Autoreducibility, Mitoticity, and Immunity

Christian Glaßer[1], Mitsunori Ogihara[2*], A. Pavan[3**], Alan L. Selman[4***], and Liyu Zhang[4]

[1] Universität Würzburg, glasser@informatik.uni-wuerzburg.de
[2] University of Rochester, ogihara@cs.rochester.edu
[3] Iowa State University, pavan@cs.iastate.edu
[4] University at Buffalo, {selman,lzhang7}@cse.buffalo.edu

Abstract. We show the following results regarding complete sets.

– NP-complete sets and PSPACE-complete sets are many-one autoreducible.
– Complete sets of any level of PH, MODPH, or the Boolean hierarchy over NP are many-one autoreducible.
– EXP-complete sets are many-one mitotic.
– NEXP-complete sets are weakly many-one mitotic.
– PSPACE-complete sets are weakly Turing-mitotic.
– If one-way permutations and quick pseudo-random generators exist, then NP-complete languages are m-mitotic.
– If there is a tally language in NP $\cap$ coNP $-$ P, then, for every $\epsilon > 0$, NP-complete sets are not $2^{n^{(1+\epsilon)}}$-immune.

These results solve several of the open questions raised by Buhrman and Torenvliet in their 1994 survey paper on the structure of complete sets.

1 Introduction

We solve several open questions identified by Buhrman and Torenvliet in their 1994 survey paper on the structure of complete sets [12]. It is important to study the computational structure of complete sets, because they, by reductions of all the sets in the class to the complete sets, represent all of the structure that a class might have. For this reason, complete sets might have simpler computational structure than some other sets in the class. Here we focus attention primarily on autoreducibility, mitoticity, and immunity.

Trakhtenbrot [28] introduced the notion of autoreducibility in a recursion theoretic setting. A set A is *autoreducible* if there is an oracle Turing machine M such that $A = L(M^A)$ and M on input x never queries x. Ladner [23] showed that there exist Turing-complete recursively enumerable sets that are not autoreducible. Ambos-Spies [2] introduced the polynomial-time variant of autoreducibility, where we require the oracle Turing machine to run in polynomial time. Yao [32] introduced the notion of coherence, which coincides with probabilistic polynomial-time autoreducibility. In this paper, we assume that all reductions are polynomial-time computable. In particular, we write "autoreducible" to mean "polynomial-time autoreducible."

The question of whether complete sets for various classes are autoreducible has been studied extensively [32, 7, 10], and is currently an area of active research [15]. Beigel and Feigenbaum [7] showed that Turing complete sets for the classes that form the polynomial-time hierarchy, $\Sigma_i^{\mathrm{P}}, \Pi_i^{\mathrm{P}}$, and Δ_i^{P}, are Turing autoreducible. Thus, all Turing complete sets for NP are Turing autoreducible. Buhrman et al. [10] showed that Turing complete sets for EXP and Δ_i^{EXP} are autoreducible, whereas there exists a Turing complete set for EESPACE that is not Turing

* Research supported in part by NSF grants EIA-0080124, EIA-0205061, and NIH grant P30-AG18254.
** Research supported in part by NSF grants CCR-0344817 and CCF-0430807.
*** Research supported in part by NSF grant CCR-0307077.

auto-reducible. They showed that answering questions about autoreducibility of intermediate classes results in interesting separation results. Regarding NP, Buhrman et al. [10] showed that all truth-table complete sets for NP are probabilistic truth-table autoreducible. Thus, all NP-complete sets are probabilistic truth-table autoreducible.

Buhrman and Torenvliet [12] asked whether all NP-complete sets are many-one autoreducible and whether all PSPACE-complete sets are many-one autoreducible. We resolve these questions positively: all NP-complete sets and PSPACE-complete sets are (unconditionally) many-one autoreducible. We generalize the two results to show that for each class in MODPH [19] (the hierarchy constructed by applying to P a constant number of operators chosen from $\{\exists\cdot, \forall\cdot\} \cup \{\mathrm{MOD}_k\cdot, \mathrm{coMOD}_k\cdot \mid k \geq 2\}$), all of its nontrivial m-complete sets are m-autoreducible. We obtain as a corollary that no m-complete sets for NP, PSPACE, or for the classes of MODPH and $2n$-generic.

Autoreducible sets can be thought of as sets having some redundant information. For example, if A is m-autoreducible by the reduction f, then x and $f(x)$ both contain the same information concerning whether x belongs to A. How much redundancy is there in complete sets? Informally, an infinite set is *mitotic* if it can be partitioned into two equivalent parts. Thus, both parts contain the same information as the original set. Ladner [23] introduced and studied mitoticity of recursively enumerable sets. Ambos-Spies [2] formulated two notions in the polynomial time setting, mitoticity and weak mitoticity. Also, he showed that every mitotic set is autoreducible. Here we settle some questions about mitoticity that Buhrman and Torenvliet raised in their 1994 survey paper. First, Buhrman, Hoene, Torenvliet [11] proved that all EXP-complete sets are weakly many-one mitotic, and showed a partial result concerning mitoticity of NEXP-complete sets. Here we strengthen these two results. We prove that all EXP-complete sets are many-one mitotic. We also prove that all NEXP complete sets are weakly many-one mitotic. In addition, we show that PSPACE-complete sets are weakly Turing mitotic. Also, we show that if one-way permutations and quick pseudo-random generators exist, then NP-complete sets are many-one mitotic.

In Section 5, we study the question of whether NP-complete sets have easy subsets. Berman [9] showed that EXP-complete sets are not P-immune, and Tran [29] showed that NEXP-complete sets are not P-immune. However, we do not have such unconditional results for NP-complete sets. Glaßer *et al.* [18] showed that if one-way permutations exist, then NP-complete sets are not 2^{n^ϵ}-immune. Here we provide another partial result in this direction. In Section 5 we show that if there exists a tally language in $\mathrm{NP} \cap \mathrm{coNP} - \mathrm{P}$, then every NP-complete set includes an infinite subset that is recognizable in time $2^{n^{(1+\epsilon)}}$.

We conclude this paper with results on a few additional properties of complete sets. In Section 6 we show, under a reasonable hypothesis about the complexity class UP, that every NP-complete set is exponentially-honest complete. Exponentially-honest reductions were defined and studied by Ganesan and Homer [17]. Section 7 studies robustness of complete sets. We prove that if a set A is Turing complete for NP and S is a log-dense set, then $A - S$ remains Turing complete for NP. This result is easier to prove than Buhrman and Torenvliet's [14] result about EXP.

2 Preliminaries

We use standard notation and assume familiarity with standard resource-bounded reductions. We consider words in lexicographic order. All used reductions are polynomial-time computable.

Definition 1 ([2]). *A set A is* polynomially T-autoreducible *(T-autoreducible, for short) if there exists a polynomial-time-bounded oracle Turing machine M such that $A = L(M^A)$ and for all x, M on input x never queries x. A set A is* polynomially m-autoreducible *(m-autoreducible, for short) if $A \leq_m^p A$ via a reduction function f such that for all x, $f(x) \neq x$.*

Definition 2 ([2]). *A recursive set A is* polynomial-time m(T)-mitotic *(m(T)-mitotic, for short) if there exists a set $B \in \mathrm{P}$ such that $A \equiv^p_{m(T)} A \cap B \equiv^p_{m(T)} A \cap \overline{B}$. A recursive set A is* polynomial-time weakly m(T)-mitotic *(weakly m(T)-mitotic, for short) if there exist disjoint sets A_0 and A_1 such that $A_0 \cup A_1 = A$, and $A \equiv^p_{m(T)} A_0 \equiv^p_{m(T)} A_1$.*

In general, for a reducibility type r, r-autoreducible sets are sets that are autoreducible with respect to $\leq^p_r$-reductions. The same convention is used for mitotic sets.

A language is DTIME($T(n)$)-*complex* if L does not belong to DTIME($T(n)$) almost everywhere; that is, every Turing machine M that accepts L runs in time greater than $T(|x|)$, for all but finitely many words x. A language L is *immune* to a complexity class $\mathcal{C}$, or $\mathcal{C}$-*immune*, if L is infinite and no infinite subset of L belongs to $\mathcal{C}$. A language L is *bi-immune* to a complexity class $\mathcal{C}$, or $\mathcal{C}$-*bi-immune*, if both L and $\overline{L}$ are $\mathcal{C}$-*immune*. Balcázar and Schöning [6] proved that for every time-constructible function T, L is DTIME($T(n)$)-complex if and only if L is bi-immune to DTIME($T(n)$).

3 Autoreducibility

Since EXP-complete sets are complete with respect to length-increasing reductions [9], they are m-autoreducible. Ganesan and Homer [17] showed that NEXP-complete sets are complete under 1-1 reductions. This implies that all NEXP-complete sets are also m-autoreducible. To see this, consider a 1-1 reduction from $0L \cup 1L$ to L, where L is the given NEXP-complete set. These techniques cannot be applied to NP-complete sets, as we do not know any unconditional result on the degree structure of NP-complete sets. Some partial results are known for NP-complete sets. Beigel and Feigenbaum [7] showed that Turing completes for NP are T-autoreducible. Buhrman et al. [10] showed that all truth-table complete sets for NP are probabilistic tt-autoreducible. It has not been known whether NP-complete sets are m-autoreducible. Buhrman and Torenvliet raised this question in their survey papers [12, 13]. Below, we resolve this question.

Note that singletons and complements of singletons cannot be m-autoreducible. Therefore, in connection with m-autoreducibility, a set L is called *nontrivial* if $|L| > 1$ and $|\overline{L}| > 1$.

Theorem 1. *All nontrivial* NP*-complete sets are m-autoreducible.*

Proof. Let L be NP-complete and let M be a nondeterministic machine that accepts L. For a suitable polynomial p we can assume that on input x, all computation paths of M have length $p(|x|)$. Since L is nontrivial, there exist different words $y_1, y_2 \in L$ and $\overline{y}_1, \overline{y}_2 \in \overline{L}$. We use the left set technique of Ogiwara and Watanabe [24]. Let

$$\mathrm{Left}(L) \overset{df}{=} \{\langle x, u\rangle \mid |u| = p(|x|) \text{ and } \exists v, |v| = |u|, \text{such that } u \leq v \text{ and } M(x) \text{ accepts along path } v\}.$$

Let $f \in \mathrm{PF}$ reduce Left(L) to L. The algorithm below defines function g which is an m-autoreduction for L. Let x be an input. Define $n \overset{df}{=} |x|$ and $m \overset{df}{=} p(|x|)$.

```
1   if $f(\langle x, 0^m\rangle) \neq x$ then output $f(\langle x, 0^m\rangle)$
2   if $f(\langle x, 1^m\rangle) = x$ then
3       if M(x) accepts along $1^m$ then
4           output a string from $\{y_1, y_2\} - \{x\}$
5       else
6           output a string from $\{\overline{y}_1, \overline{y}_2\} - \{x\}$
7       endif
```

```
8   endif
9   // here $f(\langle x, 0^m\rangle) = x \neq f(\langle x, 1^m\rangle)$
10  determine z of length m such that $f(\langle x, z\rangle) = x \neq f(\langle x, z+1\rangle)$
11  if M(x) accepts along z then output a string from $\{y_1, y_2\} - \{x\}$
12  else output $f(\langle x, z+1\rangle)$
```

Note that step 10 is an easy binary search: Start with $z_1 := 0^m$ and $z_2 := 1^m$. Let z' be the middle element between z_1 and z_2. If $f(z') = x$ then $z_1 := z'$ else $z_2 := z'$. Again, choose the middle element between z_1 and z_2, and so on. This shows $g \in \mathrm{PF}$. Clearly, $g(x) \neq x$, so it remains to show $L \leq_m^p L$ via g.

If the algorithm stops in step 1, then

$$x \in L \Leftrightarrow \langle x, 0^m\rangle \in \mathrm{Left}(L) \Leftrightarrow g(x) = f(\langle x, 0^m\rangle) \in L.$$

If the algorithm stops in step 4 or step 6, then $f(\langle x, 0^m\rangle) = f(\langle x, 1^m\rangle)$. Hence

$$x \in L \Leftrightarrow \langle x, 1^m\rangle \in \mathrm{Left}(L) \Leftrightarrow M(x) \text{ accepts along } 1^m \Leftrightarrow g(x) \in L.$$

Assume we reach step 9. There it holds that $f(\langle x, 0^m\rangle) = x \neq f(\langle x, 1^m\rangle)$. If the algorithm stops in step 11, then $x \in L$ and $g(x) \in L$. Assume we stop in step 12. So $M(x)$ does not accept along z.

$$x \in L \Leftrightarrow f(\langle x, 0^m\rangle) = f(\langle x, z\rangle) \in L \Leftrightarrow g(x) = f(\langle x, z+1\rangle) \in L.$$

□

Using the same technique we obtain the following results:

Corollary 1. *For every $k \geq 1$, all nontrivial sets that are $\leq_{k\text{-}dtt}^p$-complete for* NP *are $\leq_{k\text{-}dtt}^p$-autoreducible.*

Corollary 2. *All nontrivial sets that are $\leq_{dtt}^p$-complete for* NP *are $\leq_{dtt}^p$-autoreducible.*

Corollary 3 below is the analog of Corollaries 1 and 2 for $\leq_{1\text{-}tt}^p$-reductions. It is unknown whether the analog holds for $\leq_{2\text{-}tt}^p$-reductions, even for $\leq_{2\text{-}ctt}^p$-reductions.

Corollary 3. *All nontrivial sets that are $\leq_{1\text{-}tt}^p$-complete for* NP *are $\leq_{1\text{-}tt}^p$-autoreducible.*

Theorem 1 shows that every nontrivial NP-complete set is m-autoreducible. We show that the same result holds for PSPACE.

Theorem 2. *Each nontrivial m-complete language for* PSPACE *is m-autoreducible.*

Proof. Let S_5 be a permutation group over $\{1, \ldots, 5\}$. Cai and Furst [16] show that for each language L in PSPACE there exist a polynomial p and a polynomial-time computable function f from $\Sigma^* \times \Sigma^*$ to S_5 such that for all x $x \in L$ if and only if

$$f(x, 1^{p(|x|)}) \circ \cdots \circ f(x, 0^{p(|x|)}) = I_5,$$

where $\circ$ is the product of permutations calculated from right to left and I_5 is the identity permutation. Let L be an arbitrary m-complete language for PSPACE. Suppose that the membership of L in PSPACE is characterized by p and f. Define A to be the set of all triples $\langle x, y, \pi\rangle$ such that $|y| = p(|x|)$, $\pi \in S_5$, and

$$f(x, 1^{p(|x|)}) \circ \cdots \circ f(x, y) = \pi.$$

This set A clearly belongs to PSPACE. Also, L is $\leq_m^p$-reducible to A via a function that maps each x to $\langle x, 0^{p(|x|)}, I_5\rangle$. So, A is m-complete for PSPACE. Let g be an m-reduction from A to L.

We construct an m-autoreduction h of L as follows: Let x be an input to h. For each y, $|y| = p(|x|)$, and $\pi \in S_5$, let $Q(y, \pi) = g(\langle x, y, \pi\rangle)$. We first compute $z = Q(0^{p(|x|)}, I_5)$. We have $x \in L \Leftrightarrow \langle x, 0^{p(|x|)}, I_5\rangle \in A$ and $z \in L \Leftrightarrow \langle x, 0^{p(|x|)}, I_5\rangle \in A$. So, $x \in L \Leftrightarrow z \in L$. If $z \neq x$, we set $h(x) = z$.

So, assume $z = x$. We then compute $Q(1^{p(|x|)}, \pi), \pi \in S_5$. Suppose for some π, $Q(1^{p(|x|)}, \pi) = x$. Then it holds that $x \in L \Leftrightarrow \langle x, 1^{p(|x|)}, \pi\rangle \in A$. By definition of A, $\langle x, 1^{p(|x|)}, \pi\rangle$ belongs to A if and only if $f(x, 1^{p(|x)}) = \pi$. Since f is polynomial-time computable, this condition can be tested in polynomial time. We select the value of $h(x)$ from a set of fixed members and nonmembers so that $h(x) \neq x$ and so that $h(x) \in L$ if and only if $f(x, 1^{p(|x|)}) = \pi$.

So, assume that for all $\pi \in S_5$, $Q(1^{p(|x|)}, \pi) \neq x$. We execute binary search to find y, $0^{p(|x|)} \leq y < 1^{p(|x|)}$, such that x appears in $\{Q(y, \pi) \mid \pi \in S_5\}$ and doesn't appear in $\{Q(y+1, \pi) \mid \pi \in S_5\}$, where $y + 1$ is the successor of y in $\Sigma^{p(|x|)}$. Such y indeed exists. Since g is polynomial-time computable, the y can be computed in polynomial time. Let $\xi \in S_5$ be such that $x = Q(y, \xi)$. Since g is a many-one reduction from A to L, $x \in L$ if and only if

$$f(x, 1^{p(|x|)}) \circ \cdots \circ f(x, y + 1) \circ f(x, y) = \xi.$$

Let σ be the inverse permutation of $f(x, y)$. Since f is polynomial-time computable and every permutation has a unique inverse, σ can be computed in polynomial time. By multiplying the equation in the above by σ from right, we have $x \in L$ if and only if

$$f(x, 1^{p(|x|)}) \circ \cdots \circ f(x, y + 1) = \xi \circ \sigma.$$

Let $z = Q(y+1, \xi\circ\sigma)$. We have $z \in L$ if and only if the above equality holds. So, $x \in L \Leftrightarrow z \in L$. Since x does not appear in $\{Q(y + 1, \pi) \mid \pi \in S_5\}$, $x \neq z$. So, we set $h(x)$ to z. This completes the proof. □

We generalize the above two results using the concept of polynomial-time bit-reductions [19]. We show that every class that is polynomial-time bit-reducible to a regular language has the property that all of its nontrivial m-complete sets are m-autoreducible. As a corollary to this, we show that for every class in the MODPH hierarchy [19], all of its nontrivial m-complete sets are m-autoreducible.

Definition 3. *[19] A language A is polynomial-time bit-reducible to a language B if there exists a pair of polynomial-time computable functions, (f, g), $f : \Sigma^* \times \mathbf{N}^+ \to \Sigma$, $g : \Sigma^* \to \mathbf{N}$, such that for all x, $x \in A \Leftrightarrow f(x, 1) \cdots f(x, g(x)) \in B$.*

Theorem 3. *Let $B \notin \{\emptyset, \Sigma^*\}$ be a regular language recognized by a finite automaton $M = (Q, \{0, 1\}, \delta, q_0, q_f)$. Let $\mathcal{C}$ be the polynomial-time bit-reduction closure of B. Then each nontrivial m-complete set for $\mathcal{C}$ is m-autoreducible.*[5]

Definition 4. *[27, 31] Let $\mathcal{C}$ be a language class. Define $\exists \cdot \mathcal{C}$ (and analogously $\forall \cdot \mathcal{C}$) to be the set of all languages L for which there exist a polynomial p and an $A \in \mathcal{C}$ such that for all x,*

$$x \in L \Leftrightarrow (\exists y, |y| = p(|x|))[\langle x, y\rangle \in A].$$

[5] In terms of leaf languages, this theorem reads as follows: If B is a nontrivial regular language, then m-complete sets for $\mathrm{Leaf}_b^p(B)$ are m-autoreducible. (The latter denotes the balanced leaf-language class defined by B [19, 30].)

Definition 5. *[8, 26] Let $k \geq 2$ be an integer. Define $\mathrm{MOD}_k \cdot \mathcal{C}$ to be the set of all languages L for which there exist a polynomial p and a language $A \in \mathcal{C}$ such that for all x,*

$$x \in L \Leftrightarrow \|\{y \mid |y| = p(|x|) \wedge \langle x, y\rangle \in A\}\| \not\equiv 0 \pmod{k}.$$

Definition 6. *[19] MODPH is the hierarchy consisting of the following classes:*

- P *belongs to* MODPH.
- *If $\mathcal{C}$ is a class belonging to* MODPH *then $\exists \cdot \mathcal{C}$ and $\forall \cdot \mathcal{C}$ belong to* MODPH.
- *For each integer $k \geq 2$, if $\mathcal{C}$ is a class belonging to* MODPH *then $\mathrm{MOD}_k \cdot \mathcal{C}$ and $\mathrm{coMOD}_k \cdot \mathcal{C}$ belong to* MODPH.

Proposition 1. *[19] Each class in* MODPH *is the polynomial-time bit-reduction closure of some regular language.*

Theorem 4. *For every class in* MODPH *it holds that all of its nontrivial m-complete sets are m-autoreducible.*

Corollary 4. *Every nontrivial set that is m-complete for one of the following classes is m-autoreducible.*

- *the levels Σ_k^P, Π_k^P, and Δ_k^P of the polynomial-time hierarchy*
- 1NP
- *the levels of the Boolean hierarchy over* NP

Now we obtain corollaries about the genericity and dimension of complete sets. The notion of resource bounded genericity was defined by by Ambos-Spies, Fleischhack, and Huwig [3]. We use the following equivalent definition [5, 25].

Definition 7. *Let t be a polynomial. A set L is $t(n)$-*generic *if for every oracle Turing machine M, if $M^{L|x}(x) = L(x)$ for all x, then the running time of M is at least $t(2^{|x|})$ for all but finitely many x. Recall that $L|x = \{y \in L \mid y < x\}$.*

We obtain the following corollary regarding genericity of NP-complete sets. Earlier, it was known that there exists a $k > 0$, such that NP-complete sets are not $O(n^k)$-generic. This follows from the work on small span theorems [22, 4]. Below we improve this.

Corollary 5. NP*-complete sets are not $2n$-generic.*

Corollary 6. *Let $\mathcal{C}$ be either* PSPACE *or a class belonging to* MODPH. *Then no m-complete sets for $\mathcal{C}$ are $2b$-generic.*

Recently, Hitchcock [20] showed that the (-3)-rd order scaled dimension of NP-complete sets is zero. His proof uses a small span theorem for (-3)-rd order dimension. We can obtain Hitchcock's result using Theorem 1.

Corollary 7. *The class of* NP*-complete sets has (-3)-rd order dimension zero.*

3.1 Relativization

We obtain an oracle satisfying the following properties:

Theorem 5. *For any $k \geq 2$, there is an oracle A such that relative to A there is a set B that is $\leq^p_{k\text{-}dtt}$ complete for* NP *but not $\leq^p_{(k-1)\text{-}T}$ autoreducible.*

Proof. See Appendix.□

In light of this oracle it probably is not possible to improve Corollary 1. Also, this theorem is interesting because of the following corollary:

Corollary 8. *There is an oracle A such that relative to A there exists a $\leq^p_{2\text{-}dtt}$-complete set for* NP *that is not m-mitotic.*

Corollary 8 gives a relativized, partial negative answer to an open question of Buhrman and Torenvliet [12] as to whether all T-complete sets for NP are (weakly) $m(T)$-mitotic. It remains open whether there is an oracle relative to which not all NP-complete sets are m-mitotic. Another related question raised by Buhrman and Torenvliet is whether all tt-complete sets for NP are tt-autoreducible. The following theorem gives a partial answer to this question in a relativized world.

Theorem 6. *For any $k \geq 2$, there is an oracle A such that relative to A, there is a set B that is $\leq^p_{dtt}$ complete for* NP *but not $\leq^p_{btt}$-autoreducible.*

4 Mitoticity

Buhrman, Hoene, and Torenvliet [11] show that all EXP-complete sets are weakly m-mitotic. We improve this to show that all EXP-complete sets are m-mitotic. We want to remark that Buhrman and Torenvliet [15] cite a private communication with Kurtz for the first proof of this result.

Theorem 7. *All* EXP*-complete sets are m-mitotic.*

Actually, the following holds: for any class $\mathcal{C}$, if L is complete with length-increasing reductions, then L is m-mitotic. Agrawal [1] shows that if one-way permutations and quick pseudo-random generators exist, then every NP-complete language is complete with respect to length increasing reductions. Thus,

Corollary 9. *If one-way permutations and quick pseudo-random generators exist, then* NP*-complete languages are m-mitotic.*

If L is NP-complete with respect to honest many-one reductions, then L is complete with respect to length increasing reductions. Thus we have the following corollary.

Corollary 10. *If L is* NP*-complete with respect to honest reductions, then L is m-mitotic.*

Buhrman, Hoene, and Torenvliet [11] showed that every NEXP-complete set can be partitioned into infinitely many sets such that each one of them is NEXP-complete. They ask whether this can be improved to show that every NEXP-complete set is weakly m-mitotic. Theorem 9 answers this question in the affirmative. For the proof, we need the following result due to Ganesan and Homer [17]. They showed that all NE-complete sets are complete with respect to 1-1 reductions. Their proof works for NEXP-complete sets also.

Theorem 8 ([17]). *All $\leq_m^p$-complete sets for* NEXP *are complete under 1-1 reductions.*

Theorem 9. *All $\leq_m^p$-complete sets for* NEXP *are weakly m-mitotic.*

Proof. Let K be the standard NEXP-complete set. Let L be any given NEXP-complete set. We show that L is weakly m-mitotic.

$$K' \overset{df}{=} 0K \cup 1K$$

is NEXP-complete. K' reduces to L via some $f \in \mathrm{PF}$. L reduces to K (really K, not K') via some $g \in \mathrm{PF}$. Choose k such that f and g can be computed in time $O(n^k)$. By Theorem 8, we can assume f and g to be 1-1.

$$L_0 \overset{df}{=} \{y \mid \exists x, |x| \leq |y|^k, f(0x) = y\}$$
$$L_1 \overset{df}{=} \{y \mid \exists x, |x| \leq |y|^k, f(1x) = y\}$$

Observe that $L_0, L_1 \in \mathrm{EXP}$. Define the following function:

$$f_0(x) \overset{df}{=} \begin{cases} f(0x) & : \text{ if } |f(0x)|^k \geq |x|, \\ f_0(g(f(0x))) & : \text{ otherwise.} \end{cases}$$

Note that if $|f(0x)|^k < |x|$ then $|g(f(0x))| \leq |f(0x)|^k < |x|$. So, for some y, $|y| < |x|$, $f_0(x) = f_0(y)$. So, the recursion terminates. Thus, $f_0 \in \mathrm{PF}$. Note that for every x there exists some y such that $f_0(x) = f(0y)$ and $|f(0y)|^k \geq |y|$. This implies that for all x, $f_0(x) \in L_0$.

Similarly define f_1 as

$$f_1(x) \overset{df}{=} \begin{cases} f(1x) & : \text{ if } |f(1x)|^k \geq |x|, \\ f_1(g(f(1x))) & : \text{ otherwise.} \end{cases}$$

By following an argument similar to the one above we can show that $f_1 \in \mathrm{PF}$ and that $f_1(\Sigma^*) \subseteq L_1$.

We first show $K \leq_m^p L$ via reduction f_0. This is done by induction on the number of recursion steps r in the definition of $f_0(x)$. If $r = 0$, then $f_0(x) = f(0x)$ and therefore,

$$x \in K \Leftrightarrow 0x \in 0K \Leftrightarrow 0x \in K' \Leftrightarrow f_0(x) = f(0x) \in L.$$

If $r \geq 1$ then $f_0(x) = f_0(g(f(0x)))$. Let $y = g(f(0x))$. By our induction hypothesis, $y \in K \Leftrightarrow f_0(y) \in L$. So we obtain

$$\begin{aligned} x \in K \Leftrightarrow 0x \in 0K \Leftrightarrow 0x \in K' \Leftrightarrow f(0x) \in L \\ \Leftrightarrow y = g(f(0x)) \in K \Leftrightarrow f_0(y) = f_0(g(f(0x))) = f_0(x) \in L. \end{aligned}$$

Thus, $K \leq_m^p L$ via f_0. Analogously we show $K \leq_m^p L$ via f_1.

Since $K \leq_m^p L$ via f_0 and $f_0(\Sigma^*) \subseteq L_0$ we have $K \leq_m^p (L \cap L_0)$ via f_0. Since f is 1-1, L_0 and L_1 are disjoint. Since $f_1(\Sigma^*) \subseteq L_1$, we have $f_1(\Sigma^*) \subseteq \overline{L_0}$. Combining this with $K \leq_m^p L$ via f_1, we have $K \leq_m^p (L \cap \overline{L_0})$ via f_1. Therefore $(L \cap L_0)$ and $(L \cap \overline{L_0})$ are each NEXP-hard. Both sets are NEXP-complete since $L_0 \in \mathrm{EXP}$ and $L \in \mathrm{NEXP}$. □

Note that the above proof actually shows that each NEXP-complete set can be split into m-equivalent NEXP-complete m-mitotic parts with a set in EXP.

The following theorem settles another question raised by Buhrman and Torenvliet [12].

Theorem 10. *Every $\leq_m^p$-complete set for* PSPACE *is weakly T-mitotic.*

Proof. See Appendix.□

5 Immunity

In Glaßer et al. [18], the authors proved immunity results for NP-complete sets under the assumption that certain average-case hardness conditions are true. For example, they show that if one-way permutations exist, then NP-complete sets are not 2^{n^ϵ}-immune. Here we obtain a non-immunity result for NP-complete sets under the assumption that the following worst-case hardness hypothesis holds.

Hypothesis T: There is an NP machine N that accepts 0^* and no P-machine can compute its accepting computations. This means that for every polynomial-time machine M there exist infinitely many n such that $M(0^n)$ is not an accepting computation of $N(0^n)$.

Though the hypothesis looks verbose, we note that it is implied by a simply stated and believable hypothesis.

Observation 1 *If there is a tally language in* $\mathrm{NP} \cap \mathrm{coNP} - \mathrm{P}$*, then Hypothesis T is true.*

Theorem 11. *If Hypothesis T holds, then, for every* $\epsilon > 0$*,* NP*-complete languages are not* $2^{n(1+\epsilon)}$*-immune.*

Proof. See Appendix.□

Corollary 11. *If there is a tally language in* $\mathrm{NP}\cap\mathrm{coNP}-\mathrm{P}$*, then, for every* $\epsilon > 0$*,* NP*-complete languages are not* $2^{n(1+\epsilon)}$*-immune.*

Next we consider the possibility of obtaining an unconditional result regarding non-immunity of NP-complete languages. We show that for certain type of NP-complete languages we get an unconditional result.

Definition 8. *A language L does not have superpolynomial gaps, if there exists $k > 0$ such that for all but finitely many n, there exists a string x in L such that $n \leq |x| \leq n^k$.*

We show that NP-complete languages that have no superpolynomial gaps are not immune.

Theorem 12. *If L is an* NP*-complete language that has no superpolynomial gaps, then for every $\epsilon > 0$, L is not* $2^{n(1+\epsilon)}$*-immune.*

The above theorem prompts the following question: Are there NP-complete languages with superpolynomial gaps? We have the following result. Given two complexity classes $\mathcal{A}$ and $\mathcal{B}$, we say $\mathcal{A} \subseteq io\text{-}\mathcal{B}$, if for every language $A \in \mathcal{A}$, there exists a language $B \in \mathcal{B}$ such that for infinitely many n, $A^n = B^n$.

Theorem 13. *If* NP *has a complete language with superpolynomial gaps, then for every $\epsilon > 0$,* $\mathrm{NP} \subseteq io\text{-}\mathrm{DTIME}(2^{n^\epsilon})$.

Combining Theorems 12 and 13 we have the following corollary.

Corollary 12. *If for some $\delta > 0$,* $\mathrm{NP} \not\subseteq io\text{-}\mathrm{DTIME}(2^{n^\delta})$*, then for every $\epsilon > 0$, no* NP*-complete language is* $2^{n(1+\epsilon)}$*-immune.*

6 Exponentially Honest Reductions

In the previous section we showed that if $\mathrm{NP} \cap \mathrm{coNP} - \mathrm{P}$ has a tally language, then there exist reductions from 0^* to NP-complete sets that are infinitely often exponentially honest. In this section, we consider a stronger hypothesis and show that if this hypothesis holds, then every NP-complete set is complete with respect to exponentially honest reductions.

We first make a few observations regarding Lemma 1 and its proof.

Corollary 13. *Assume there exists an* NP *machine that accepts* 0^* *such that no* P*-machine can compute infinitely many accepting computations of* M*. Then for every* NP*-complete language* L *and for every* $k > 0$*, there is a polynomial-time many-one reduction* f *from* 0^* *to* L *such that for all but finitely many* n, $|f(0^n)| > k \log n$.

Corollary 14. *Assume there exists an* NP *machine that accepts* 0^* *such that no* 2^{2n}*-time-bounded machine can compute infinitely many accepting computations of* M*. Let* L *be any* NP*-complete language, and* S *be any set in* NP*. For every* $k > 0$*, there is a polynomial-time computable, partial function* f *such that*

(i) $f(x)$ *is defined for all* $x \in S$,
(ii) *if* $f(x)$ *is defined then* $(x \in S \Leftrightarrow f(x) \in L)$ *and* $|f(x)| > k \log n$*, and*
(iii) *if* $f(x)$ *is undefined then* $x \notin S$*. Moreover, in polynomial time we can decide whether* $f(x)$ *is defined or not.*

Now we consider the following hypothesis.

UP-machine Hypothesis: For some $\epsilon > 0$, there is a UP-machine M that accepts 0^* such that no 2^{n^ϵ}-time-bounded machine can compute infinitely many accepting computations of M.

We first make the following observation.

Observation 2 *If the* UP*-machine hypothesis holds, then, there exists a* UP*-machine* M *that accepts* 0^* *such that no* 2^{2n}*-time-bounded machine can compute infinitely many accepting computations of* M.

Theorem 14. *Assume the* UP*-machine hypothesis holds. Let* L *be any* NP*-complete language. For every* $S \in \mathrm{NP}$ *and every* $k > 0$*, there is a many-one reduction* f *from* S *to* L *such that for every* x, $|f(x)| > k \log |x|$.

7 Robustness

Recently Buhrman and Torenvliet [14] proved that Turing-complete sets for EXP are robust against log-dense sets in P. Using similar ideas, we can prove the same for NP. The proof is easier though, due to the fact that search reduces to decision for all Turing-complete sets for NP [21]. A set S is *log-dense* if there is a constant $c > 0$ such that for all n, $\|S^{\leq n}\| \leq c \log n$.

Theorem 15. *If a set* A *is Turing-complete for* NP *and* S *is a* log*-dense set in* P*, then* $A - S$ *is Turing-complete for* NP.

Proof. Let A be Turing complete for NP and let S be a log-dense set in P. Then $A - S$ belongs to NP, because S belongs to P. Now we show that A is Turing reducible to $A - S$ in polynomial time. Consider the machine T that reduces search to decision for A in time $p(n)$, for some polynomial p. Let $c > 0$ be a constant such that for all n, $\|S^{\leq n}\| \leq c \log n$. Then the following machine M is a polynomial time Turing reduction from A to $A - S$:

```
Input $x$
For all bit strings $s$ of length $c \log p(|x|)$ do
    $k \longleftarrow 0$;
    Repeat
        Simulate T on x until T makes the next query q;
        If $q \notin S$ then continue simulation with YES iff $q \in A - S$;
        Else
            Continue the simulation with YES iff $s_k = 1$;
            //$s_k$ is the $k$-th bit of $s$
            $k \longleftarrow k + 1$;
    Until $T$ outputs $y_s$ or rejects;
    If $T$ outputs $y_s$ and $y_s$ is a witness of $x$ then ACCEPT;
Endfor;
REJECT.
```

If $q \notin S$, then a query to $A - S$ has the same answer as a query to A. So in this case M's simulation of T behaves correctly. At most $c \log p(|x|)$ queries q belong to S. For each of these queries, the bit string s determines the response. If $x \notin A$, then no witness y_s for x exists. So in this case M does not accept x. If $x \in A$, then the bit string s that represents the correct answers to the queries that are in S will make T output a correct witness y_s. Hence, M accepts x. Therefore, the algorithm is correct.

For the running time, there are at most $p(|x|)^c$ iterations of the outer loop, each of which is a simulation of T on x, which costs $p(|x|)$ time. Hence, the total running time is bounded by a polynomial in $|x|$. $\square$

References

1. M. Agrawal. Pseudo-random generators and structure of complete degrees. In *17th Annual IEEE Conference on Computational Complexity*, pages 139–145, 2002.
2. K. Ambos-Spies. P-mitotic sets. In E. Börger, G. Hasenjäger, and D. Roding, editors, *Logic and Machines, Lecture Notes in Computer Science 177*, pages 1–23. Springer-Verlag, 1984.
3. K. Ambos-Spies, H. Fleischhack, and H. Huwig. Diagonalizations over polynomial time computable sets. *Theoretical Computer Science*, 51:177–204, 1987.
4. K. Ambos-Spies, H. Neis, and A. Terwijn. Genericity and measure for exponential time. *Theoretical Computer Science*, 168(1):3–19, 1996.
5. J. Balcazar and E. Mayordomo. A note on genericty and bi-immunity. In *Proceedings of the Tenth Annual IEEE Conference on Computational Complexity*, pages 193–196, 1995.
6. J. Balcázar and U. Schöning. Bi-immune sets for complexity classes. *Mathematical Systems Theory*, 18(1):1–18, 1985.
7. R. Beigel and J. Feigenbaum. On being incoherent without being very hard. *Computational Complexity*, 2:1–17, 1992.
8. R. Beigel and J. Gill. Counting classes: Thresholds, parity, mods, and fewness. *Theoretical Computer Science*, 103:3–23, 1992.
9. L. Berman. *Polynomial Reducibilities and Complete Sets.* PhD thesis, Cornell University, Ithaca, NY, 1977.
10. 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.
11. H. Buhrman, A. Hoene, and L. Torenvliet. Splittings, robustness, and structure of complete sets. *SIAM Journal on Computing*, 27:637–653, 1998.
12. H. Buhrman and L. Torenvliet. On the structure of complete sets. In *Proceedings 9th Structure in Complexity Theory*, pages 118–133, 1994.

13. H. Buhrman and L. Torenvliet. Complete sets and structure in subrecursive classes. In *Proceedings of Logic Colloquium '96*, pages 45–78. Springer-Verlag, 1998.
14. H. Buhrman and L. Torenvliet. Separating complexity classes using structural properties. In *Proceedings of the 19th IEEE Conference on Computational Complexity*, pages 130–138, 2004.
15. H. Buhrman and L. Torenvliet. A Post's program for complexity theory. *Bulleting of the EATCS*, 85:41–51, 2005.
16. J.-Y. Cai and M. Furst. PSPACE survives constant-width bottlenecks. *International Journal of Foundations of Computer Science*, 2:67–76, 1991.
17. K. Ganesan and S. Homer. Complete problems and strong polynomial reducibilities. *SIAM Journal on Computing*, 21:733–742, 1992.
18. C. Glaßer, A. Pavan, A. Selman, and S. Sengupta. Properties of NP-complete sets. In *Proceedings of the 19th Annual IEEE Conference on Computational Complexity*, pages 184–197, 2004.
19. U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, and K. W. Wagner. On the power of polynomial time bit-reductions. In *Proceedings 8th Structure in Complexity Theory*, pages 200–207, 1993.
20. J. Hitchcock. Small spans in scaled dimension. In *19th IEEE Conference on Computational Complexity*, pages 104–112, 2004.
21. S. Homer and A. Selman. *Computability and Complexity Theory.* Texts in Computer Science. Springer, New York, 2001.
22. D. W. Juedes and J. H. Lutz. The complexity and distribution of hard problems. *SIAM Joutnal on Computing*, 24:279–295, 1995.
23. R. Ladner. Mitotic recursively enumerable sets. *Journal of Symbolic Logic*, 38(2):199–211, 1973.
24. M. Ogiwara and O. Watanabe. On polynomial-time bounded truth-table reducibility of NP sets to sparse sets. *SIAM Journal of Computing*, 20(3):471–483, 1991.
25. A. Pavan and A. Selman. Separation of NP-completeness notions. *SIAM Journal on Computing*, 31(3):906–918, 2002.
26. U. Schöning. Probabilistic complexity classes and lowness. *Journal of Computer and System Sciences*, 39:84–100, 1989.
27. L. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.
28. B. Trahtenbrot. On autoreducibility. *Dokl. Akad. Nauk SSSR*, 192, 1970. Translation in Soviet Math. Dokl. 11: 814– 817, 1970.
29. N. Tran. On P-immunity of nondeterministic complete sets. In *Proceedings of the 10th Annual Conference on Structure in Complexity Theory*, pages 262–263. IEEE Computer Society Press, 1995.
30. K. W. Wagner. Leaf language classes. In *Proceedings International Conference on Machines, Computations, and Universality*, volume 3354 of *Lecture Notes in Computer Science*. Springer Verlag, 2004.
31. C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3:23–33, 1977.
32. A. Yao. Coherent functions and program checkers. In *Proceedings of the 22nd Annual Symposium on Theory of Computing*, pages 89–94, 1990.

Appendix

Proof of Theorem 5

Proof. Let $k' = \lceil \log k \rceil$. Without loss of generality, we assume k' is odd (the proof for the case where k' is even is essentially the same). Define G to be the set consisting of the first k strings of length k'. We assume a polynomial-time computable one-to-one pairing function that can take any finite number of inputs such that its range does not intersect with 0^* and such that for all i, x, and l, $|\langle i, x, 0^l \rangle| > l$. Such pairing function can be found very easily. Let $\{M_j\}_{j \geq 1}$ be an enumeration of polynomial-time $(k-1)$-Turing reductions. Let $\{N_i\}_{i \geq 1}$ be an enumeration of all nondeterministic polynomial-time oracle Turing machines. For each $j \geq 1$, let p_j be a polynomial that bounds the running time of both M_j and N_j.

For any set A, let $K^A = \{\langle i, x, 0^l \rangle \mid N_i^A \text{ accepts } x \text{ within } l \text{ steps }\}$ be the canonical complete set for NP^A. Now define

$$B_e^A = \{s\langle i, x, 0^l \rangle \mid s \in G \wedge [\exists y[|y| = |\langle i, x, 0^l \rangle| \wedge s\langle i, x, 0^l \rangle y \in A]]\},$$

$$B_d^A = \{0^{2n} \mid \exists y[|y| = 2n \wedge y \in A]\},$$

and

$$B^A = B_e^A \cup B_d^A.$$

Clearly B_e^A, B_d^A and B^A all belong to NP^A. We will construct A such that for all but finitely many combinations of i, x, and l,

$$\langle i, x, 0^l \rangle \in K^A \Leftrightarrow \exists s[s \in G \wedge s\langle i, x, 0^l \rangle \in B^A],$$

which implies $K^A \leq^p_{k\text{-}dtt} B^A$. Also, we will construct A such that for every j, there exists an even number n_j such that

$$M_j^A \text{ accepts } 0^{n_j} \text{ with oracle } B^A \Leftrightarrow 0^{n_j} \notin B^A$$

or

$$M_j^A \text{ on input } 0^{n_j} \text{ queries } 0^{n_j} \text{ to oracle } B^A,$$

which will ensure that M_j is not an $\leq^p_{(k-1)\text{-}T}$ autoreduction of B.

We construct A in stages. We initialize $n_1 = 2$, so that $p_1(n_1) < 2^{n_1}$, and define $A^{<2} = \emptyset$. At Stage j, we assume the construction has been done up to strings of length $n_j - 1$, where n_j is some even number chosen in Stage $j-1$. Let $A_{j-1} = A^{\leq n_j - 1}$. Now the construction for Stage j proceeds in two steps. In the first step, we will *diagonalize* against M_j. If $M_j^{A_{j-1}}$ on input 0^{n_j} does not query the string 0^{n_j} to $B_{A_{j-1}}$, then we will find a string y of length n_j such that $M_j^{A_{j-1} \cup \{y\}}$, making $k-1$ queries to $B^{A_{j-1} \cup \{y\}}$ accepts 0^{n_j} if and only if $0^{n_j} \notin B^{A_{j-1} \cup \{y\}}$. In the second step, we will do the *encoding*. We will choose some sufficiently large even number n_{j+1} and make sure, for every $\langle i, x, 0^l \rangle$ such that $n_j < |s\langle i, x, 0^l \rangle y| < n_{j+1}$, $s \in G$, and $|y| = |\langle i, x, 0^l \rangle|$, that $\langle i, x, 0^l \rangle \in K^A$ if and only if $\exists s \exists y[s \in G \wedge |y| = |\langle i, x, 0^l \rangle \wedge s\langle i, x, 0^l \rangle y \in A$. This will ensure for those $\langle i, x, 0^l \rangle$ that $\langle i, x, 0^l \rangle \in K^A$ if and only if $\exists s[s \in G \wedge s\langle i, x, 0^l \rangle \in B^A]$.

We first show that the string y of length n_j needed for the diagonalization step exists. We use a set N_A for reserving strings of length greater than or equal to n_j for $\overline{A}$. N_A is set to $\emptyset$ at the beginning of each stage. We then simulate M_j on input 0^{n_j} and resolve each query in the following way. For each query q made to A, we answer YES if and only if $q \in A_{j-1}$. If $|q| \geq n_j$, set N_A to $N_A \cup \{q\}$. For each query q made to B^A, if $q = 0^{n_j}$, we proceed to the encoding step. (In this case, M_j is not an autoreduction.) Now $q \neq 0^{n_j}$. We answer q with YES if and only if $q \in B^{A_{j-1}}$. For those q's such that q is of the form 0^{2n}, where $2n > n_j$, we set N_A to

$N_A \cup \{y \| y| = |q|\}$, which will ensure that $q = 0^{2n} \notin B^A$. For those q's such that $q = s\langle i, x, 0^l\rangle$, $s \in G$, and $k' + 2|\langle i, x, 0^l\rangle| > n_j$, we set N_A to $N_A \cup \{s\langle i, x, 0^l\rangle y \mid |y| = |\langle i, x, 0^l\rangle|\}$, which will ensure that $q = s\langle i, x, 0^l\rangle \notin B^A$. Observe that strings of length n_j (an even length) can only be put into N_A if they are queried by M_j to A. So $\|N_A \cap \Sigma^{\leq n_j}\| \leq p_j(n_j)$ since M_j is p_j-time bounded. We will see below that in Stage $j-1$, n_j was chosen such that $p_j(n_j) < 2^{n_j} = \|\Sigma^{n_j}\|$. So we choose a string y of length n_j such that $y \notin N_A$. For this y, $M_j^{A_{j-1}}(0^{n_j})$, which makes $k-1$ queries to $B^{A_{j-1}}$, accepts if and only if $M_j^{A_{j-1}\cup\{y\}}(0^{n_j})$, making $k-1$ queries to $B^{A_{j-1}\cup\{y\}}$, accepts. Thus we add y to A_{j-1} if and only if $M_j^{A_{j-1}}(0^{n_j})$ rejects with oracle $B^{A_{j-1}}$. This completes the diagonalization step.

Now we need to do the encoding step. At the same time we want to maintain the diagonalization properties of the oracle from the diagonalization step. This can be ensured by not putting any string in N_A into A. However, by doing so we have to make sure encoding can be done up to length $\max_{y \in N_A} |y|$. Also we need $p_j(n_j) < 2^{n_j}$ for the diagonalization step at any Stage j. To fulfill all these requirements, we choose n_{j+1} to be the minimum even number such that the following hold:

- $p_{j+1}(n_{j+1}) < 2^{(n_{j+1}-k')/2}$
- $n_{j+1} > n_j$
- $n_{j+1} > \max_{y \in N_A} |y|$

Now we encode each triple $\langle i, x, 0^l\rangle$ such that $n_j < |s\langle i, x, 0^l\rangle y| < n_{j+1}$, $s \in G$, and $|y| = |\langle i, x, 0^l\rangle|$. We have to decide whether to put some string $s\langle i, x, 0^l\rangle y$ into A according to whether $\langle i, x, 0^l\rangle \in K^A$, so as to make sure $\langle i, x, 0^l\rangle \in K^A$ if and only if $\exists s[s \in G \wedge s\langle i, x, 0^l\rangle \in B^A]$. As we said earlier, this task is nontrivial since some of these strings might have already been put into N_A. We need to show that there exist strings that are unreserved for each triple $\langle i, x, 0^l\rangle$, to be used for encoding. We will also need a set Y_A that contains strings reserved for A during the encoding process. Now we use the following procedure in Fig. 1 to do the encoding from length $n_j + 1$ to $n_{j+1} - 1$.

```
1 $Y_A := \emptyset$;
  //$N_A$ is initially the set constructed in the diagonalization step.
2 $Q_j := \{\langle i, x, 0^l\rangle \mid n_j < k' + 2|\langle i, x, 0^l\rangle| < n_{j+1}\}$;
  // set of triples $\langle i, x, 0^l\rangle$ that we need to consider for encoding.
3 For all triples $\langle i, x, 0^l\rangle$ in $Q_j$ in lexicographical order do
4     Simulate $N_i$ on $x$ for $l$ steps with oracle $A_{j-1} \cup Y_A$;
5     If $N_i$ accepts then
6         Choose $s\langle i, x, 0^l\rangle y \notin N_A$, such that $s \in G$ and $|y| = |\langle i, x, 0^l\rangle|$;
7         $Y_A := Y_A \cup \{s\langle i, x, 0^l\rangle y\}$;
8     Else
9         $N_A := N_A \cup \{s\langle i, x, 0^l\rangle y \mid s \in G \wedge |y| = |\langle i, x, 0^l\rangle|\}$;
10 EndFor;
11 $A_j := A_{j-1} \cup Y_A$;
```

Fig. 1. Procedure for the encoding step at Stage j in Theorem 5 (comments start with //).

The correctness of the above encoding procedure follows from the following claims.

Claim 1 *For every $\langle i, x, 0^l\rangle \in Q_j$ in the above procedure, there exists a string $s\langle i, x, 0^l\rangle y \notin N_A$ such that $s \in G$ and $|y| = |\langle i, x, 0^l\rangle|$, to select when step 6 is executed.*

Proof. We look back at the diagonalization step. For each query q made to A by M_j, we put q into N_A only if $|q| \geq n_j$. Since there are at most $p_j(n_j)$ many such queries due to the fact that M_j is p_j-time bounded, we put at most $p_j(n_j)$ many strings into N_A in because of queries to A. Now for each $\langle i, x, 0^l \rangle \in Q_j$, for each $s \in G$ consider $S_s = \{s\langle i, x, 0^l\rangle y || y| = |\langle i, x, 0^l\rangle|\}$. For each query q made to B^A by M_j, at most one such S_s might be put into N_A. Since M_j only makes $k-1$ queries to B^A, there must exist one S_s such that no string in S_s is put into N_A in due to queries made to B^A by M_j. Let S_s be such a set. Note that $\|S_s\| = 2^{|\langle i,x,0^l\rangle|} \geq 2^{(n_j-k')/2} > p_j(n_j)$, where the last inequality follows from the choice of n_j at Stage $j-1$. So there must exist $z \in S_s$ that is not put into N_A because of queries made to A by M_j. Hence, z is a string of the form $s\langle i, x, 0^l\rangle y$ such that $s \in G$ and $|y| = |\langle i, x, 0^l\rangle|$, and $z \notin N_A$ by the end of the diagonalization step. Now during the encoding step, we will only put into N_A strings of the form $s\langle i', x', 0^{l'}\rangle y$, where $\langle i', x', 0^{l'}\rangle \neq \langle i, x, 0^l\rangle$, upon processing triples $\langle i', x', 0^{l'}\rangle$ that are lexicographically less than $\langle i, x, 0^l\rangle$. So z cannot have been put into N_A during the encoding step by the time the triple $\langle i, x, 0^l\rangle$ is processed. Hence, the claim follows. □

Claim 2 *For any $\langle i, x, 0^l\rangle \in Q_j$ in the above procedure,*

$$\langle i, x, 0^l\rangle \in K^{A_j} \Leftrightarrow \exists s[s \in G \wedge s\langle i, x, 0^l\rangle \in B^{A_j}].$$

Proof. First note that the following equivalence holds after the triple $\langle i, x, 0^l\rangle \in Q_j$ is processed:

$$\langle i, x, 0^l\rangle \in K^{A_{j-1}\cup Y_A} \Leftrightarrow \exists s[s \in G \wedge s\langle i, x, 0^l\rangle \in B^{A_{j-1}\cup Y_A}]. \quad (*)$$

For every triple $\langle i', x', 0^{l'}\rangle \in Q_j$ that is processed after $\langle i, x, 0^l\rangle$, either a string in the form $s'\langle i', x', 0^{l'}\rangle y'$ such that $s' \in G$ and $|y'| = |\langle i', x', 0^{l'}\rangle|$ is put into Y_A or all the members of the set $\{s'\langle i', x', 0^{l'}\rangle y' \mid s' \in G \wedge |y'| = |\langle i', x', 0^{l'}|\}$ are put into N_A. Neither of these will change the truth value of the right-hand side of $(*)$. Also note that all the strings added to Y_A or N_A after $\langle i, x, 0^l\rangle$ is processed are of length at least $k' + 2|\langle i, x, 0^l\rangle| > l$. So the truth value of the left side of $(*)$ will not change either after $\langle i, x, 0^l\rangle$ is processed since the first l steps of N_i's computation on x makes queries of length at most l. Therefore, $(*)$ becomes a loop invariant once the triple $\langle i, x, 0^l\rangle$ is processed. Since by the end of the loop we have $A_j = A_{j-1} \cup Y_A$, it follows that

$$\langle i, x, 0^l\rangle \in K^{A_j} \Leftrightarrow \exists s[s \in G \wedge s\langle i, x, 0^l\rangle \in B^{A_j}].$$

□

Claim 3 *For every triple $\langle i, x, 0^l\rangle$ such that $|\langle i, x, 0^l\rangle| > 2$,*

$$\langle i, x, 0^l\rangle \in K^{A} \Leftrightarrow \exists s[s \in G \wedge s\langle i, x, 0^l\rangle \in B^{A}].$$

Proof. Let $|\langle i, x, 0^l\rangle| > 2$. Then $\langle i, x, 0^l\rangle$ belongs to Q_j, for some $j \geq 1$. Then, by Claim 2, that

$$\langle i, x, 0^l\rangle \in K^{A_j} \Leftrightarrow \exists s[s \in G \wedge s\langle i, x, 0^l\rangle \in B^{A_j}].$$

We know from the above construction that only strings of length greater than or equal to n_j will be added into A after Stage j and $n_j > k' + 2|\langle i, x, 0^l\rangle| > l$ for any $\langle i, x, 0^l\rangle \in Q_j$. Hence

$$\begin{aligned} &\langle i, x, 0^l\rangle \in K^{A} \\ \Leftrightarrow\ &\langle i, x, 0^l\rangle \in K^{A_j} \\ \Leftrightarrow\ &\exists s[s \in G \wedge s\langle i, x, 0^l\rangle \in B^{A_j}] \\ \Leftrightarrow\ &\exists s[s \in G \wedge s\langle i, x, 0^l\rangle y \in B^{A}]. \end{aligned}$$

□

This finishes the proof of Theorem 5. □

Proof of Theorem 10

Proof. Let L be $\leq_m^p$-complete for PSPACE. There exists $f \in \mathrm{PF}$ such that $L \leq_m^p \overline{L}$ via f. f clearly has the property that for all x $f(x) \neq x$. Choose $k \geq 1$ such that f is computable in time $n^k + k$.

Let t be a "tower" function defined by: $t(0) \overset{df}{=} -1$, $t(1) \overset{df}{=} 2$, and $t(i+1) \overset{df}{=} t(i)^{k+1}$ for $i \geq 1$. Define the inverse tower function as $t^{-1}(n) \overset{df}{=} \min\{i \mid t(i) \geq n\}$. Note that $t^{-1} \in \mathrm{PF}$.

We first sketch the basic ideas of the proof. We partition Σ^* into sets $A, B \in \mathrm{PSPACE}$ such that L, $L \cap A$, and $L \cap B$ are Turing-equivalent. It essentially suffices to show $L \leq_T^p L \cap B$. Given any x, we consider the sequence $x, f(x), f(f(x)), \ldots$. These elements alternatively belong to L and $\overline{L}$. So in order to determine whether $x \in L$ it suffices to find out for an arbitrary $n \geq 0$ whether $f^n(x) \in L$. We define A and B such that for every x the first polynomially many words in the sequence contain an alternation between A and B; that is, it contains at least one member of A and at least one member of B.

We embed in each sequence "anchors" with the property that if y is an anchor and $f(y)$ isn't one then one of y and $f(y)$ is a member of A and the other is a member of B, and with the property that the membership in A as well as the membership in B of any anchor can be tested in polynomial time.

We consider the following three cases:

Case 1: The first polynomially many elements of the sequence are all anchors. We make sure that in this case the length of the words in the sequence is strictly increasing. This leads us to an alternation in the sequence (since an anchor belongs to A if and only if the inverse tower function of its length is even). So within the first polynomially many words there are anchors $f^a(x) \in B$. Asking the oracle whether $f^a(x) \in L \cap B$ tells us whether $f^a(x) \in L$.

Case 2: Within the first polynomially many words of the sequence, there exists a non-anchor that is followed by an anchor. In this case we will be able to identify the anchor easily. Exactly one of both words belongs to B, and we can easily find out which one (since testing membership in B is easy for anchors). Say $f^n(x) \in B$. Again with help of the oracle we can find out whether $f^n(x) \in L$.

Case 3: There are no anchors within the first polynomially many words. However, A and B are constructed such that every second word in the sequence causes an alternation between A and B. In contrast to Case 1 and Case 2, here we do not know the exact positions of these alternations. Nevertheless, we can gain information as follows: We test 9 consecutive words from the sequence for membership in $L \cap B$. We know that two of these words belong to B such that one is at an even position in the sequence, while the other one is at an odd position. By the choice of f, the sequence strictly alternates between L and $\overline{L}$. So at least one of the 9 consecutive words belongs to $L \cap B$. This gives an n such that $f^n(x) \in L$.

This suffices to show that $L \leq_T^p L \cap B$.

Now we present the proof in full detail.

For every word x, we define its accompanying anchor $h(x)$ to be $f^{g(x)}(x)$, where $g(x)$ is given as follows: If there exists an m such that $|f^{m+1}(x)| > |f^m(x)|$ then $g(x)$ is the smallest such m; otherwise, $g(x)$ is the smallest n such that $f^n(x) = \min\{y \mid \text{for some } i \text{ and } j,\ i \neq j,\ y = f^i(x) = f^j(x)\}$. If the word-length never increases in the sequence $x, f(x), f^2(x), \ldots$, then it necessarily enters a cycle. Then $h(x)$ is the smallest word in the cycle and $g(x)$ is the smallest index that produces $h(x)$. Note that both g and h are computable in PSPACE.

Call a word x an *anchor* if $h(x) = x$. Observe that for every x, $h(x)$ is an anchor.

Claim 4 *For every x, x is an anchor if and only if $g(x) = 0$,* If $g(x) = 0$, then $h(x) = x$ and x is an anchor. Assume $g(x) > 0$. If $g(x) = m$ according to the first case of g's definition, then $|f^{m+1}(x)| > |f^m(x)|$ but $|f(x)| \leq |x|$. Therefore, $h(x) = f^m(x) \neq x$ and x is not an anchor. If $g(x) = n$ according to the second case of g's definition, then for all $i \geq 0$, $|f^{i+1}(x)| \leq |f^i(x)|$. So the sequence $x, f(x), f(f(x)), \ldots$ leads into a cycle, and n (in g's definition) is the smallest number such that $f^l(x)$ belongs to this cycle. Note that $f^n(x)$ is the lexicographically minimal word of the cycle. Observe that $f^n(x) \neq x$, since otherwise $g(x) = n = 0$ which contradicts our assumption. Therefore, $h(x) = f^n(x) \neq x$ and x is no anchor. This proves Claim 4.

Claim 5 *For every non-anchor x, $g(x) = g(f(x)) + 1$.*

By Claim 4, $g(x) > 0$. Assume $g(x) = m$ according to the first case of g's definition. So $m > 0$ is the smallest number such that $|f^m(f(x))| > |f^{m-1}(f(x))|$. Therefore, $g(f(x)) = m - 1$ and hence $g(x) = g(f(x)) + 1$. Assume $g(x) = n$ according to the second case of g's definition. So the sequence $x, f(x), f(f(x)), \ldots$ leads into a cycle. By definition, $n > 0$ is the smallest number such that $f^{n-1}(f(x))$ is the lexicographically minimal word of the cycle. Therefore, $g(f(x)) = n - 1$ and hence $g(x) = g(f(x)) + 1$. This shows Claim 5.

We work toward the definition of A and B. For Case 3 we have to make sure that for every x, every second word in the sequence $x, f(x), f(f(x)), \ldots$ causes an alternation between A and B. Additionally, Case 2 needs such an alternation whenever a non-anchor is followed by an anchor. The following set of natural numbers provides the desired alternation structure.

$$N \overset{df}{=} \{n \mid n = 0 \text{ or } n \equiv 2 \pmod 4 \text{ or } n \equiv 3 \pmod 4\}$$

We also must take Case 1 into account which calls for an alternation whenever we have a sequence of polynomially many anchors. By the definition of anchors it is clear that a sequence of consecutive anchors is strictly length-increasing. So we have to make sure that in such a sequence once in a while we alternate between A and B. We do this by partitioning Σ^* according to the inverse tower function. For $i \in \{0, 1\}$ let

$$S_i \overset{df}{=} \{x \mid t^{-1}(|x|) \equiv i \pmod 2\}.$$

Observe that $S_0, S_1 \in \mathrm{P}$. Now let $A \overset{df}{=} (A_0 \cup A_1)$ and $B \overset{df}{=} (B_0 \cup B_1)$ where

$$\begin{aligned} A_0 &\overset{df}{=} \{x \mid h(x) \in S_0 \text{ and } g(x) \in N\}, \\ B_0 &\overset{df}{=} \{x \mid h(x) \in S_0 \text{ and } g(x) \notin N\}, \\ A_1 &\overset{df}{=} \{x \mid h(x) \in S_1 \text{ and } g(x) \notin N\}, \text{ and} \\ B_1 &\overset{df}{=} \{x \mid h(x) \in S_1 \text{ and } g(x) \in N\}. \end{aligned}$$

Observe that (A_0, A_1, B_0, B_1) form a partition of Σ^*. Hence (A, B) is a partition as well. $A_0, B_0, A_1, B_1, A, B \in \mathrm{PSPACE}$.

In every sequence $x, f(x), f(f(x)), \ldots$ either we can find a non-anchor (Cases 2 and 3) or we can find an anchor in B (Case 1). The following function takes care of this (note that an anchor belongs to B if and only if it belongs to S_1).

$$r(x) \overset{df}{=} \min\{i \mid |f^{i+1}(x)| \leq |f^i(x)| \text{ or } f^i(x) \in S_1\}$$

In Claim 7 below we will see that r is a total function. For the moment let us assume $r(x)$ to be infinite if the minimum in r's definition does not exist.

Claim 6 *For every x and $0 \leq i < r(x)$, $|f^i(x)| < |x|^{k+1}$.*

Assume there exist x and i such that $0 \leq i < r(x)$ and $|f^i(x)| \geq |x|^{k+1}$. So $t^{-1}(|x|) < t^{-1}(|x|^{k+1}) \leq t^{-1}(|f^i(x)|)$ which shows that $|x|$ and $|f^i(x)|$ have different inverse tower functions. Hence there must exist $j < i$ such that $|f^j(x)|$ and $|f^{j+1}(x)|$ have different inverse tower functions. Note that $f^j(x), f^{j+1}(x) \in S_0$; otherwise, $r(x) \leq j+1 \leq i$, which is not possible by the choice of i. This means $t^{-1}(|f^{j+1}(x)|) - t^{-1}(|f^j(x)|) \geq 2$ and therefore, $|f^{j+1}(x)| > |f^j(x)|^{k+1}$. This contradicts f's computation time and proves Claim 6.

Claim 7 *For every x, $r(x) \leq |x|^{k+1}$. Particularly, r is a total function in* FP.

Assume there exists x such that $r(x) > |x|^{k+1}$. Let $n \overset{df}{=} |x|^{k+1}$ and consider the sequence $f^0(x), f^1(x), \ldots, f^n(x)$. By assumption, $f^i(x) \notin S_1$ for $i \leq n$ and $|f^0(x)| < |f^1(x)| < \cdots < |f^n(x)|$. Hence $|f^n(x)| \geq n = |x|^{k+1}$ which contradicts Claim 6. So for every x, $r(x) \leq |x|^{k+1}$. Particularly, r is total. Containment in FP follows from Claim 6. This proves Claim 7.

The polynomial-time algorithm given in Figure 2 accepts an input x if and only if $x \in L$. For every x, this algorithm asks at most 9 queries to its oracle $L \cap B$. In particular, $L \leq_{btt}^{p} L \cap B$.

```
n := $r(x)$
if $|f^{n+1}(x)| > |f^n(x)|$ then
    // here $f^n(x) \in S_1$, $g(f^n(x)) = 0$, and $f^n(x) \in B_1$
    accept if and only if [$f^n(x) \in L \cap B \Leftrightarrow n \equiv 0 \pmod 2$]
else
    Q := $\{f^{n+i}(x) \mid 0 \leq i \leq 8\}$
    if $\forall i \in [1,8]$, $|f^{n+i}(x)| \geq |f^{n+i+1}(x)|$ then
        // here for all $i \in [0,8]$, $|f^{n+i}(x)| \geq |f^{n+i+1}(x)|$
        if $\|Q\| = 9$ then
            choose smallest $j \in [0,8]$ such that $f^{n+j}(x) \in L \cap B$
            accept if and only if $n + j \equiv 0 \pmod 2$
        else
            // $\exists i \in [0,7], \exists j \in [(i+1),8]$ such that $f^{n+i}(x) = f^{n+j}(x)$
            choose the smallest such i
            choose $l \in [(i+1),8]$ s.t. $f^{n+l}(x) = \min\{f^{n+j}(x) \mid j \in [i,8]\}$
            a := $n + l$
        endif
    else
        // Q has an anchor whose length increases under f
        choose smallest $i \in [0,8]$ such that $|f^{n+i}(x)| < |f^{n+i+1}(x)|$
        a := $n + i$
    endif
    // $a \geq 1$, $f^a(x)$ is an anchor, and $f^{a-1}(x)$ is no anchor
    if $f^a(x) \in S_1$ then
        // here $f^a(x) \in S_1$ and therefore $f^a(x) \in B_1$
        accept if and only if [$f^a(x) \in L \cap B \Leftrightarrow a \equiv 0 \pmod 2$]
    else
        // here $f^a(x) \in S_0$ and therefore $f^{a-1}(x) \in B_0$
        accept if and only if [$f^{a-1}(x) \in L \cap B \Leftrightarrow a \equiv 1 \pmod 2$]
    endif
endif
```

Fig. 2. Algorithm for $L \leq_{btt}^{p} L \cap B$ in Theorem 10.

Claim 8 *The algorithm works in polynomial time.*

By Claim 7, $n \leq |x|^{k+1}$ after step 1. From Claim 6 it follows that for $0 \leq i \leq 8$, $f^{n+i}(x)$ can be computed in polynomial time. This shows that all steps of the algorithm can be carried out in polynomial time. This proves Claim 8.

Claim 9 *If the algorithm stops in step 4, then it accepts if and only if $x \in L$.*

If we reach step 3, then by the definition of $r(x)$, $f^n(x) \in S_1$. Note that $f^n(x)$ is an anchor, since its length increases under f. Hence $h(f^n(x)) = f^n(x) \in S_1$ and $g(f^n(x)) = 0 \in N$. Therefore, $f^n(x) \in B_1 \subseteq B$ and hence

$$\begin{aligned} x \in L &\Leftrightarrow [f^n(x) \in L \Leftrightarrow n \equiv 0 \pmod 2] \\ &\Leftrightarrow [f^n(x) \in L \cap B \Leftrightarrow n \equiv 0 \pmod 2]. \end{aligned}$$

This proves Claim 9.

Claim 10 *If the algorithm stops in step 11, then it accepts if and only if $x \in L$.*

Step 11 can only be reached via step 8. Observe that in step 8,

$$\forall i \in [0,8], |f^{n+i}(x)| \geq |f^{n+i+1}(x)|. \tag{1}$$

$\|Q\| = 9$ once we reach step 10. So for $i \in [0,8]$, all $f^{n+i}(x)$ are pairwise different.

First we want to see that if we apply g to the elements of Q, then this yields at least 5 consecutive natural numbers. Formally, we claim that there exists $t \in [0,4]$ such that

$$\forall i \in [t, t+3], g(f^{n+i}(x)) = g(f^{n+i+1}(x)) + 1. \tag{2}$$

If Q does not contain anchors, then by Claim 5, for $i \in [0,7]$, $g(f^{n+i}(x)) = g(f^{n+i+1}(x)) + 1$ and we are done by choosing $t = 0$. If Q contains anchors, then by (1), these must be anchors whose lengths do not increase under f. All these anchors are in the same cycle (see second case of g's definition). Since every cycle has exactly one anchor, Q must contain exactly one anchor. Say the anchor is $f^{n+j}(x)$ where $j \in [0,8]$. If $j \leq 3$, then for $i \in [4,8]$, $f^{n+i}(x)$ is no anchor. So (2) holds for $t = 4$. If $j \geq 4$, then (2) holds for $t = 0$. This proves equation (2).

From (1) it follows that all $w \in Q$ have the same anchor $h(w)$. Choose $t \in [0,4]$ such that (2) holds. By the definitions of A, B, and N, the following can be verified, by using the periodical structure of N and the property about the five consecutive indices, as shown in (2).[6]

1. $\exists i \in [t, t+4]$ such that i is odd and $f^{n+i}(x) \in A$
2. $\exists i \in [t, t+4]$ such that i is even and $f^{n+i}(x) \in A$
3. $\exists i \in [t, t+4]$ such that i is odd and $f^{n+i}(x) \in B$
4. $\exists i \in [t, t+4]$ such that i is even and $f^{n+i}(x) \in B$

Choose $i, j \in [t, t+4] \subseteq [0,8]$ that satisfy items 3 and 4, respectively. Since i is odd and j is even, one of $f^{n+i}(x)$ and $f^{n+j}(x)$ has to be in L, since f alternates between L and $\overline{L}$. So either $f^{n+i}(x)$ or $f^{n+j}(x)$ belongs to $L \cap B$. Therefore, the choice in step 10 is possible. After step 10, $f^{n+j}(x) \in L$ and therefore, $x \in L \Leftrightarrow n + j \equiv 0 \pmod 2$. This proves Claim 10.

Claim 11 *In step 23, $a \geq 1$, $f^a(x)$ is an anchor, and $f^{a-1}(x)$ is not an anchor.*

[6] This is the point where we need the fact that N alternates at every second number (except for 0 and 1).

We can reach step 23 either via step 13 or via step 19. Assume we reach it via step 13. We already observed that (1) holds in step 8. If we reach step 13, then there must exist i, j such that $0 \leq i < j \leq 8$ and $f^{n+i}(x) = f^{n+j}(x)$. So Q contains a cycle, and in step 14 we choose the first element that belongs to this cycle. In step 15 we choose l such that $f^{n+l}(x)$ is the smallest element of the cycle. Hence $f^{n+l}(x) = f^a(x)$ is an anchor. Note that $a \geq 1$. Moreover, $f^{a-1}(x)$ already belongs to the cycle and is therefore no anchor.

Assume now that we reach step 23 via step 19. If we reach step 19, then by the condition in step 7, in Q there exists an anchor whose length increases under f. We choose the first such anchor we reach. After step 21, this anchor can be described as $f^a(x)$. Note that $a \geq 1$, since $i > 0$ by the condition in step 2. Moreover, $|f^{a-1}(x)| \geq |f^a(x)|$ which in turn implies that $f^{a-1}(x)$ is not an anchor. This proves Claim 11.

Claim 12 *The algorithm accepts L.*

By Claims 9 and 10 it remains to argue for the case that the algorithm stops in step 26 or in step 29.

Assume we stop in step 26. Since $f^a(x)$ is an anchor, $h(f^a(x)) = f^a(x)$ and $g(f^a(x)) = 0 \in N$. Hence $f^a(x) \in B_1 \subseteq B$, and thus,

$$\begin{aligned} x \in L &\Leftrightarrow [f^a(x) \in L \Leftrightarrow a \equiv 0 \pmod 2] \\ &\Leftrightarrow [f^a(x) \in L \cap B \Leftrightarrow a \equiv 0 \pmod 2]. \end{aligned}$$

Assume we stop in step 29. $f^a(x)$ belongs to S_0, since (S_0, S_1) is a partition of Σ^*. We know that $f^{a-1}(x)$ is no anchor and $f^a(x)$ is an anchor. Hence $h(f^{a-1}(x)) = f^a(x) \in S_0$ and $g(f^{a-1}(x)) = 1 \notin N$.[7] It follows that $f^{a-1}(x) \in B_0 \subseteq B$. Thus,

$$\begin{aligned} x \in L &\Leftrightarrow [f^{a-1}(x) \in L \Leftrightarrow a \equiv 1 \pmod 2] \\ &\Leftrightarrow [f^{a-1}(x) \in L \cap B \Leftrightarrow a \equiv 1 \pmod 2]. \end{aligned}$$

This proves Claim 12.

From Claim 12 it immediately follows that $L \leq^p_{btt} L \cap B$. In a similar way we show that $L \leq^p_{btt} L \cap A$. For this we only have to modify a few things in the proof above. We have to define r as

$$r(x) \stackrel{df}{=} \min\{i \,\big|\, |f^{i+1}(x)| \leq |f^i(x)| \text{ or } f^i(x) \in S_0\}.$$

Claims 6, 7, 8, 9, 11, and 12 can be proved analogously for A instead of B. In the proof of Claim 10 we have to use the items 1 and 2 instead of 3 and 4. For completeness, we present in Figure 3 the $\leq^p_{btt}$-reduction from L to $L \cap A$.

It follows that L, $L \cap A$, and $L \cap B$ are Turing-equivalent. Therefore, L is weakly T-mitotic. This completes the proof of Theorem 10. □

[7] This is the point where we need the fact that N starts with an alternation, i.e., $0 \in N$ and $1 \notin N$. Suppose we defined $N = \{2, 3, 6, 7, 10, 11, \ldots\}$ and let $w_i \stackrel{df}{=} f^{n+i}(x)$. Then it could happen that $|w_0| = 10$, $|w_1| = 9$, $|w_2| = 10$, $|w_3| = 9$, and so on. The words $w_1, w_3, w_5, \ldots$ are anchors. Assume that for all i, $h(w_i) \in S_1$. So in the whole sequence $w_0, w_1, w_2, \ldots$ there is no word that belongs to B.

```
n := r(x)
if $|f^{n+1}(x)| > |f^{n}(x)|$ then
    // here $f^{n}(x) \in S_0$, $g(f^{n}(x)) = 0$, and $f^{n}(x) \in A_0$
    accept if and only if $[f^{n}(x) \in L \cap A \Leftrightarrow n \equiv 0 \pmod 2]$
else
    $Q := \{f^{n+i}(x) \mid 0 \leq i \leq 8\}$
    if $\forall i \in [1, 8]$, $|f^{n+i}(x)| \geq |f^{n+i+1}(x)|$ then
        // here for all $i \in [0, 8]$, $|f^{n+i}(x)| \geq |f^{n+i+1}(x)|$
        if $\|Q\| = 9$ then
            choose smallest $j \in [0, 8]$ such that $f^{n+j}(x) \in L \cap A$
            accept if and only if $n + j \equiv 0 \pmod 2$
        else
            // $\exists i \in [0, 7], \exists j \in [(i + 1), 8]$ such that $f^{n+i}(x) = f^{n+j}(x)$
            choose the smallest such i
            choose $l \in [(i + 1), 8]$ s.t. $f^{n+l}(x) = \min\{f^{n+j}(x) \mid j \in [i, 8]\}$
            $a := n + l$
        endif
    else
        // Q has an anchor whose length increases under f
        choose smallest $i \in [0, 8]$ such that $|f^{n+i}(x)| < |f^{n+i+1}(x)|$
        $a := n + i$
    endif
    // $a \geq 1$, $f^{a}(x)$ is an anchor, and $f^{a-1}(x)$ is no anchor
    if $f^{a}(x) \in S_0$ then
        // here $f^{a}(x) \in S_0$ and therefore $f^{a}(x) \in A_0$
        accept if and only if $[f^{a}(x) \in L \cap A \Leftrightarrow a \equiv 0 \pmod 2]$
    else
        // here $f^{a}(x) \in S_1$ and therefore $f^{a-1}(x) \in A_1$
        accept if and only if $[f^{a-1}(x) \in L \cap A \Leftrightarrow a \equiv 1 \pmod 2]$
    endif
endif
```

Fig. 3. Algorithm for $L \leq^{p}_{btt} L \cap A$ in Theorem 10.

Proof of Theorem 11

Proof. Let L be any NP-complete language. We first show that there is a reduction from 0^* to L that is infinitely often exponentially honest. We then use this fact to show that L is not $2^{n(1+\epsilon)}$-immune.

Lemma 1. *Assume that the hypothesis holds. For every* NP *complete language L, there exists a constant $c > 0$, such that for every $k' > 0$, there exists a reduction f from 0^* to L such that for infinitely many n, $|f(0^n)| > k' \log n$. The reduction f can be computed in time $O(n^{k'+c})$.*

Proof. Let M be the NP machine from the hypothesis. Let a_n be the lexicographically maximum accepting computation of M on 0^n. Consider the set

$$S = \{\langle 0^n, y\rangle \mid y \leq a_n\}.$$

It is obvious that S is in NP. Let g be a many-one reduction from S to L. We now describe a reduction from 0^* to L with the desired properties. Let T be the computation tree of M on 0^n. Without loss of generality, assume T is a complete binary tree, and let d denote the depth of T. The reduction traverses T in stages. At stage k it maintains a list of nodes at level k. Initially at Stage 1, $list_1$ contains the root of the tree. The reduction also maintains a variable called $next$, whose value is initially undefined. We now describe Stage $k > 1$.

1. Let $list_{k-1} = \langle u_1, u_2, \ldots, u_l\rangle$. Let $v_1, v_2, \ldots, v_{2l}$ be the children of nodes in $list_{k-1}$. Assume $v_1 < v_2 < \cdots < v_{2l}$. Set $list_k = \langle v_1, \ldots, v_{2l}\rangle$.
2. Consider the first j such that $|g(\langle 0^n, v_j\rangle)| > k' \log n$.
3. If such j exists, then let $list_k = \langle v_1, v_2, \ldots, v_{j-1}\rangle$ and $next = v_j$.
4. Prune $list_k$, i.e., if there exist $i < r$ such that $g(\langle 0^n, v_i\rangle) = g(\langle 0^n, v_r\rangle)$, then remove $v_i, v_{i+1}, \ldots, v_{r-1}$ from $list_k$.

The following is the desired reduction f from 0^* to L. Let x be the input and let $n = |x|$.

- If $x \neq 0^n$, then output a fixed string not in L.
- Run Stages $1, \ldots, d$.
- All nodes in $list_d$ are leaf nodes. If $list_d$ contains an accepting node, then output a fixed string in L, else output $g(\langle 0^n, next\rangle)$.

We claim that the above reduction has the desired properties. It is obvious that the reduction is correct on non-tally strings. So we focus only on tally strings. We show a series of claims that help us in showing the correctness of the reduction.

We first make a couple of observations.

Observation 3 *Consider Step 4 of Stage k, if a node v_m is removed from $list_k$, then the rightmost accepting computation of M on 0^n does not pass through v_m.*

Proof. Step 4 removes nodes $v_i, \ldots, v_{r-1}$ from $list_k$, if there exist $i < r$ such that $g(\langle 0^n, v_i\rangle) = g(\langle 0^n, v_r\rangle)$. Assume that the right most accepting computation passes through a node v_m such that $i \leq m \leq r-1$. By definition of S, $\langle 0^n, v_i\rangle \in S$ and $\langle 0^n, v_r\rangle \notin S$. However, g is a many-one reduction from S to L, and $g(\langle 0^n, v_i\rangle) = g(\langle 0^n, v_r\rangle)$. This is a contradiction. □

Observation 4 *Let k be the first stage at which the variable next is defined. For every $r \geq k$ and for all $u \in list_r$ it holds that $u < next$.*

Proof. We prove the claim by induction. Assume *next* is defined for the first time during Stage k. This happens in Step 3, where $next = v_j$ and $list_k = \langle v_1, \ldots v_{j-1} \rangle$. In Step 4 some nodes from $list_k$ are deleted. Since $v_1 < v_2 < \cdots < v_{j-1} < v_j = next$, the claim holds after Stage k. We now show that if the claim holds at the beginning of Stage r, then it holds at the end of Stage r. Consider Stage r. At the beginning of this stage, $list_r = \langle u_1, \ldots, u_l \rangle$ where $u_i < next$ for $1 \leq i \leq k$. During this stage the value of *next* may or may not change. If the value of *next* changes, then it happens in Step 3. Here we set $next = v_j$, $list_r$ will be a subset of $\{v_1, \ldots, v_{j-1}\}$, thus the claim holds after the stage. Suppose the value of *next* does not change. At the end of the Stage r, $list_r = \langle v_1, \cdots, v_m \rangle$ and every node v in $list_r$ is a child of some node u_i. Since $u_i < next$ and v is a child of u_i, we have $v < next$. □

Claim 13 *For every $k > 1$, one of the following statements holds at the end of Stage k.*

A *next is undefined, and the rightmost accepting computation of M on 0^n passes through a node in $list_k$.*

B *next is defined and the rightmost accepting computation passes through a node in $list_k$.*

C *next is defined, and the rightmost accepting computation either passes through next or lies to the right of next.*

Proof. We will show that if one of the three statements hold before the beginning of Stage k, then one of the three statements, possibly different, hold at the end of Stage k. Initially, at the beginning of Stage 2, $list_2$ contains *root* and *next* is undefined, so Statement **A** holds. Let $k \geq 2$ and assume that one of the statements holds at the beginning of Stage k. We consider three cases.

Case 1: Statement **A** holds. Let $list_k = \langle u_1, \ldots, u_l \rangle$ at the beginning of the stage. After Step 2 $list_k$ becomes $\langle v_1, v_2, \ldots, v_{2l} \rangle$. Since the v_i's are children of the u_i's, the rightmost accepting computation must pass through a node v_r that is in $list_k$. Consider Step 3. If there is no j such that $g(\langle 0^n, v_j \rangle)| > k' \log n$, then *next* remains undefined at the end of Stage k, and no changes are made to list in Step 3. Thus $v_r \in list_k$ after Step 3. In Step 4, we apply pruning. By Observation 3, v_r is not removed from $list_k$ during Step 5. Thus at the end of the Stage k, the rightmost accepting computation passes v_r and $v_r \in list_k$, and so at the end of Stage k, Statement **A** holds.

Now consider the case where the procedure finds j such that $|g(\langle 0^n, v_j \rangle)| > k' \log n$. This sets *next* to v_j. If $j \leq r$, then v_r is removed from $list_k$. However, since $v_j \leq v_r$, the rightmost accepting computation either passes through *next* or lies to the right of *next*. Statement **C** thus holds. If $j > r$, then after Step 3 $v_r \in list_k$. By Observation 3, v_r remains in the list after Stage 4, and so Statement **B** holds at the end of Stage k.

Case 2: Statement **B** holds at the beginning of Stage k. This implies that after Step 2 the rightmost accepting computation passes through a node v_r in $list_k$. Again, there are three possibilities for Step 2: (i) no such j is found, (ii) j is defined and $j \leq r$, and (iii) j is defined and $j > r$. By following an argument similar to that of the previous case, we can show that one of the statements holds at the end of the Stage k.

Case 3: Statement **C** holds at the beginning of Stage k. Let $list_k = \langle u_1, \ldots, u_l \rangle$ at the beginning. If the value of *next* does not change during this stage, then Statement **C** holds after this stage. The value of *next* changes if there is a j such that $|g(\langle 0^n, v_j \rangle)| > k' \log n$. By Observation 4, for every $u \in list_k$, $u < next$. Since v_j is a child of some node u, we have $v_j < u < next$. Thus the right most accepting computation lies to the right of v_j. Since *next* becomes v_j, Statement **C** holds at the end of this stage. □

Claim 14 *The above reduction f is correct, i.e., for every n, $f(0^n)$ is a string in L.*

Proof. Note that $f(0^n)$ is a fixed string in L, if $list_d$ contains an accepting leaf node. In this case the reduction is obviously correct. Suppose none of the leaf nodes in $list_d$ is an accepting node. Then the right most accepting computation does not pass through any node in $list_d$. By the previous claim, at the end of Stage d, $next$ is defined and the right most accepting computation either passes through $next$ or lies to the right of $next$. Thus $(\langle 0^n, next\rangle) \in S$. Since the reduction outputs $g(\langle 0^n, next\rangle)$, and g is a many-one reduction from S to L, the claim follows. $\Box$

Our next claim bounds the size of $list_r$.

Claim 15 *For every $k > 0$, at the beginning of Stage k, the size of $list_k$ is at most $2n^{k'}$.*

Proof. Note that after Step 3, for every node u in $list_k$, $|g(\langle 0^n, u\rangle)| \leq k' \log n$. After Step 4, for every pair of nodes u and v in $list_r$, we have $g(\langle 0^n, u\rangle) \neq g(\langle 0^n, v\rangle)$. Thus, after Stage k, $|list_k| \leq 2n^{k'}$. $\Box$

Next we show that the reduction runs in polynomial time. Assume that, for any node $u \in T$, the computation of $g(\langle 0^n, u\rangle)$ takes n^r steps. Let the depth of the tree T be n^l. We define c to be $r + l$. Note that the constant c depends only on L, and is independent of k'.

Claim 16 *The running time of the reduction is $O(n^{k'+c})$.*

Proof. First we calculate the running time of Stage k. Observe that Step 2 is the most expensive step. By the previous claim, at the beginning of Stage k, $|list_k| \leq 2n^{k'}$. After Step 1, $|list_k| \leq 4n^{k'}$. In Step 2, we compute g for every node in $list_k$. The time required for this computation is $O(n^{k'+r})$. So, Stage k requires $O(n^{k'+r})$ steps. Since there are $d = n^l$ stages, the time taken for the reduction is $O(n^{k'+r+l}) = O(n^{k'+c})$. $\Box$

Claim 17 *For infinitely many n, after Stage d, $list_d$ does not contain accepting nodes.*

Proof. Assume that for all but finitely many n, after Stage d, $list_d$ contains an accepting node. By Claim 16, the time taken by the reduction is $O(n^{k'+c})$. Thus, for all but finitely many n, the procedure computes a polynomial-size list that contains an accepting node. Thus, for all but finitely many n, there is a polynomial-time algorithm that outputs an accepting computation of M on 0^n. This contradicts Hypothesis T. $\Box$

Claim 18 *For infinitely many n, the reduction outputs a string whose length is bigger than $k' \log n$.*

Proof. By the previous claim, for infinitely many n, $list_d$ does not contain an accepting node. So, for infinitely many n, the rightmost accepting computation of M on 0^n does not pass through a node in $list_d$. Thus, by Claim 13, Statement **C** holds after Stage d. Thus, $next$ is defined, and the reduction outputs $g(\langle 0^n, next\rangle)$. Note that the reduction defines $next$ to be v, only if $|g(\langle 0^n, v\rangle)| > k' \log n$. Thus for infinitely many n, the reduction outputs a string whose length is bigger than $k' \log n$. $\Box$

This concludes the proof of Lemma 1. $\Box$

Combing the following lemma with Lemma 1 completes the proof of the theorem.

Lemma 2. *Let L be any* NP*-complete language. If there is a constant $c > 0$ such that for every $k > 0$ there is a reduction f from 0^* to L such that, for infinitely many n, $|f(0^n)| > k \log n$, and f can be computed in time $O(n^{k+c})$, then for every $\epsilon > 0$, L is not $2^{n(1+\epsilon)}$-immune.*

Proof. Given $\epsilon > 0$, pick k such that $k > (c+1)/\epsilon$. There is a reduction f from 0^* to L such that for infinitely many n, $|f(0^n)| > k \log n$. From this it follows that there exist infinitely many $x, |x| = m$, for which there exist $n < 2^{m/k}$ such that $f(0^n) = x$. Consider the algorithm that, on input x, $|x| = m$, behaves as follows:

1. For $i = 1$ to $2^{m/k}$, if $f(0^i) = x$ then accept x and halt.
2. Reject x.

The above algorithm accepts a string x only when it finds that for some i, $f(0^i) = x$. Since f is a many-one reduction from 0^* to L, the above algorithm accepts a subset of L. There exist infinitely many x for which $f(0^i) = x$ for some i, $1 \leq i \leq 2^{m/k}$. So, the above algorithm accepts an infinite set. The most time-consuming step of the algorithm is computation of $f(0^i)$ and this requires i^{k+c} steps. Since i ranges between 1 and $2^{m/k}$, the running time of the algorithm is $2^{m(1+\frac{c+1}{k})}$. Since $\frac{c+1}{k} < \epsilon$, the time taken by the algorithm is $2^{m(1+\epsilon)}$. Thus L is not $2^{n(1+\epsilon)}$-immune. □

Theorem 11 follows from the Lemmas 1 and 2. □