Separating Cook Completeness from Karp-Levin Completeness under a Worst-Case Hardness Hypothesis*

Debasis Mandal[1], A. Pavan[2], and Rajeswari Venugopalan[3]

1 Department of Computer Science
Iowa State University, USA
debasis@iastate.edu
2 Department of Computer Science
Iowa State University, USA
pavan@cs.iastate.edu
3 Department of Computer Science
Iowa State University, USA
advaithi@cs.iastate.edu

Abstract

We show that there is a language that is Turing complete for NP but not many-one complete for NP, under a *worst-case* hardness hypothesis. Our hypothesis asserts the existence of a non-deterministic, double-exponential time machine that runs in time $O(2^{2^{n^c}})$ (for some $c > 1$) accepting Σ^* whose accepting computations cannot be computed by bounded-error, probabilistic machines running in time $O(2^{2^{\beta 2^{n^c}}})$ (for some $\beta > 0$). This is the first result that separates completeness notions for NP under a worst-case hardness hypothesis.

1998 ACM Subject Classification F.1.3

Keywords and phrases Cook reduction, Karp reduction, NP-completeness, Turing completeness, many-one completeness

1 Introduction

The notion of polynomial-time reductions is pervasive in theoretical computer science. In addition to their critical role in defining NP-completeness, polynomial-time reductions play an important role in establishing several results in various areas such as complexity theory, cryptography, learning theory etc. Informally, reductions translate instances of one problem to instances of another problem; a problem A is polynomial-time reducible to a problem B if A can be solved in polynomial-time by making queries to problem B. By varying the manner in which the queries are allowed to make, we obtain a wide spectrum of reductions. At one end of the spectrum is *Cook/Turing reduction* where multiple queries are allowed and the ith query made depends on answers to previous queries. On the other end is the most restrictive reduction, *Karp-Levin/many-one reduction*, where each positive instance of problem A is mapped to a positive instance of problem B, and so are the negative instances. In between are *truth-table/non-adaptive reductions*, and *bounded truth-table reductions*. Interestingly, the seminal paper of Cook [7] used Turing reduction to define NP-completeness, whereas the works of Karp [16] and Levin [19] used many-one reductions.

* Research Supported in part by CCF: 0916797.

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 1–14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Understanding the differences between many-one reductions and Turing reductions is one of the fundamental problems in complexity theory. Compared to many-one reductions, our knowledge about Turing reductions is limited. Extending certain assertions that are known to be true for many-one reductions to the case of Turing reductions yield much sought after separation results in complexity theory. For example, it is known that polynomial time many-one complete sets for EXP are not sparse [24]. Extending this result to the case of Turing reductions implies that EXP does not have polynomial-size circuits. In the context of resource-bounded measure, it is known that "small span theorem" holds for many-one reductions. Establishing a similar result for Turing reductions separates EXP from BPP [15]. In addition, Turing reductions are crucial to define the Polynomial-time hierarchy.

The differences between various types of polynomial-time reductions have been studied in different contexts. Selman [23] showed that if NE ∩ co-NE does not equal E, then there exist languages A and B in NP such that A polynomial-time Turing reduces to B, but does not polynomial-time many-one reduce to B. Aida *et al.* [1] showed a similar result in the average-case world; if P does not equal NP, then there is a distributional problems (A, μ_A) and (B, μ_B) in DistNP such that (A, μ_A) Turing reduces to (B, μ_B) but does not many-one reduce to (B, μ_B). The differences between Turing and truth-table reductions have been studied extensively in the context of random self-reductions and coherence [4, 8, 9, 11]. For example, Feigenbaum *et al.* [8] showed that if nondeterministic triple exponential time is not in bounded-error, probabilistic triple exponential time, there exists a function in NP that is Turing random self-reducible, but not truth-table random-self reducible.

In this paper we study the differences between many-one and Turing reductions in the context of completeness. Even though, it is standard to define completeness using many-one reductions, one can also define completeness using Turing reductions. A language L is *Turing complete* for a class $\mathcal{C}$ if L is in class $\mathcal{C}$ and every language in $\mathcal{C}$ Turing reduces to L. To capture the intuition that if a complete problem for a class $\mathcal{C}$ is "easy", then the entire class is easy, Turing reductions are arguably more appropriate to define completeness. However, all known natural languages turn out to be complete under many-one reductions. This raises the following question: For a complexity class $\mathcal{C}$, is there a Turing complete language that is not many-one complete? This question was first posed by Ladner, Lynch, and Selman [18].

This question has been completely resolved for the complexity classes EXP and NEXP. Works of Ko and Moore [17] and Watanabe [27] showed that for EXP, almost all completeness notions are mutually different. Similar separation results are obtained for NEXP [5]. See survey articles [6, 14] for more details on these results.

For the case of NP, the progress has been very slow. The first result that achieves a separation between Turing and many-one completeness in NP, under a reasonable hypothesis, is due to Lutz and Mayordomo [20]. They showed that if NP does not have P-measure 0 (known as *measure hypothesis*), then Turing completeness for NP is different from many-one completeness. Ambos-Spies and Bentzien [2] achieved a finer separation under a weaker hypothesis known as *genericity hypothesis*. Subsequently, Turing and many-one completeness notions are shown to be different under even weaker hypotheses known as NP *machine hypothesis*, *bi-immunity hypothesis*, and *partial bi-immunity hypothesis* [13, 21, 22].

All of the above mentioned hypotheses are known as *almost everywhere hardness hypotheses*. Informally, these hypotheses assert that there exists a language in NP such that every algorithm that decides L must take more than subexponential time on *all but finitely many inputs*. Even though we believe that NP is subexponentially hard, we do not have any candidate languages in NP that are almost everywhere hard. All natural problems have an infinite set of instances that can be decided in polynomial time. Thus these hypotheses are considered "strong hypotheses". It has been open whether a separation can be achieved using a *worst-case hardness* hypothesis (such as P $\neq$ NP, or NE $\neq$ E). The only partial result in this direction is due to Gu, Hitchcock, and Pavan [10] who showed that if there exist one-way permutations and there exists a language in NEEE $\cap$ co-NEEE that can not be solved in deterministic triple exponential time with logarithmic advice, then Turing completeness for NP differs from many-one completeness. Even though the latter hypothesis is a worst-case hardness hypothesis, the former is a average-case hardness hypothesis.

In this paper, we separate Turing completeness for NP from many-one completeness using a worst-case hardness hypothesis. This is the first result of this nature. Below is an informal statement of our result. Please see Section 3 for a more formal statement.

▶ **Main Theorem.** *There exist an* NEEXP *machine N accepting Σ^* and running in time $t(n)$ and $\delta > 0$ such that no zero-error, probabilistic machine Z running in time $2^{t(n)^\delta}$ can compute accepting computation of N with non-trivial probability. Here we require that $t(n)$ is $2^{2^{n^c}}$ for some constant $c > 1$.*

The rest of the paper is organized as follows. Section 2 is the preliminaries section. In Section 3, we formally state our worst-case hardness hypothesis, and provide a proof of the separation theorem. Section 4 relates the hypothesis used in this paper to a few other hypotheses studied in the context of separating completeness notions.

2 Preliminaries

We use standard notions and definitions in complexity theory [3]. All languages are defined over the the binary alphabet $\Sigma = \{0, 1\}$, Σ^n denotes the set of all binary strings of length n. We use $|x|$ to denote the length of a string x. Non-deterministic double-exponential time is defined by NEEXP $= \bigcup_{c>1} \text{NTIME}(2^{2^{n^c}})$ and co-NEEXP is its complement class. We say that a non-nedeterministic machine is a NEEXP machine, if its runtime is bounded by $2^{2^{n^c}}$ for some $c > 1$. A language L is in ZPTIME($t(n)$), if there is a probabilistic machine Z running in time $O(t(n))$ such that for every x, $\Pr[Z(x) = L(x)]$ is atleast $1/4$, and the probability that Z outputs an incorrect answer is zero. The machine Z may output $\perp$ with certain probability (at most $3/4$).

▶ **Definition 1.** Suppose N is a nondeterministic machine accepting a language S. We say that a *$t(n)$-time bounded, zero-error, probabilistic machine computes accepting computations of N* if there exists a probabilistic machine Z such that

- For every $x \in S$, for every choice of random bits $Z(x)$ either outputs a string from Σ^* or outputs the special symbol $\perp$.
- for every $x \in S$, $\Pr[Z(x)$ is an accepting computation of $N(x)] > 1/4$, and
- for every $x \in S$, $\Pr[Z(x) \neq \perp$ and is not an accepting computation of $N(x)] = 0$.

Our proof uses the notion of P-selective sets introduced by Selman [23].

▶ **Definition 2.** A set $S \subseteq \Sigma^*$ is P-*selective* if there is a polynomial time computable function $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that for all strings $x, y \in \Sigma^*$, (1) $f(x, y) \in \{x, y\}$, and (2) if either of x and y is in S, then $f(x, y)$ is in S. The function f is called the P-*selector* of S.

The well-known example of P-selective sets are the *left-cut* sets $L(r) = \{x \mid x < r\}$, where r is an infinite binary sequence, and $<$ is the dictionary order with $0 < 1$. The following lemma is due to Toda [25].

▶ **Lemma 3.** *For every* P*-selective set L, there is a polynomial time algorithm that given any finite set of strings Q as input, outputs a sequence $x_1, \cdots, x_m$ such that $\{x_1, \cdots, x_m\} = Q$, such that for some integer p, $0 \leq p \leq m$, $Q \cap L = \{x_i \mid i \leq p\}$ and $Q \cap \bar{L} = \{x_i \mid i > p\}$.*

Consider two languages A and B. A is *polynomial time Turing reducible* to B, denoted by $A \leq_T^{\mathrm{P}} B$, if there is a polynomial time oracle Turing machine M such that $A = L(M^B)$. Note that M can make at most polynomially many queries to B and they can be *adaptive*. The language A is *polynomial-time truth-table reducible* to B, denoted by $A \leq_{tt}^{\mathrm{P}} B$, if there is a pair of polynomial time computable functions $\langle f, g \rangle$ such that for every $x \in \Sigma^*$, (1) $f(x)$ is query set $Q = \{q_1, q_2, \cdots, q_k\}$ and (2) $x \in A \iff g(x, B(q_1), B(q_2), \cdots, B(q_k)) = 1$. We call f the *query generator* and g the *truth-table evaluator*. Given a polynomial time reducibility $\leq_r^{\mathrm{P}}$, a set B is $\leq_r^{\mathrm{P}}$*-complete* for NP if B is in NP and for every set $A \in$ NP, A is $\leq_r^{\mathrm{P}}$ reducible to B.

Notation. Let $\tau : \mathbb{N} \to \mathbb{N}$ be a function defined as $\tau(n) = 2^{2^n}$. The functions of the form $2^{2^{f(n)}}$, that are used in many places throughout this paper, are not visually appealing; from now we represent such functions as $\tau(f(n))$. Then $\tau(\delta f(n))$ represents $2^{2^{\delta f(n)}}$. We use $\tau^\epsilon(n)$ to denote $(\tau(n))^\epsilon$. Further, $\log^c n$ represents $(\log n)^c$.

3 Separation Theorem

In this section we prove the main result of this paper. First, we formally state our hypothesis.

Hypothesis W. *There exist a constant $\delta > 0$ and an* NEEXP *machine N_1 accepting Σ^* that runs in time $t(n)$ such that no $2^{t(n)^\delta}$-time bounded, zero-error, probabilistic machine can compute the accepting computations of N_1. Here $t(n) = 2^{2^{n^c}}$ for some constant $c > 1$.*

▶ **Theorem 4.** *If Hypothesis* W *holds, then there is a Turing complete language for* NP *that is not truth-table complete for* NP*.*

Before we provide a formal proof, we first describe proof outline. Our proof proceeds in four steps. Note that Hypothesis W is a "worst-case hardness hypothesis". This means that for every probabilistic, $2^{t(n)^\delta}$-time bounded, machine Z_1 there exists *infinitely many* inputs x such that the probability that $Z_1(x)$ computes an accepting computation of $N_1(x)$ is very small. This is equivalent to the following: there exist *infinitely many* input lengths n for which there exists *at least one string x of length n* so that the probability that $Z_1(x)$ is an accepting computation of $N_1(x)$ is very small. In the first step (Section 3.1), we amplify the hardness of N_1 and obtain an NEEXP machine N_2 with the following property: For every $2^{t(n)^\delta}$-time bounded, probabilistic machine Z_2, there exist *infinitely many input lengths n* at which *for every string x of length n* the probability that $Z_2(x)$ is an accepting computation of $N_2(x)$ is small.

In the second step (Section 3.2), we first define a padding function $pad : \Sigma^* \to \mathbb{N}$. Via standard padding arguments we obtain an NP-machine N running in time $p(n)$ that accepts a tally set $T = \{0^{pad(x)} \mid x \in \Sigma^*\}$. For $\ell \geq 0$, let $T_\ell = \{0^{pad(x)} \mid x \in \Sigma^\ell\}$. The NP-machine N has the following hardness property: For every $f(n)$-time bounded, probabilistic machine Z (for an appropriate choice of f) there exist infinitely many integers ℓ such that Z fails to compute accepting computations on *every string* from T_ℓ.

Using the NP-machine N, we define the Turing complete language L in step three (Section 3.3). The language L is formed by taking disjoint union of two NP languages L_1 and L_2. The language L_1 consists of tuple of the form $\langle x, a \rangle$ so that $x \in C$ (for some NP-complete language C), and a is an accepting computation of $N(0^n)$ (for some n that depends on x).

In L_2, we encode accepting computations of N using a P-selective set. It follows that C can be Turing reduced to L by first obtaining an accepting computation of N (by making queries to L_2) and then by making one query to L_1. The idea of forming L_1 is borrowed from [21], and encoding accepting computations of an NP-machine as a P-selective sets is well known. For example see [11].

Finally, in step four (Section 3.4), we show that if L is truth-table complete, then there is a probabilistic machine Z such that for every ℓ there exists atleast one string in T_ℓ so that Z computes an accepting computation of N on that string with high probability. Using this, we in turn show that there exists a probabilistic machine Z_2 so that for every input length ℓ, there exists atleast one string $x \in \Sigma^\ell$ such that $Z_2(x)$ outputs an accepting computation of the NEEXP machine $N_2(x)$. This will be a contradiction. The most technical part of the proof lies in this step. We now give proof details.

3.1 Hardness Amplification

The first step amplifies the hardness of the NEEXP machine N_1 to obtain a new NEEXP machine N_2.

▶ **Lemma 5.** *Suppose that the hypothesis* W *holds. Then there exist an* NEEXP *machine* N_2 *accepting* Σ^* *and running in time* $O(2^n\tau(n^c))$ *and a constant* $\beta < \delta$ *such that for every probabilistic machine* Z_2 *that runs in time* $\tau(\beta 2^{n^c})$*, there exist* infinitely many input lengths $n > 0$ *such that for* every $x \in \Sigma^n$*,*

$$\Pr[Z_2(x) = \textit{ an accepting computation of } N_2(x)] \leq 1/4.$$

Proof. Let N_1 be the nondeterministic machine from Hypothesis W whose running time is bounded by $O(t(n))$, where $t(n) = \tau(n^c)$ (for some $c > 1$). Length of every accepting computation of $N_1(x)$ is bounded by $O(t(|x|))$. Consider a machine N_2 that behaves as follows: On an input x of length n, it runs $N_1(y)$ on every string y of length n (in a sequential manner). The running time of N_2 is $O(2^n \times t(n))$. Since N_1 accepts Σ^*, the machine N_2 also accepts Σ^*. We claim that N_2 has the required property.

Suppose not. Then there is a probabilistic machine Z_2 that runs in time $O(\tau(\beta 2^{n^c}))$ (for some $\beta < \delta$) such that for for all but finitely many n, there exists a string $y_n \in \Sigma^n$ such that

$$\Pr[Z_2(y_n) = \text{ an accepting computation of } N_2(y_n)] > 1/4.$$

By the definition of N_2, the accepting computation of $N_2(x)$ encodes the accepting computation of $N_1(y)$ for every y whose length is same as the length of x. Consider a machine Z_1 that on any input x of length n behaves as follows:

> It runs $Z_2(y)$ on every y of length n. It verifies that the output of $Z_2(y)$ is an accepting computation of $N_2(y)$, and if the verification succeeds, then it extracts the accepting computation of $N_1(x)$ and outputs it. If $Z_2(y)$ does not output an accepting computation of $N_2(y)$, then Z_1 outputs $\perp$.

Let x be any input of length n. By our assumption, there exists a $y_n \in \Sigma^n$ such that $Z_2(y_n)$ outputs an accepting computation of $N_2(y_n)$ with probability at least 1/4. The above machine clearly runs $Z_2(y_n)$ on input x. Since an accepting computation of $N_1(x)$ can be retrieved from an accepting computation of $N_2(y_n)$, the above machine outputs an accepting computation of $N_1(x)$. Thus for all but finitely many n, for every $x \in \Sigma^n$, Z_1 outputs an accepting computation of $N_1(x)$ with probability at least 1/4. The running time of Z_1 is clearly $O(2^n \times \tau(\beta 2^{n^c}))$, which is less than $\tau(\delta 2^{n^c})$ (as $\beta < \delta$). This contradicts Hypothesis W. ◀

3.2 Defining an NP machine

In this section, we define an NP machine N from the above NEEXP machine N_2. Fix $\epsilon < \beta$. Consider the following padding function $pad : \Sigma^* \to \mathbb{N}$, defined by

$$pad(x) = \lfloor \tau^\epsilon (\log^c r_x) \rfloor,$$

where r_x is the rank of string x in the standard lexicographic order of Σ^*, so that $2^\ell - 1 \leq r_x \leq 2^{\ell+1} - 2$, for every $x \in \Sigma^\ell$. Note that *pad* is 1-1 and so $pad^{-1}(n)$ (if exists) is well defined. To keep the calculation simple, we drop the floors henceforth. Now we define the following tally language based on the padding function:

$$T = \left\{ 0^{pad(x)} \mid x \in \Sigma^* \right\},$$

Our NP machine N that accepts a tally language behaves as follows: On input 0^m, it computes $x = pad^{-1}(m)$. Upon finding such x, it runs $N_2(x)$. If no such x is found, then N rejects. Note that $|x| < (\log \log m^{2/\epsilon})^{1/c}$. So running time of N is bounded by $m^{3/\epsilon}$. Thus N is an NP machine. Note that N accepts the tally language T.

3.3 Turing-complete language

At this point, we are ready to define the language L in NP that we prove to be Turing complete, but not truth-table complete for NP.

Let L_T be the range of the padding function *pad*.

$$L_T = \{\tau^\epsilon (\log^c i) \mid i \in \mathbb{N}\}.$$

By definition, N accepts only those tally strings whose length is in the set L_T. We use n_i to denote $pad(i)$. Given a length $n \in L_T$, define a_n to be the lexicographically maximum accepting computation of $N(0^n)$. Let a be the infinite binary string $a_{n_1} a_{n_2} a_{n_3} \cdots$ where $n_i \in L_T$ and $n_1 < n_2 < n_3 < \cdots$. Let $|a_n|$ denotes the length of the accepting computation a_n. Let SAT$'$ consist of the SAT formulas with lengths only in L_T, *i.e.*,

$$\mathrm{SAT}' = \mathrm{SAT} \cap \{x \in \Sigma^* \mid |x| \in L_T\}.$$

Since there exists a polynomial p such that $n_{i+1} \leq p(n_i)$, it can be shown via padding that SAT many-one reduces to SAT$'$ and thus SAT$'$ is NP-complete.

We define L_1 and L_2 as follows.

$$L_1 = \left\{ \langle \phi, u \rangle \mid |\phi| = n, \ u \text{ is an accepting computation of } N \text{ on } 0^n, \ \phi \in \mathrm{SAT}' \right\}$$

$$L_2 = L(a) = \{z \mid z < a\},$$

where $<$ is the dictionary order with $0 < 1$. Then our Turing-complete language L is the disjoint union of L_1 and L_2, *i.e.*,

$$L = L_1 \oplus L_2 = 0L_1 \cup 1L_2.$$

Note that both L_1 and L_2 are in NP, and so is L.

▶ **Lemma 6.** *L is $\leq_T^{\mathrm{P}}$-complete for* NP.

Proof. Reduce SAT$'$ to L: On input ϕ of length n, make adaptive queries to L_2 to find a_n. Accept ϕ if and only if $\langle \phi, a_n \rangle \in L_1$. ◀

3.4 L is not truth-table complete

In this section, we show that L is not truth-table complete for NP. Before we proceed with the proof, we provide the intuition behind the proof. Suppose that L is truth-table complete. We achieve a contradiction by exhibiting a procedure to compute accepting computations of NEEXP machine N_2. Since the NP-machine N is padded version of N_2, it suffices to compute the accepting computations of N. We partition T into sets $T_1, T_2, \cdots$, where $T_\ell = \{0^{pad(x)} \mid x \in \Sigma^\ell\}$. Clearly, $|T_\ell| = 2^\ell$ and $T = \bigcup_\ell T_\ell$. Note that an accepting computation of $N_2(x)$ can be computed by computing an accepting computation of $N(0^{pad(x)})$, and if $|x| = \ell$, then $0^{pad(x)} \in T_\ell$.

Recall that N_2 has the following property: For every probabilistic machine Z_2 that attempts to compute its accepting computations, there exist infinitely many input lengths ℓ and Z_2 fails on every string at those lengths. Informally, this translates to the following hardness property of N: For every probabilistic machine Z that attempts to compute accepting computations of N, there exist infinitely many integers ℓ such that Z fails on every string from T_ℓ. Thus to achieve a contradiction, it suffices to exhibit a probabilistic procedure Z such that for all but finitely many ℓ, Z outputs an accepting computation of $N(0^n)$ for some $0^n \in T_\ell$, with non-negligible probability. We will now (informally) describe how to compute accepting computations of N.

For the sake of simplicity, let us first assume that the NP machine N has exactly one accepting computation on every input from T. The first task is to define a set S that encodes the accepting computations of the machine N. One way to define S as

$$S = \{\langle 0^n, i\rangle \mid i\text{th bit of accepting computation of } N(0^n) \text{ is } 1\}.$$

Since we assumed that N has exactly one accepting computation, deciding S is equivalent to computing accepting computations of N. Since S is in NP, there is a truth-table reduction from S to L. We make another simplifying assumption that all queries are made to L_1 part of L. Consider an input $\langle 0^n, i\rangle$ where $0^n \in T_\ell$ (for some $\ell > 0$). All the queries produced on this input are of the form $\langle \phi, u\rangle$. It is easy to check if u is an accepting computation of $N(0^m)$ for some m. If u is not an accepting computation, then $\langle \phi, u\rangle$ does not belong to L, and thus it is easy to decide the membership of $\langle 0^n, i\rangle$ in S. Suppose that u is an accepting computation of $N(0^m)$ for some m. Then there are two cases. First case is the "short query" case, where m is much smaller than n. In this case $\langle \phi, u\rangle$ is in L_1 only when $|\phi|$ equals m and $\phi \in \mathrm{SAT}'$. Since $m << n$, we can decide whether $\phi \in \mathrm{SAT}'$ using a brute force algorithm in time $O(2^m)$, this in turn enables us to decide the membership of $\langle 0^n, i\rangle$ in S. Thus if all the queries are small, we can decide the memberships of $\langle 0^n, i\rangle$ (for all i), and thus can compute accepting computation of $N(0^n)$. The second case is the "large query" case: Suppose that for some query, m is not much smaller than n. In this case, we are in the following scenario: The reduction outputs accepting computation of $N(0^m)$ and m is somewhat large. In this case, we argue that for an appropriate choice of n, 0^m also lies in T_ℓ. This will enable us to design a procedure that outputs accepting computation of some string from T_ℓ. This is the gist of the proof.

The above argument assumed that N has exactly one accepting computation, which is not true in general. We get around this problem by applying Valiant-Vazirani lemma [26] to isolate one accepting computation. Thus our language S will involve the use of isolation lemma. It is also very much possible that the reduction makes queries to L_2 also. Recall that L_2 is a P-selective set and it is known that if an NP-language A reduces to a P-selective set, then A must be "easy" [25, 23]. We use this in combination with the above mentioned approach. A technically involved part is to define the correct notion of "small" and "large" queries. There is a fine interplay among the choice of pad function, notion of small query, and the runtime of probabilistic machine that computes the accepting computations of N. We now provide a formal proof.

▶ **Lemma 7.** *L is not $\leq_{tt}^{\mathrm{P}}$-complete for* NP.

Proof. For the sake of contradiction, assume that L is truth-table complete for NP. Consider the following set S.

$$S = \{\langle 0^n, k, r_1, r_2, \ldots, r_k, i\rangle \mid n \in L_T, 1 \leq k \leq |a_n|, r_i \in \Sigma^{|a_n|}, \text{ there is a } u \text{ such that}$$
$$u \text{ is an accepting computation of } N(0^n), u{\cdot}r_1 = u{\cdot}r_2 = \cdots = u{\cdot}r_k = 0, \text{ and the } i\text{th bit of } u = 1\},$$

where $u \cdot r_i$ denotes the inner product over GF[2].

It is easy to see that S is in NP. Since L is $\leq_{tt}^{\mathrm{P}}$-complete for NP, S is $\leq_{tt}^{\mathrm{P}}$ reducible to L via polynomial time computable functions $\langle g, h\rangle$, where g is the query generator and h is the truth-table evaluator. Since g is polynomial-time computable, there exists a constant $b > 0$ such that every query generated by it is of length at most n^b.

At this point, our goal is to compute an accepting computation of N. We start with the following algorithm $\mathcal{A}$ that classifies all the queries of the query generator into two sets, "Large Query" and "Small Query".

1. Input 0^n, where $n = \tau^\epsilon(\log^c i)$ for some $i \in \mathbb{N}$. Clearly, $n \in L_T$.
2. For $1 \leq j \leq n^2$ repeat the following:
   - Pick k^j uniformly at random from $\{1, \cdots, |a_n|\}$.
   - Pick each of $r_1^j, r_2^j, \ldots, r_{k_j}^j$ uniformly at random from $\Sigma^{|a_n|}$.
3. Let Q^j be the set of queries generated by g on inputs $\langle 0^n, k^j, r_1^j, \cdots, r_{k_j}^j, i\rangle$, $1 \leq i \leq |a_n|$. Compute Q^j for $1 \leq j \leq n^2$ and set $Q = \bigcup_j Q^j$. Note that the length of each query is bounded by n^b.
4. Partition Q into two sets Q_1 and Q_2 such that Q_1 is the set of all queries to L_1 and Q_2 is the set of all queries to L_2.
5. If Q_1 contains a query $\langle \phi, u_t\rangle$ for some t, where u_t is an accepting computation of $N(0^t)$ and
$$t > \tau^\epsilon(((\log\log n^{b/\epsilon})^{1/c} - 1)^c),$$
then print u_t, output "Large Query", and halt.
6. Otherwise, output "Small Query" and halt.

It is clear that the algorithm $\mathcal{A}$ runs in time polynomial in n.

Before we give our probabilistic algorithm to compute the accepting computations of N, we bound the probabilities of certain events of interest. T is partitioned into sets $T_1, T_2, \cdots$ each of cardinality 2^ℓ, where

$$T_\ell = \left\{0^{\tau^\epsilon(\log^c r_x)} \mid x \in \Sigma^\ell\right\}.$$

Fix $\ell > 0$. For a fixed $0^n \in T_\ell$ and j, $1 \leq j \leq n^2$, let $\mathcal{E}_{n,j}$ denote the following event:

There exists *exactly one* u such that
- u is an accepting computation on $N(0^n)$,
- $u \cdot r_1^j = u \cdot r_2^j = \cdots = u \cdot r_{k_j}^j = 0$.

By Valiant-Vazirani, we have that $\Pr[\mathcal{E}_{n,j}] \geq \frac{1}{n^2}$. Let $\mathcal{E}_n$ denote the event that for some j, $1 \leq j \leq n^2$, $\mathcal{E}_{n,j}$ occurs. The probability of $\mathcal{E}_n$ is at least $1 - \frac{1}{2^{n^2}}$. Finally, let $\mathcal{E}_\ell$ denote the event that for every $0^n \in T_\ell$, the event $\mathcal{E}_n$ occurs. Again, we have that $\Pr[\mathcal{E}_\ell] \geq 1 - \frac{1}{2^\ell}$.

Thus for every ℓ, the probability that the event $\mathcal{E}_\ell$ occurs is very high. Fix an ℓ. From now on, we assume that the event $\mathcal{E}_\ell$ has occurred.

Now our goal is to arrive at the machine that computes an accepting computation of atleast one string from T_ℓ. For this we will analyze the behavior of the above algorithm on a specific string $0^{V_\ell} \in T_\ell$, where

$$V_\ell = \tau^{\epsilon/b}(\log^c(2^{\ell+1} - 2)).$$

We stress that this *unique* string 0^{V_ℓ} depends only on the length ℓ. When we run algorithm $\mathcal{A}$ on 0^{V_ℓ}, either it outputs “Large Query” on it, or it outputs “Small Query”.

▶ **Lemma 8** (Key Lemma). *One of the following holds.*

1. *If $\mathcal{A}$ outputs “Small Query” on 0^{V_ℓ}, then there is an algorithm $\mathcal{B}_1$ that on input 0^{V_ℓ} runs in time polynomial in $\tau(\epsilon 2^{((\log\log {V_\ell}^{b/\epsilon})^{1/c}-1)^c})$, and correctly outputs an accepting computation of $N(0^{V_\ell})$.*
2. *If $\mathcal{A}$ outputs “Large Query” on 0^{V_ℓ}, there exist an algorithm $\mathcal{B}_2$ such that for every string in T_ℓ it runs in time polynomial in V_ℓ, and there exists a $0^t \in T_\ell$ for which $\mathcal{B}_2(0^t)$ outputs an accepting computation of $N(0^t)$.*

We defer the proof of this lemma to appendix and complete the proof of main theorem by describing a probabilistic machine that computes accepting computation of the NEEXP machine N_2.

Computing accepting computations of N_2

Consider the probabilistic machine Z_2 that does the following on input $x \in \Sigma^\ell$:

1. Compute V_ℓ. Run $\mathcal{A}$ on 0^{V_ℓ}.
2. If $\mathcal{A}(0^{V_\ell})$ outputs “Small Query”,
   - Verify if $x = pad^{-1}(V_\ell)$. If it is, then run $\mathcal{B}_1$ on 0^{V_ℓ} and if it outputs an accepting computation of $N(0^{V_\ell})$, then output that accepting computation. This is also the accepting computation of $N_2(x)$.
3. If $\mathcal{A}(0^{V_\ell})$ outputs “Large Query”, do the following:
   - For every string 0^i in T_ℓ, run the algorithm $\mathcal{B}_2$ on it. If it outputs the accepting computation of $N(0^t)$ for some 0^t, then verify if $x = pad^{-1}(0^t)$. If it is, then output that accepting computation. This is also the accepting computation of $N_2(x)$.

We analyze the behavior of Z_2 under the assumption that the event $\mathcal{E}_\ell$ happens. Recall that this happens with very high probability. If $\mathcal{A}(0^{V_\ell})$ outputs “Small Query”, then by part (1) of Lemma 8, $\mathcal{B}_1$ outputs an accepting computation of $N(0^{V_\ell})$. Note that every accepting computation of $N(0^{V_\ell})$ is an accepting computation of $N_2(0^{pad^{-1}(V_\ell)})$. Since $pad^{-1}(V_\ell)$ is of length ℓ, there exists a string $x \in \Sigma^\ell$, on which Z_2 outputs an accepting computation of $N_2(x)$. Now consider the case where $\mathcal{A}(0^{V_\ell})$ outputs “Large Query”, then by part (2) of Lemma 8, there exists a $0^t \in T_\ell$ such that $\mathcal{B}_2(0^t)$ outputs an accepting computation of $N(0^t)$. Thus Z_2 will find that 0^t through iteration. Similarly, $pad^{-1}(0^t) \in T_\ell$ is of length ℓ, thus there exists a x in Σ^ℓ on which Z_2 outputs an accepting computation of $N_2(x)$. Thus Z_2 always outputs an accepting computation of atleast one string x from Σ^ℓ.

We will now bound the runtime of Z_2. This is bounded by runtime of $A(0^{V_\ell})$, plus the runtime of $\mathcal{B}_1(0^{V_\ell})$, and the time taken in step 3 of the above algorithm. By part (1) of Lemma 8, the runtime of $\mathcal{B}_1(0^{V_\ell})$ is $\tau^d(\epsilon 2^{((\log\log {V_\ell}^{b/\epsilon})^{1/c}-1)^c})$ for some constant $d > 0$, which is bounded by

$$\tau^d(\epsilon 2^{((\log\log {V_\ell}^{b/\epsilon})^{1/c}-1)^c}) = \tau^d(\epsilon 2^{((\log^c(2^{\ell+1}-2)^{1/c}-1)^c}) = \tau^d(\epsilon 2^{((\log(2^\ell-1))^c}) < \tau^d(\epsilon 2^{\ell^c}).$$

Let p be a constant such that $\mathcal{A}(0^{V_\ell})$ runs in time V_ℓ^p and $\mathcal{B}_2(0^i)$, $0^i \in T_\ell$, runs in time V_ℓ^p. Step 3 runs $\mathcal{B}_2$ on every string from T_ℓ, and there are 2^ℓ strings in T_ℓ. Thus the combined runtime of $\mathcal{A}(0^{V_\ell})$ in step 1 and step 3 is bounded by

$$2^{\ell+1}V_\ell^p = 2^{\ell+1}\tau^{p\epsilon/b}(\log^c(2^{\ell+1}-2)) \leq 2^{\ell+1}\tau^{p\epsilon/b}((\ell+1)^c) \leq \tau^q((\ell+2)^c)$$

for some constant $q > p$. Thus the total running time of Z_2 is bounded by $\tau(\beta 2^{\ell^c})$, as $\beta > \epsilon$.

Thus for all but finitely many ℓ, the machine Z_2 computes an accepting computation of $N_2(x)$ for atleast one string x from Σ^ℓ with non-trivial probability. This contradicts the hardness of NEEXP machine N_2 in Lemma 5. This completes the proof of Lemma 7. ◀

This also completes the proof the main theorem.

4 Power of the hypothesis

In this section, we show some results that explain the power of Hypothesis W and also compare it to some of the previously studied hypotheses that are used to separate NP-completeness notions. All proofs in this section appear in the appendix.

Even though Hypothesis W talks about the difficulty of computing accepting computations of NEEXP machines, our first result states that it can be related to the hardness of the complexity class NEEXP $\cap$ co-NEEXP.

Hypothesis 2. There exist $c > 1$ and $\delta < 1$ such that NTIME$(t(n)) \cap$ co-NTIME$(t(n)) \not\subseteq$ ZPTIME$(2^{t(n)^\delta})$, for $t(n) = 2^{2^{n^c}}$.

Now we show that our hypothesis follows from this worst-case separation hypothesis.

▶ **Proposition 1.** *Hypothesis 2 implies Hypothesis W.*

Pavan and Selman [21] showed that the NP-completeness notions differ under the following hypothesis.

Hypothesis 3. (NP-machine Hypothesis) There exist an NP machine N accepting 0^* and $\beta > 0$ for every 2^{n^β}-time bounded deterministic algorithm M, $M(0^n)$ does not output an accepting computation of $N(0^n)$ *for all but finitely many n.*

Note that the hypothesis requires that every machine that attempts to compute accepting computations of N must fail on *all but finitely many inputs.* This type of hardness hypothesis is called “almost every hardness hypothesis”. In contrast, Hypothesis W requires that every machine that attempts to compute accepting computations of the NEEXP machine must fail on only *infinitely many strings.*

Ideally, we would like to show that NP-machine hypothesis implies Hypothesis W. However, NP-machine hypothesis concerns with hardness against deterministic algorithms whereas Hypothesis W concerns with hardness against probabilistic algorithms. If we assume well-accepted derandomization hypotheses, we can show Hypothesis W is weaker than the NP-machine hypothesis.

▶ **Proposition 2.** *Suppose that* $\mathrm{ZPP} = \mathrm{P}$*. If* NP*-machine hypothesis holds, then Hypothesis W holds.*

Lutz and Mayordomo [20] achieved the separation of NP-completeness notions under the *Measure Hypothesis*. Hitchcock and Pavan [12] showed that *Measure hypothesis* implies the above NP-machine hypothesis. Thus we have the following.

▶ **Proposition 3.** *Suppose that* $\mathrm{ZPP} = \mathrm{P}$*. Measure hypothesis implies Hypothesis W.*

Pavan and Selman [22] showed that if NP-contains 2^{n^ϵ}-*bi-immune* sets, then completeness in NP differ. Informally, the hypothesis means the following: There is a language L in NP such that every 2^{n^ϵ}-time bounded algorithm that attempts to decide L must fail on *all but finitely many strings*. Thus this hypothesis concerns with almost-everywhere hardness, whereas Hypothesis W concerns with worst-case hardness. We are not able to show that the bi-immunity hypothesis implies Hypothesis W (even under the assumption $\mathrm{ZPP} = \mathrm{P}$). However, we note that if $\mathrm{NP} \cap \text{co-NP}$ has bi-immune sets, then Hypothesis W follows. Pavan and Selman [21] showed that if $\mathrm{NP} \cap \text{co-NP}$ has a $\mathrm{DTIME}(2^{n^\epsilon})$-*bi-immune* set, then NP-machine hypothesis follows.

▶ **Proposition 4.** *Suppose that* $\mathrm{ZPP} = \mathrm{P}$*. If* $\mathrm{NP} \cap \text{co-NP}$ *has a* $\mathrm{DTIME}(2^{n^\epsilon})$*-bi-immune set, then Hypothesis W holds.*

5 Conclusions

This paper, for the first time, shows that Turing completeness for NP can be separated from many-one completeness under a worst-case hardness hypothesis. Our hypothesis concerns with hardness of nondeterministic, double exponential time. An obvious question is to further weaken the hypothesis. Can we achieve the separation under the assumption that there exists a language in NE that can not be solved in deterministic/probabilistic time $O(2^{\delta 2^n})$?

References

1 S. Aida, R. Schuler, T. Tsukiji, and O. Watanabe. On the difference between polynomial-time many-one and truth-table reducibilities on distributional problems. In *18th International Symposium on Theoretical Aspects of Computer Science*, 2001.

2 K. Ambos-Spies and L. Bentzien. Separating NP-completeness under strong hypotheses. *Journal of Computer and System Sciences*, 61(3):335–361, 2000.

3 S. Arora and B. Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, 2009.

4 L. Babai and S. Laplante. Stronger separations ofor random-self-reducibility, rounds, and advice. In *14th IEEE Conference on Computational Complexity*, pages 98–104, 1999.

5 H. Buhrman, S. Homer, and L. Torenvliet. Completeness notions for nondeterministic complexity classes. *Mathematical Systems Theory*, 24:179–200, 1991.

6 H. Buhrman and L. Torenvliet. On the structure of complete sets. In *9th IEEE Annual Conference on Structure in Complexity Theory*, pages 118–133, 1994.

7 S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd ACM Symposium on Theory of Computing*, pages 151–158, 1971.

8 J. Feigenbaum, L. Fortnow, C. Lund, and D. Spielman. The power of adaptiveness and additional queries in random-self-reductions. In *Proc. 7th Annual Conference on Structure in Complexity Theory*, pages 338–346, 1992.

9 J. Feigenbaun, L. Fortnow, S. Laplante, and A. Naik. On coherence, random-self-reducibility, and self-correction. In *Proceedings of the Eleventh Annual IEEE Conference on Computational Complexity*, pages 224–232, 1996.

10 X. Gu, J. Hitchcock, and A. Pavan. Collapsing and separating completeness notions under average-case and worst-case hypotheses. *Theory of Computing Systems*, 51(2):248–265, 2011.

11 E. Hemaspaandra, A. Naik, M. Ogiwara, and A. Selman. P-selective sets and reducing search to decision vs. self-reducibility. *Journal of Computer and System Sciences*, 53(2):194–209, 1996.

12 J. Hitchcock and A. Pavan. Hardness hypotheses, derandomization, and circuit complexity. *Computational Complexity*, 17(1):119–146, 2008.

13 J. Hitchcock, A. Pavan, and N. V. Vinodchandran. Partial bi-immunity, scaled dimension and np-completeness. *Theory of Computing Systems*, 42(2):131–142, 2008.

14 S. Homer. Structural properties of complete problems for exponential time. In L. Hemaspaandra and A. Selman, editors, *Complexity Theory Retrospective II*, pages 135–153. Springer-Verlag, 1997.

15 D. W. Juedes and J. H. Lutz. The complexity and distribution of hard problems. *SIAM Joutnal on Computing*, 24:279–295, 1995.

16 R. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–104. Plenum Press, New York, 1972.

17 K. Ko and D. Moore. Completeness, approximation and density. *SIAM Journal on Computing*, 10(4):787–796, Nov. 1981.

18 R. Ladner, N. Lynch, and A. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1:103–123, 1975.

19 L. Levin. Universal sorting problems. *Problems of Information Transmission*, 9:265–266, 1973. English translation of original in *Problemy Peredaci Informacii.*

20 J. H. Lutz and E. Mayordomo. Cook versus Karp-Levin: Separating completeness notions if NP is not small. *Theoretical Computer Science*, 164:141–163, 1996.

21 A. Pavan and A. Selman. Separation of NP-completeness notions. *SIAM Journal on Computing*, 31(3):906–918, 2002.

22 A. Pavan and A. Selman. Bi-immunity separates strong NP-completeness notions. *Information and Computation*, 188:116–126, 2004.

23 A. Selman. P-selective sets, tally languages, and the behavior of polynomial time reducibilities on NP. *Mathematical Systems Theory*, 13:55–65, 1979.

24 S. Tang, B. Fu, and T. Liu. Exponential time and subexponential time sets. In *Theoretical Computer Science*, volume 115, pages 371–381, 1993.

25 S. Toda. On polynomial-time truth-table reducibilities of intractable sets to P-selective sets. *Mathematical Systems Theory*, 24(2):69–82, 1991.

26 L. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.

27 O. Watanabe. A comparison of polynomial time completeness notions. *Theoretical Computer Science*, 54:249–265, 1987.

A Proof of Key lemma

We prove the two parts of the Lemma 8 separately.

Proof of Part 1. Fix the input 0^{V_ℓ} from the hypothesis. Since we are operating under the assumption that the event $\mathcal{E}_\ell$ has occurred, there is a j, $1 \le j \le {V_\ell}^2$, such that $\mathcal{E}_{V_\ell,j}$ has occurred.

Recall that Q^j is the set of all queries made by the truth-table reduction $\langle g, h\rangle$ on inputs $\langle 0^n, k^j, r_1^j, \cdots, r_{k_j}^j, i\rangle$ for $1 \le i \le |a_{V_\ell}|$. Let Q_1^j be the set of all queries made to L_1 and let Q_2^j be the set of all queries made to the set L_2.

We will now describe how to decide answers to queries from Q_1^j. Note that each query $q \in Q_1^j$ is of the form $\langle \phi, z\rangle$, where ϕ is of length t. If z is not an accepting computation of $N(0^t)$, then $\langle \phi, z\rangle$ in not in L_1. Suppose z is an accepting computation of $N(0^t)$. Then it must be the case that $t \le \tau^\epsilon(((\log\log {V_\ell}^{b/\epsilon})^{1/c} - 1)^c)$. Now $\langle \phi, z\rangle$ is in L_1 if and only if $\phi \in \mathrm{SAT}'$. Since $|\phi| = t$, this can be tested in time 2^t. Thus we can decide the membership of all queries from Q_1^j in time polynomial in $\tau(\epsilon 2^{((\log\log {V_\ell}^{b/\epsilon})^{1/c}-1)^c})$.

Now we deal with the queries to L_2; we can not hope to decide membership of queries in Q_2^j in L_2 in general. However, note that L_2 is P-selective. Thus by Toda's lemma (Lemma 3) all elements of Q_2^j can be ordered as $q_1, q_2, \cdots, q_m$ and for some r, $1 \le r \le m$, all of $q_1, \cdots, q_r$ are in L_2 and none of $q_{r+1}, \cdots, q_m$ are in L_2. This ordering can be done in time polynomial in m. We do not know the value of r. However, there are only m possibilities for r. For each $i \in [1, m]$ we set r as i and determine the possible membership of queries in L_2.

We are now in the following situation: For every query in Q_1^j we know the membership in L_1, and for every query in Q_2^j we know candidate memberships in L_2. Using these and evaluating h, we obtain a candidate for the ith bit of u, for $1 \le i \le |a_{V_\ell}|$. From this we construct a string $\hat{u}$ and if $\hat{u}$ is an accepting computation of $N(0^n)$ we output it. If u is not an accepting computation of $N(0^n)$, we proceed with next choice for r. By our assumption, there is an accepting computation of $N(0^{V_\ell})$ that is isolated. Thus this process will find that accepting computation. Note that the total time taken by this process is still some polynomial in $\tau(\epsilon 2^{((\log\log {V_\ell}^{b/\epsilon})^{1/c}-1)^c})$.

Finally, we do not know the value of j for which the isolation occurs. We repeat the above process for every possible choice of j, and there are only n^2 choices for j. Thus the total time taken by this algorithm is still a polynomial in $\tau(\epsilon 2^{((\log\log {V_\ell}^{b/\epsilon})^{1/c}-1)^c})$. ◀

Proof of Part 2. The length of every query generated by $\mathcal{A}(0^{V_\ell})$ is at most ${V_\ell}^b$. Then it can be seen from algorithm $\mathcal{A}$ that on input 0^{V_ℓ}, when it outputs "Large Query", it outputs an accepting computation u_t of $N(0^t)$ such that

$$\tau^\epsilon(((\log\log {V_\ell}^{b/\epsilon})^{1/c} - 1)^c) < t < {V_\ell}^b.$$

Now if we can show that 0^t lies in T_ℓ for all possible values of t for such a V_ℓ, then there is at least one 0^t in T_ℓ whose accepting computation can be computed from a string 0^{V_ℓ} in T_ℓ. Here $V_\ell = \tau^{\epsilon/b}(\log^c(2^{\ell+1} - 2))$. Remember that $T_\ell = \{0^{\tau^\epsilon(\log^c r_x)} \mid 2^\ell - 1 \le r_x \le 2^{\ell+1} - 2\}$. Then it's easy to verify that the upper and lower bounds on t gives the last and the first strings in T_ℓ, respectively. We use this observation to construct $\mathcal{B}_2$:

> On input $0^t \in T_\ell$, run $\mathcal{A}(0^{V_\ell})$ to get u_t. Then verify if u_t is an accepting computation of $N(0^t)$. If it is, then output u_t.

The above observation implies that there is such a 0^t in T_ℓ on which $\mathcal{B}_2$ outputs u_t. The algorithm runs in time polynomial in V_ℓ. ◀

B Proof from Section 4

Proof of Proposition 1. Suppose L is a language that satisfies the hypothesis. Let N_1 and N_2 be two machines that witness that L and $\overline{L}$ are in NTIME($t(n)$). Consider the following machine N: On input x, guess $b \in \{1, 2\}$ and run N_b. Clearly, N is an NTIME($t(n)$) machine and accepts Σ^*. Suppose there is a probabilistic machine M running in time $2^{t(n)^\delta}$, such that for all but finitely many x, $M(x)$ outputs an accepting computation of $N(x)$ with probability at least $1/4$. Note that by looking at an accepting computation of $N(x)$, we can decide whether $x \in L$ or not. Consider a machine Z that on input x does the following:

> Run $M(x)$. If the output of $M(x)$ is an accepting computation of N_1, then Z accepts x. If the output of $M(x)$ is an accepting computation of N_2, then Z rejects x. Otherwise, Z outputs $\perp$.

Clearly, Z is a zero-error machine whose running time is $O(2^{t(n)^\delta})$ and for every x, Z correctly decides L with probability at least $1/4$. By running this machine $O(1)$ times, we obtain that L is in ZPTIME($2^{t(n)^\delta}$). This contradicts Hypothesis 2. ◀

Proof of Proposition 2. Consider the NP machine N from the NP-machine Hypothesis that runs in time n^d for some $d > 0$. Define a padding function $pad : \Sigma^* \to \mathbb{N}$ by

$$pad(x) = \tau((\log r_x - 1)^2),$$

where $c > 1$, and r_x is the lexicographic rank of x in Σ^*. Now define the NEEXP machine N_1 as follows: On input x, compute $m = pad(x)$ and run $N(0^m)$. Clearly, N accepts Σ^* and runs in time $\tau^d(n^2)$, for $x \in \Sigma^n$ (so $r_x \leq 2^{n+1} - 2$). Assume $t(n) = \tau^d(n^2)$.

Suppose there is a probabilistic machine Z_1 running in time $2^{t(n)^\delta}$, such that for all but finitely many x, $Z_1(x)$ outputs an accepting computation of $N_1(x)$ with probability atleast $1/4$. Consider a machine Z that on input 0^m does the following: Compute $x = pad^{-1}(m)$ and run $Z_1(x)$. Clearly, Z is a zero-error machine that runs in time

$$O(2^{t(n)^\delta}) = \tau(\delta d 2^{n^2}) < \tau(\beta 2^{n^2}) = 2^{m^\beta}$$

for some appropriate $\beta > \delta d$. Note that for every n, this time bound *holds* for the last string in Σ^n and *does not* hold for the first string in Σ^n. Clearly, there are infinitely many 0^m where Z correctly computes the accepting computation of $N(0^m)$ in time 2^{m^β}. If we assume that full derandomization is possible, then we can replace Z by a deterministic machine M that runs in time $poly((2^{m^\beta}))$ which is 2^{m^ϵ} for an appropriate $\epsilon > \beta$. Hence contradiction. ◀