Definition 3.2.** Define a class $C$ of languages to be *effectively presentable* if there exists a computable function $f : \mathbb{N} \to \Sigma^*$ such that

- $M$ is the encoding of a Turing machine for every $M \in range(f)$, and
- $C = \{L(M) \mid M \in range(f)\}$.

**Corollary 3.9.** *Let $L$ be an* NP*-complete set. Then there exists an effectively presentable set of* NP*-complete languages $\{R_k\}_{k\in\mathbb{N}}$, where*

- $\bigcup_{k\in\mathbb{N}} R_k = L$, *and*
- $\forall i \neq j \in \mathbb{N}\ R_i \cap R_j = \emptyset$.

**Proof.** Given an NP-complete language $L$, let $N$ and $h$ be the nondeterministic Turing machine and m-reduction as defined in the premise of Theorem 3.8, respectively. Let $\{N_w\}_{w\in\Sigma^*}$ be the set of nondeterministic polynomial-time Turing machines obtained by applying Theorem 3.8 on $L$, $N$, and $h$. Therefore, $\{N_w\}_{w\in\Sigma^*}$ is an effectively presentable set. For every $k \in \mathbb{N}$ define $R_k = L(N_{1^k0})$. Clearly $\{R_k\}_{k\in\mathbb{N}}$ is effectively presentable. It is also straightforward to verify that all $R_k$'s are NP-complete and satisfy both statements in this corollary. □

Note that with the additional assumption that NP ≠ P, the language $L$ and all the resulting languages $R_k$ in Corollary 3.9 are infinite.

## CRediT authorship contribution statement

**Liyu Zhang:** Conceptualization, Investigation, Methodology, Supervision, Validation, Writing – original draft, Writing – review & editing. **Mahmoud Quweider:** Conceptualization, Investigation, Writing – original draft, Writing – review & editing. **Fitra Khan:** Conceptualization, Investigation, Writing – original draft, Writing – review & editing. **Hansheng Lei:** Conceptualization, Investigation, Writing – original draft, Writing – review & editing.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

No data was used for the research described in the article.

## References

[1] H. Buhrman, L. Fortnow, D. van Melkebeek, L. Torenvliet, Using autoreducibility to separate complexity classes, SIAM J. Comput. 29 (5) (2000) 1497–1520.
[2] H. Buhrman, A. Hoene, L. Torenvliet, Splittings, robustness and structure of complete sets, SIAM J. Comput. 27 (3) (1998) 637–653.
[3] H. Buhrman, L. Torenvliet, A Post's program for complexity theory, Bull. EATCS 85 (2005) 41–51.
[4] C. Glaßer, A. Selman, S. Travers, L. Zhang, Non-mitotic sets, Theor. Comput. Sci. 410 (21–23) (2009) 2011–2033.

[5] C. Glaßer, D. Nguyen, A. Selman, M. Witeck, Introduction to autoreducibility and mitoticity, in: A. Day, et al. (Eds.), Computability and Complexity, Essays Dedicated to Rodney G. Downey on the Occasion of His 60th Birthday, in: Lecture Notes in Computer Science, vol. 10010, Springer, 2017, pp. 56–78.
[6] C. Glaßer, M. Ogihara, A. Pavan, A. Selman, L. Zhang, Autoreducibility and mitoticity, ACM SIGACT News 40 (3) (2009) 60–76.
[7] C. Glaßer, M. Ogihara, A. Pavan, A.L. Selman, L. Zhang, Autoreducibility, mitoticity, and immunity, J. Comput. Syst. Sci. 73 (2007) 735–754.
[8] C. Glaßer, A. Pavan, A. Selman, L. Zhang, Splitting NP-complete sets, SIAM J. Comput. 37 (5) (2008) 1517–1535.
[9] L. Hemaspaandra, M. Ogihara, The Complexity Theory Companion, Springer, 2002.
[10] S. Homer, A. Selman, Computability and Complexity Theory. Texts in Computer Science, 2nd edition, Springer, New York, December 2011.
[11] M. Ogiwara, O. Watanabe, On polynomial-time bounded truth-table reducibility of NP sets to sparse sets, SIAM J. Comput. 20 (3) (1991) 471–483.
[12] L. Zhang, M. Quweider, H. Lei, F. Khan, Weak mitoticity of bounded disjunctive and conjunctive truth-table autoreducible sets, Theor. Comput. Sci. 822 (2020) 36–48.