A Generic Time Hierarchy for Semantic Models With One Bit of Advice

Dieter van Melkebeek* Konstantin Pervyshev†

April 21, 2007

Abstract

We show that for any reasonable semantic model of computation and for any positive integer a and rationals $1 \leq c < d$, there exists a language computable in time n^d with a bits of advice but not in time n^c with a bits of advice. Our result implies the first such hierarchy theorem for randomized machines with zero-sided error, quantum machines with one- or zero-sided error, unambiguous machines, symmetric alternation, Arthur-Merlin games of any signature, etc. Our argument yields considerably simpler proofs of known hierarchy theorems with one bit of advice for randomized and quantum machines with two-sided error.

Our paradigm also allows us to derive stronger separation results in which the machine with the smaller running time can receive more advice than the one with the larger running time. We present a unified way to derive such results for randomized and quantum machines with two-sided error and for randomized machines with one-sided error. In each case the advice for the machine with the larger running time has a very natural meaning.

1 Introduction

Hierarchy theorems address one of the most fundamental questions in computational complexity: Can we decide more languages on a certain model of computation when given somewhat more of a certain resource? In fact, a time hierarchy for deterministic Turing machines constitutes the main technical contribution in the paper by Hartmanis and Stearns [HS65] that founded the field. Later on, Cook [Coo73], Seiferas, Fischer and Meyer [SFM78], and Žàk[Ž83] established time hierarchies for nondeterministic Turing machines. Their techniques apply to virtually any *syntactic* model of interest, i.e., one for which there exists a computable enumeration of exactly the machines in the model.

Several models we care about are not syntactic, though. Examples include randomized or quantum machines with two-, one-, or zero-sided error, unambiguous machines, symmetric alternation, Arthur-Merlin games of any signature, etc. Each of these models has a computable enumeration that contains all machines of the model but not one that consists exactly of all machines in the model. For example, we can computably enumerate all randomized machines but there does not exist a computable enumeration of only those randomized machines that satsify the promise of

*Department of Computer Sciences, University of Wisconsin - Madison. Email: `dieter@cs.wisc.edu`. Partially supported by NSF Career award CCR-0133693, EIA-0205236, and EMT-0523680.

†Department of Mathematics and Mechanics, St. Petersburg State University. Email: `pervyshev@logic.pdmi.ras.ru`.

bounded error on all inputs. We dub models with that property as *semantic*. See Section 4.1 for more on nomenclature.

To date, except for a few cases in which a non-syntactic model is known to be equivalent in power to a syntactic one, no hierarchy is known for any non-syntactic model[1]. In particular, it remains open whether for every constant c there exists a language that can be solved on randomized machines with two-sided error in polynomial time but not in time n^c.

In 2002, Barak [Bar02] used instance checkers for exponential-time complete languages to prove the latter statement in a slightly nonuniform version of the model, namely a model in which the machines get $a(n)$ bits of advice for some function $a(n) = O(\log \log n)$ and where the machine is only required to satisfy the promise of bounded error for the correct advice string.[2] In other words, he established the result for randomized machines with two-sided error whose descriptions can depend on the input length n in such a way that the size of the variable part is bounded by $a(n)$. Subsequently, several authors tried to get as close as possible to the desired uniform result and managed to reduce the amount of advice to a single bit [FS04, GST04]. Barak's argument also applies to quantum machines with two-sided error but not to any of the other non-syntactic models on our list. Roughly speaking, due to the use of instance checkers, the model has to be closed in an efficient way under randomized reductions with two-sided error for the proof to carry through.

More recently, Fortnow, Santhanam, and Trevisan [FST05] gave a specific argument for randomized machines with one-sided error and one bit of advice. They also developed an approach that works for all of the above models but needs considerably more advice: They obtain a hierarchy theorem for any reasonable semantic model of computation with $a(n)$ bits of advice where $a(n)$ is some function in $O(\log n \cdot \log \log n)$.

As our main result, we manage to get the best of both worlds and thereby improve both lines of research.

Theorem 1 *For any reasonable semantic model of computation and any constants a and c, there exists a language computable in polynomial time with one bit of advice but not in time n^c with a bits of advice.*

As a corollary to Theorem 1, we obtain the following hierarchy with a bits of advice for any constant $a \geq 1$.

Theorem 2 *For any reasonable semantic model of computation and any positive integer a and rationals $1 \leq c < d$, there exists a language computable in time n^d with a bits of advice but not in time n^c with a bits of advice.*

We refer to Section 4.2 for a precise definition of "reasonable" but all of the specific models listed above fall under the notion. Theorems 1 and 2 generalize to superpolynomial time bounds – see Section 4.5 for the statement.

We use the technique of delayed diagonalization adapted to the setting of computations with advice. Our approach differs from Barak's as well as the one by Fortnow et al. Like the latter but unlike the former, our proof relativizes. Since instance checkers are the sole culprit of nonrelativization in Barak's argument, our proof shows that that component is not critical for obtaining

[1] Here, we are assuming that one interprets "somewhat more time" as implying "at most a polynomial amount more time." The exceptions we are aware of follow from the characterizations IP = PSPACE, MIP = NEXP, and PCP($\log n$, 1) = NP, and BP$\cdot\oplus$P = $\Sigma_2^{\oplus \mathrm{P}}$.

[2] This notion of advice differs from the one by Karp and Lipton. See the end of Section 4.1 for more details.

a time hierarchy for randomized machines with two-sided error and one bit of advice. Apart from yielding stronger results and being more widely applicable, our approach also provides considerably simpler proofs for all the hierarchy theorems with one bit of advice that were known before [FS04, GST04, FST05], and the one bit of advice we need has a very natural meaning – whether a certain machine of the enumeration satisfies the promise at a certain length. We refer to Section 2 for a more detailed comparison of techniques.

As is clear from the statement of Theorem 1, the proof of our main result actually yields more than a hierarchy theorem because we can accommodate up to a bits of advice for any constant a at the smaller time bound while still only needing a single bit of advice at the larger time bound. Barak's argument goes further along that road and handles up to $a \log n$ instead of a bits of advice but only for a more restrictive subclass of semantic models. We show how to match Barak's bound of $a \log n$ using our approach.

Theorem 3 *For any reasonable randomized semantic model of computation that is efficiently closed under randomized reductions with two-sided error, and any constants a and c, there exists a language computable in polynomial time with one bit of advice but not in time n^c with $a \log n$ bits of advice.*

We refer to Section 5.2 for a full specification of the models to which Theorem 3 applies; the list includes randomized and quantum machines with two-sided error. Our proof of Theorem 3 uses instance checkers again but in a different way than Barak and for a more limited purpose. Thus, we further relegate the use of instance checkers in this context. The role of the one bit of advice is also more clear in our approach.

Theorem 3 does not seem to apply to randomized machines with one-sided error. For that specific model, Fortnow et al.'s argument[3] yields a somewhat weaker separation theorem, namely for $a(\log n)^{1/c}$ bits of advice instead of $a \log n$ bits at the smaller time bound of n^c. We show how to obtain that result using our approach, too.

Theorem 4 *For any constants a and c there exists a language computable by randomized machines with one-sided error in polynomial time with one bit of advice but not in time n^c with $a(\log n)^{1/c}$ bits of advice.*

A modification of the proof of Theorem 4 also allows us to establish Theorem 3 for the specific model of randomized machines with two-sided error in a *relativizable* way, i.e., without the use of instance checkers. Thus, the paradigm we present offers a unified way for deriving new as well as known separation results within non-syntactic models of computation. In all of our constructions, the one bit of advice has the same intuitive meaning described above.

The rest of this paper is organized as follows. In Section 2 we provide an overview of the arguments that have been used for deriving hierarchy theorems in the past. Section 3 describes the intuition behind our constructions and develops them in an informal way. Section 4 contains the formal presentation of our generic hierarchy theorem, and Section 5 does the same for our separation theorems. Finally, in Section 6, we present some possible directions for further research.

[3]Fortnow et al. [FST05] actually only prove the result for $(\log n)^{1/2c}$ bits of advice but a small modification of their argument works up to $a(\log n)^{1/c}$ bits of advice at the smaller time bound of n^c.

2 Previous Work

In this section, we survey the arguments that have been used in hierarchy theorems and that exhibit a close relationship to ours. We focus on techniques and qualitative improvements rather than quantitative ones. Readers who would like to skip to Section 3 for a description of our constructions can do so without loss of continuity.

For their seminal hierarchy theorem, Hartmanis and Stearns [HS65] used a diagonalization technique rooted in Cantor's proof that the reals are not countable. They assume the model of computation has a computable enumeration of machines and a universal machine U. They pick an infinite sequence of inputs $x_1, x_2, \ldots$, and use x_i to diagonalize against the ith machine M_i of the enumeration by running the universal machine on $\langle M_i, x_i, 0^t\rangle$, where t denotes the allotted amount of time, and doing the opposite. This approach results in a time hierarchy for essentially any syntactic model with an efficient universal machine for which "doing the opposite" is easy.

We don't know whether "doing the opposite" is easy in models like nondeterministic machines. We can run a deterministic simulation and complement the result but that involves an exponential slowdown. Cook [Coo73] was the first to get around the need for easy complementation. His proof works by contradiction and goes as follows.

Assume the hierarchy theorem for nondeterministic machines fails. Then for every polynomial-time nondeterministic machine there exists an equivalent nondeterministic machine that runs in time n^c for some fixed c. Applying this speedup $O(\log n)$ times in a uniform way (exploiting the existence of a universal machine) shows that even every exponential-time nondeterministic machine has an equivalent nondeterministic machine that runs in time n^c. We can simulate the latter nondeterministic machine on a deterministic one in time 2^{n^c}. On the other hand, deterministic machines are also nondeterministic machines. Thus, we obtain a simulation of every exponential-time deterministic machine by another deterministic machine that runs in time 2^{n^c} – a contradiction with the time hierarchy for deterministic machines.

Seiferas et al. [SFM78] use a more direct argument and explicitly construct a language L that witnesses the nondeterministic time hierarchy for a given constant c. They start from any computable language L' that cannot be decided by nondeterministic machines in time n^{c+1}, e.g., a complete language for double exponential time. They define L as the language decided by the nondeterministic machine M that acts as follows on strings of the form $\langle x, i, 0^k\rangle$. Let M' denote a fixed deterministic machine that decides L'. If k is larger than the running time of M' on x, then M outputs the result of that computation. Otherwise, M uses the universal machine to simulate M_i on input $\langle x, i, 0^{k+1}\rangle$ for n^c steps. M runs in polynomial time but the language L it defines cannot be decided by nondeterministic machines that run in time n^c. Indeed, suppose that M_i were such a machine. For small k, we have that $M_i(\langle x, i, 0^k\rangle) = M(\langle x, i, 0^k\rangle) = M_i(\langle x, i, 0^{k+1}\rangle)$, and for large k that $M_i(\langle x, i, 0^k\rangle) = M(\langle x, i, 0^k\rangle) = M'(x)$. It follows that $M'(x) = M_i(\langle x, i, \epsilon\rangle)$ for each x. Since M_i runs in time n^c, this contradicts the fact that the language L' decided by M' cannot be decided by a nondeterministic machine in time n^{c+1}.

Žàk's argument [Ž83] is similar but replaces the use of a difficult language L' by delayed diagonalization. Essentially, on inputs of the form $\langle x, i, 0^k\rangle$, the role of M' is taken over by the complement of the deterministic simulation of M_i for n^c steps. The rest of the argument is analogous: Suppose that M_i runs in n^c steps and is equivalent to M, and let k be the first large value (for the given i). We have on the one hand that $M_i(\langle x, i, 0^k\rangle) = M_i(\langle x, i, 0^{k-1}\rangle) = \ldots = M_i(\langle x, i, \epsilon\rangle)$ and on the other hand that $M_i(\langle x, i, 0^k\rangle) = \neg M_i(\langle x, i, \epsilon\rangle)$. Thus, M_i is not equivalent to M or takes more than n^c steps.

As a side note, we point out that it suffices for the machine M in Žàk's construction to act as described on *some* input x, say $x = \epsilon$, whereas Seiferas et al. in principle need the behavior on *every* x. Thus, Žàk's argument naturally leads to a *unary* language L that can be decided by nondeterministic machines in polynomial time but not in time n^c.

The constructions by Cook, Seiferas et al., and Žàk work for any syntactic model that has an efficient universal machine and is efficiently closed under deterministic transducers. For Cook's argument, we also need the existence of deterministic simulations that incur a non-exorbitant slowdown; exponential overhead as in the case of nondeterministic machines is fine. This essentially corresponds to what we mean by a "reasonable" syntactic model of computation. See Section 4.2 for the formal definitions.

Unfortunately, none of these techniques seem to extend to *semantic* models because they all involve simulations of arbitrary machines of the enumeration. For example, in the case of randomized machines with two-sided error, simulating a randomized machine M_i on an input on which M_i accepts with probability 50% would take M outside of the model because its error probability is not bounded away from 50%.

Instance checkers are tools that enable us to refrain from making errors. Recall that an instance checker for a language L' is a polynomial-time randomized oracle machine C that can output 0, 1, or "I don't know" on any input x such that the following properties hold: (i) $C^{L'}(x)$ outputs $L'(x)$ with probability 1, and (ii) for any oracle P, $C^P(x)$ outputs $\neg L'(x)$ with exponentially small probability. Barak [Bar02] had the insight that an instance checker for a language L' in exponential time yields a randomized machine M' with two-sided error that decides L' and has a running time that is optimal up to a polynomial factor. The machine M' acts as follows: For $k = 1, 2, \ldots$ and for $i = 1, \ldots, k$, run C^{M_i} for k steps and halt as soon as one of the runs of the instance checker comes to a 0/1 conclusion; then output that conclusion. Let $t(n)$ denote the worst-case high-confidence running time of M' on inputs of length n. The properties of the instance checker imply that (a) M' decides L' with exponentially small two-sided error, (b) $t(n)$ is exponentially bounded, and (c) for some positive constant α, no machine M_i can do the same in $(t(n))^\alpha$ steps. The details of the argument are not relevant for us but the intuition for the optimality property (c) is that M' would start running the instance checker with oracle M_i as soon as $k \geq i$; if M_i were to decide L' with high confidence within $(t(n))^\alpha$ steps for some sufficiently small positive constant α, then M' would halt with high confidence within fewer than $t(n)$ steps.

If L' is complete for exponential time and $t(n)$ is polynomially bounded then we can efficiently transform every exponential-time deterministic machine into an equivalent polynomial-time randomized machine with two-sided error. We can trivially transform a polynomial-time randomized machine into an equivalent exponential-time deterministic machine. The desired hierarchy theorem for randomized machines with two-sided error (at the polynomial-time level) then follows from the hierarchy theorem for deterministic machines (at the exponential-time level).

If $t(n)$ is not polynomially bounded then for any constant c there are infinitely many input lengths n such that $(t(n))^{\alpha/4c} \geq n + 1$. Suppose we could efficiently compute a value $t^*(n)$ such that $(t(n))^{\alpha/4c} < t^*(n) \leq (t(n))^{\alpha/2c}$. Then padding strings of length n in L' to length $t^*(n)$ would yield a language $L = \{x10^{t^*(|x|)-|x|-1} \mid x \in L' \text{ and } t^*(|x|) \geq |x| + 1\}$ computable by randomized machines with two-sided error in polynomial time but not in time n^c. We chose the range for $t^*(n)$ such that there exists a (unique) value of the form $t^*(n) = 2^{2^{\tau^*(n)}}$ in that range with $\tau^*(n)$ integer. Computing $\tau^*(n)$ may be difficult but its value can be specified using $\log\log t^*(n)$ bits. Therefore, L can be decided by a randomized machine M with two-sided error in polynomial time with $a(n) = \log\log n$ bits of advice but not by such machines in time n^c without advice.

This isn't a fair time hierarchy theorem yet – for that, the time n^c machines should be allowed the same amount of advice as M. We can satisfy that requirement by tweaking the construction of the machine M' such that it runs each of the machines M_i with every possible advice string of length $\log\log k$. In fact, we can accommodate up to $a \log k$ bits of advice for the M_i's for any constant a. Both the case where $t(n)$ is polynomially bounded (now needing a hierarchy theorem for deterministic machines with advice) and the other case carry through.

Moreover, once the advice for the witnessing machine M is under $\log n$ bits, we can apply a translation technique and obtain a hierarchy theorem with a single bit of advice. This involves another level of padding to encode the $a(n) < \log n$ bits of the original advice for M in the padding length and using the one bit of new advice to indicate whether the padding length is valid. See [FS04, GST04] for the details. This way, we obtain a language which randomized machines with two-sided error can decide in polynomial time and one bit of advice but not in time n^c and $a \log n$ bits of advice. The same strong separation holds for any reasonable semantic model of computation with the additional property of being efficiently closed under randomized reductions with two-sided error. We refer to Section 5.2 for the formal definitions.

Semantic classes with one-sided error typically do not exhibit the latter additional closure property. For the specific model of randomized machines with one-sided error, Fortnow et al. [FST05] use a modification of the above two-case approach to derive a somewhat weaker separation result, namely with $a(\log n)^{1/c}$ instead of $a \log n$ bits of advice at the smaller time bound of n^c. See Theorem 4 and footnote 3 on page 3 for the precise statement. Instead of an exponential-time complete language L' and Barak's optimal algorithm based on instance checkers, Fortnow et al. consider an NP-complete language L and Levin's optimal algorithm based on searching for NP-witnesses [Lev73]. The more restrictive advice bound of $a(\log n)^{1/c}$ is dictated by the separation result for nondeterministic machines with advice, which is needed for the case where $t(n)$ is polynomially bounded.

For their actual hierarchy theorem (where the length of the advice is the same for both time bounds considered), Fortnow et al. manage to eliminate the need for additional model requirements but they can only do so for some advice function in $O(\log n \cdot \log\log n)$. Their approach can be viewed as running Cook's argument with advice. The $\log n$ term in the advice bound comes from the $O(\log n)$ levels in Cook's argument. The $\log\log n$ term per level comes from a padding argument similar to Barak's.

Using a different strategy, we manage to get the advice down to a single bit. In fact, we obtain a hierarchy theorem with a bits of advice for any reasonable semantic model and any constant $a \geq 1$. We view our approach as extending Žàk's delayed diagonalization argument to machines with a bits of advice. A similar extension of Seiferas et al.'s argument leads to the same result but the proof becomes more complicated [Per05]. The latter approach seems more suitable for obtaining hierarchy theorems with one bit of advice that are conditional on a complexity class separation such as P $\neq$ NP [GHP05].

3 Intuition and Informal Derivation

In this section, we first sketch the construction of our generic hierarchy theorem with a constant number of bits of advice, and then the argument for our separation theorems. The formal proofs will be given in Sections 4 and 5, respectively.

3.1 Hierarchy Theorem

Consider a semantic model of computation with enumeration $M_1, M_2, \ldots$. We assume that there exists some underlying notion of "promise" which allows us to tell whether M_i with advice sequence $\alpha = \alpha_0, \alpha_1, \alpha_2, \ldots$, satisfies the promise on a given input x. Whether the latter is the case only depends on the behavior on input x; in particular, it is determined by M_i and the component $\alpha_{|x|}$ of the advice sequence α. We use the notation M_i/α to denote M_i with advice sequence α, and $M_i//\alpha_n$ to denote M_i with advice α_n at a fixed length n. M_i/α falls within the model iff $M_i//\alpha_n$ satisfies the promise at every length n.

Let us try to use straightforward diagonalization to establish a hierarchy theorem with $a \geq 0$ bits of advice. For a given constant $c \geq 1$, we would like to construct a machine M and an advice sequence α of modulus a (i.e., $|\alpha_n| = a$ for each length n), such that M/α falls within the model, takes not much more than n^c time, and disagrees with each M_i/β for each advice sequence β of modulus a for which M_i/β falls within the model and runs in time n^c.

With each M_i we associate a length n_i and distinct strings $x_{i,b}$ of length n_i for each value of $b \in \{0,1\}^a$. If $M_i//b$ satisfies the promise on $x_{i,b}$ and runs in time n_i^c, we would like to have M/α do the opposite of $M_i//b$ on that input. Assuming the existence of an efficient universal machine U, we would set

$$M/\alpha\,(x_{i,b}) = \neg U(\langle M_i//b, x_{i,b}, 0^{n_i^c}\rangle). \tag{1}$$

There are two problems with this approach. First, complementation may not be easy within the model. Second, even if complementation is easy, the simulation (1) may violate the promise. Recall that M/α has to satisfy the promise everywhere, whereas $M_i//b$ (run for n_i^c steps) may violate the promise on input $x_{i,b}$ for some values of b. Of course, there is no need to diagonalize in the case where b does not work for M_i on input $x_{i,b}$, i.e., if $M_i//b$ does not satisfy the promise on $x_{i,b}$ or takes more than n_i^c time. In that case, M/α can do something trivial, e.g., reject irrespective of the input. However, figuring out whether b works for M_i on input $x_{i,b}$ may not be easy. We could tell M for each value of b whether b works for M_i on $x_{i,b}$ but that would require $2^a > a$ bits of advice for M_i at length n_i. In fact, with 2^a bits of advice we could tell M explicitly how to behave like (1) on the 2^a strings $x_{i,b}$.

By adapting the technique of delayed diagonalization, we can spread out the above 2^a bits of advice needed at some length over different smaller lengths and thereby reduce the amount of advice M needs at any given length to a single bit. Delayed diagonalization consists of a slow complementation executed at a larger input length n_i^* and a process to copy down the complementary behavior to length n_i. We will use a slow but safe simulation of $\neg U$, i.e., a machine S which always satisfies the promise and agrees with $\neg U$ on input x whenever U satisfies the promise on x. M may not have enough time to run S on $\langle M_i//b, x_{i,b}, 0^{n_i^c}\rangle$ at length n_i but it certainly does at a sufficiently larger length n_i^*, typically $n_i^* = 2^{n_i^c}$. We then set up M and α on lengths between n_i and n_i^* in such a way that if M_i/β satisfies the promise, runs in time n^c, and agrees with M/α for some advice sequence β of modulus a, then M/α "copies" its behavior at length n_i^* down to certain smaller and smaller lengths. If we can reach length $n = n_i$, we have the following contradiction for $b = \beta_n$:

$$M_i/\beta\,(x_{i,b}) = M/\alpha\,(x_{i,b}) = S(\langle M_i//b, x_{i,b}, 0^{n^c}\rangle) = \neg U(\langle M_i//b, x_{i,b}, 0^{n^c}\rangle) = \neg M_i/\beta\,(x_{i,b}). \tag{2}$$

This way we succeed in diagonalizing against M_i/β for any advice sequence β of modulus a.

The copying process capitalizes on M's ability to spend polynomially more time than the n^c steps M_i is allotted. This allows M to simulate M_i on polynomially larger inputs. Consider length

$n' = n_i^*$ and each possible value of $b \in \{0,1\}^a$. We say that b works for M_i at length n' if b works for M_i on all inputs of length n', i.e., $M_i//b$ satisfies the promise on all inputs of length n' and runs in time $(n')^c$. In that case, we pick some smaller but polynomially related length n and allow M/α on inputs x of length n to run $M_i//b$ on the input $0^{n'-n}x$ of length n'. As a result, we have that

$$(\forall\, x \in \{0,1\}^n)\ M/\alpha\,(x) = M_i//b\,(0^{n'-n}x).$$

We say that M/α at length n copies $M_i//b$ at length n'. If b does not work for M_i at length n', we let M/α act trivially at length n'. We use different lengths n for different values of b in such a way that b and n' are efficiently recoverable from n. Thus, M only needs a single bit of advice α_n at each length n, namely whether or not b works for M_i at length n'.

We then recursively apply the process to all 2^a lengths n we introduced, each time fixing the behavior of M/α at new lengths n. Provided we do not run out of lengths, we reach a point where the lengths n become so small that $S(\langle M_i//b, x, 0^{n^c}\rangle)$ runs in time polynomial in n_i^* for strings x of length n. At that point, the copying process bottoms out and we try to diagonalize as indicated above: For each $b \in \{0,1\}^a$, we pick a different string $x_{i,b,n}$ of length n, e.g., $x_{i,b,n} = 10^{n-a-1}b$, and define

$$M/\alpha\,(0^{n_i^*-n}x_{i,b,n}) = S(\langle M_i//b, x_{i,b,n}, 0^{n^c}\rangle). \tag{3}$$

The pattern $1(0+1)^*$ for the strings $x_{i,b,n}$ ensures the compatibility of (3) for different lengths n. On strings of length n_i^* that are not of the form $0^*x_{i,b,n}$, M/α acts trivially. If we make sure that n_i^* and the bottom-out lengths n are efficiently recognizable, M does not need any advice at length n_i^*.

One can think of the copying process as constructing a tree from the root n_i^* to the leaves. Each copying step creates 2^a siblings that are connected to their parent n' through an edge labeled with a corresponding value of $b \in \{0,1\}^a$. The process associates a unique length to each non-root node and determines the behavior of M/α at that length by specifying the corresponding advice bit. It leaves the behavior at the root length n_i^* free to be used for the diagonalization.

Now, suppose that for some advice sequence β of modulus a, M_i/β falls within the model, runs in time n^c, and agrees with M/α. Consider the path from the root n_i^* to a leaf n obtained by selecting at every non-leaf node n' the edge labeled $\beta_{n'}$. For each edge on that path, its label works for M_i at the parent node n' so M/α at the child node n copies M_i/β at the parent node n'. Since M/α and M_i/β agree, this means that the behavior of M/α is copied down along that path and that M/α at length n copies M/α at length n_i^*. However, (3) then leads to the contradiction (2) for $b = \beta_n$ and $x_{i,b} = x_{i,\beta_n,n}$.

To finish the argument, we need to argue that we have enough lengths n available to execute the above process. We can assign subsequent lengths from left to right to any given level of the copying tree, with gaps between the intervals used for adjacent levels. Let n_i denote the start of the first interval and k_i denote the number of intervals, i.e., the number of levels of the tree. The jump from the start of any interval to the start of the next one can be an arbitrary but fixed polynomial, say from n to n^d. Assuming the safe simulation S runs in exponential time, we need $\Theta(\log n_i/\log d)$ such jumps to go from n_i to n_i^* so we set $k_i = \Theta(\log n_i/\log d)$. The first interval forms the bottleneck for the embedding because it is the largest one and the gap that is available for it is smallest. The first interval contains $(2^a)^{k_i-1} = n_i^{\Theta(a/\log d)}$ elements, which fit within the gap between n_i and n_i^d provided d is a sufficiently large. Thus, we can accommodate all intervals without overlap. We refer to the formal proof in Section 4.3 for a more detailed calculation.

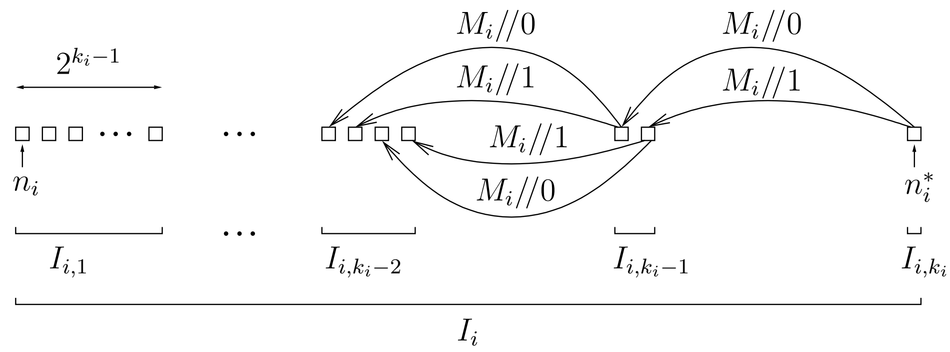

Figure 1: Construction of M on I_i for $a = 1$ in Theorem 1. An arrow from length n' to length n labeled $M_i//b$ denotes that $M//1$ at length n copies $M_i//b$ at length n'.

Figure 1 illustrates the process for $a = 1$. In that case, the tree is binary; interval $I_{i,j}$ in the figure contains the 2^{k_i-j} nodes at depth $k_i - j$ of the copying tree, $1 \leq j \leq k_i$.

We managed to let M/α diagonalize against M_i/β for any β of modulus a. We did so by specifying the behavior of M/α on some lengths n in the interval $I_i = [n_i, n_i^*]$, while always making sure that M/α satisfies the promise and runs in some fixed polynomial amount of time. To handle all machines M_i in one construction, we use disjoint intervals I_i for different machines M_i and let M/α act trivially on those lengths n we do not use during the process.

The above technique applies to any semantic model that has an efficient universal machine which can be complemented within the model in exponential time, and that is efficiently closed under deterministic transducers. Taking these properties as the definition for a reasonable semantic model, we obtain Theorem 1. Theorem 2 follows from Theorem 1 by a standard padding argument. We refer to Section 4 for the details.

Before moving on to our stronger separation results, let us point out the intuitive role the one bit of advice for M plays: It allows us to prevent M/α from simulating machines M_i/β on inputs where they do not satisfy the promise – a critical issue in semantic non-syntactic models.

3.2 Separation Theorems

The above approach only works for bounded modulus $a(n)$. For unbounded modulus $a(n)$, the number of leaves of the copying tree becomes super-polynomial in the largest length ℓ associated to a leaf, which is incompatible with the requirement that each leaf maps to a unique length. Even if we are willing to give M $a(n)$ bits of advice at length n, the issue remains.

We get around the problem by restricting the behavior of M/α in such a way that it can be safely recovered at length n' from any list of machines at least one of which works appropriately at length n'. By the latter we mean: satisfying the promise at length n', running in time $(n')^c$, and agreeing with M/α at length n'. We can then modify the process for copying from length n' to length n as follows. At length n, M/α gets as advice whether there exists a string $b \in \{0,1\}^{a(n')}$ such that $M_i//b$ works appropriately at length n'. In case the advice bit is set, on an input x of length n, M/α runs the above recovery procedure for M/α on input $0^{n'-n}x$ using the list of machines $M_i//b$ for each $b \in \{0,1\}^{a(n')}$; as a result, M/α at length n copies M/α at length n'. Otherwise, M/α acts trivially at length n.

Notice that there no longer is a need for multiple lengths n to map to the same length n'. The copying tree becomes a line with root at length n_i^* and a unique leaf at length n_i. There also no longer is a need to make large (polynomially bounded) jumps from n to n'. We needed those in Section 3.1 just to ensure enough space for embedding the intervals. Since the intervals are now of length 1, we could set $n' = n + 1$. Since there is only one leaf, the structure of the copying tree on its own does not impose any limitations on the size of the modulus. As the recovery procedure needs to consider $M_i//b$ for each possible $b \in \{0,1\}^{a(n')}$, $a(n')$ has to be logarithmically bounded for M/α to run in polynomial time. Therefore, logarithmic moduli are the best one can hope for using this approach.

Safe recovery is only possible in some settings. We know of two basic mechanisms, namely instance checking and membership proof recovery. Both severely restrict the behavior of M/α and take away the freedom to define M/α at length n_i^* so as to complement $M_i//b$ at length n_i. Thus, for each mechanism we need new strategies to diagonalize. The models of computation also need to have the necessary closure properties to accommodate the recovery process based on instance checkers or membership proofs, respectively.

We use an instance checker to copy down EXP-complete behavior and then exploit that to diagonalize assuming the model allows complementation in EXP. We develop this approach in Section 3.2.1. It works up to the limit of logarithmic modulus.

We use membership proofs to copy down NP-complete behavior. Assuming the model allows an efficient simulation in NP, we obtain an efficient safe simulation which we then use to simplify the construction from Section 3.1. We develop this approach in Section 3.2.2. It works up to modulus $\Theta((\log n)^{1/c})$ for randomized machines with one-sided error and up to modulus $\Theta(\log n)$ for randomized machines with two-sided error.

3.2.1 Copying using instance checking

Recall that an instance checker for a language L is a polynomial-time randomized oracle machine C that can output 0, 1, or "I don't know" on any input x such that the following properties hold: (i) $C^L(x)$ outputs $L(x)$ with probability 1, and (ii) for any oracle P, $C^P(x)$ outputs $\neg L(x)$ with exponentially small probability. There exist instance checkers for certain paddable exponential-time complete languages L that only make queries of length $f \cdot n$ on inputs of length n for some constant $f \geq 1$. For ease of exposition, we assume in this section that $f = 1$. The formal proof in Section 5.3 shows how to eliminate that assumption.

The key for safe recovery of L is roughly the following computation: For each possible advice string b of length $a(n)$, run the instance checker C with the oracle defined by n^c computation steps of $M_i//b$ at length n; halt as soon as one of these runs produces a 0/1 conclusion and then output that conclusion. Provided the model of computation is closed under randomized reductions with two-sided error, the properties of the instance checker guarantee that this computation works appropriately as long as there exists at least one advice string b for which $M_i//b$ works appropriately.

Let us be a bit more precise. Let $n_{i,j}$, $1 \leq j \leq k_i$, denote the lengths associated to the nodes of the copying line, where $n_{i,1} = n_i$ and $n_{i,k_i} = n_i^*$. On input $0^{n_i^* - n_i}x$, where x is a string of length n_i, M runs a fixed deterministic exponential-time algorithm for L on input x. For any $1 \leq j < k_i$, $M//1$ acts as follows on inputs of the form $0^{n_{i,j} - n_i}x$ where x is a string of length n_i: For each advice string b of length $a(n_{i,j+1})$, run the instance checker C on input x answering each query y by taking the majority vote of a linear number of independent runs of $U(\langle M_i//b, 0^{n_{i,j+1} - n_i}y, 0^{(n_{i,j+1})^c}\rangle)$; halt as soon as one of these computations yields a 0/1 conclusion and then output that conclusion. $M//1$ acts trivially on other inputs of length $n_{i,j}$, as does $M//0$ on all inputs.

We say that $M_i//b$ works appropriately at length n if b works for M_i at that length and L at length n_i is a copy of $M_i//b$ at length n, i.e., for each string x of length n_i, $M_i//b\,(0^{n_{i,j}-n_i}x) = L(x)$. We set $\alpha_{n_{i,j}}$ for $1 \leq j < k_i$ to indicate whether there exists a string b of length $a(n_{i,j+1})$ such that $M_i//b$ works appropriately at length $n_{i,j+1}$. If so, we know that L at length n_i is a copy of M/α at length $n_{i,j}$.

If the copying process succeeds, we have that $\alpha_{n_i} = 1$ and therefore M/α agrees with the exponential-time complete language L at length n_i. We exploit this fact to accomplish the desired diagonalization as follows. We introduce a new length $\tilde{n}_i$ smaller than n_i. For any string b of length $a(\tilde{n}_i)$, consider the complement of the deterministic simulation of $U(\langle M_i//b, 0^{\tilde{n}_i - a(\tilde{n}_i)}b, 0^{\tilde{n}_i^c}\rangle)$. Assuming that computation runs in deterministic exponential time, we can compute in polynomial time a string $z_{i,b}$ such that $L(z_{i,b}) = \neg U(\langle M_i//b, 0^{\tilde{n}_i - a(\tilde{n}_i)}b, 0^{\tilde{n}_i^c}\rangle)$. Using the paddability of L, we can set up things such that the length of $z_{i,b}$ equals n_i.

$M//1$ on input $0^{\tilde{n}_i - a(\tilde{n}_i)}b$ then runs $M//1$ on input $z_{i,b}$. Like before, $M//1$ acts trivially on other strings of length $\tilde{n}_i$, as does $M//0$ on all strings. We set $\alpha_{\tilde{n}_i}$ to indicate whether $M//1$ agrees with L on inputs of length n_i.

Now, suppose there exists an advice sequence β of modulus $a(n)$ such that M_i/β falls within the model, runs in time n^c, and agrees with M/α. Then the copying process is guaranteed to succeed and we obtain a contradiction similar to (2): For $b = \beta_{\tilde{n}_i}$ and $x_{i,b} = 0^{\tilde{n}_i - a(\tilde{n}_i)}b$,

$$M_i/\beta(x_{i,b}) = M/\alpha\,(x_{i,b}) = M/\alpha\,(z_{i,b}) = L(z_{i,b}) = \neg U(\langle M_i//b, x_{i,b}, 0^{\tilde{n}_i^c}\rangle) = \neg M_i/\beta\,(x_{i,b}).$$

Note that M/α at length n runs the instance checker C at most $2^{a(n')}$ times, where $n' = n^{O(1)}$. It follows that M/α runs in polynomial time as long as $a(n') \leq a \log n'$ for some constant a and the model is efficiently closed under randomized reductions with two-sided error. This approach works for any reasonable randomized semantic model with the latter closure property, thus establishing Theorem 3.

Let us end the informal treatment by reiterating the role of the instance checkers in our construction: They provide us an advice efficient way to realize the desired copying by M while always satisfying the promise. We want the copying to happen as soon as there exists at least one advice string b for which $M_i//b$ behaves appropriately at length n'. Before, M needed a separate bit of advice for each possible advice string b, namely to indicate whether $M_i//b$ behaves appropriately at length n'. Now, we can handle all possibilities for b at once using a single bit of advice for M, namely whether there exists at least one choice of b for which $M_i//b$ behaves appropriately at length n'.

3.2.2 Copying using membership proof recovery

Consider a language L that has membership proofs and for which the search for a membership proof at length n reduces to L at length n. Satisfiability is an example of such a language L. The crux for the safe recovery of L is the following computation: For each possible string b of length $a(n)$, run the reduction using the oracle defined by n^c computation steps of $M_i//b$ at length n; verify those candidate membership proofs and accept iff at least one of them is valid. Models like randomized machines with one- or two-sided error allow the efficient simulation of the above process. Provided the model has the latter closure property, we can develop a copying process with one bit of advice in a similar way as in Section 3.2.1. It uses a sequence of lengths from m_i to $m_i^* = 2^{m_i^{O(1)}}$ with jumps bounded by some fixed polynomial, and allows us to assume that $M//1$ decides L at length m_i.

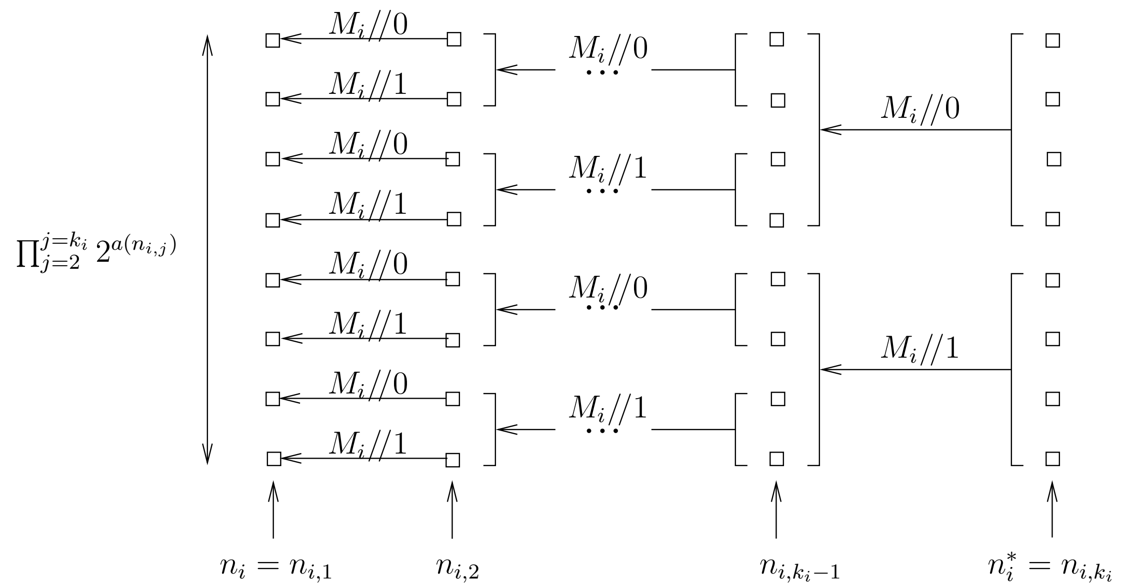

Figure 2: Partial construction of M on I_i for $a(n) = 1$ in Theorem 3. Each box contains two distinct strings, one corresponding to each value in $\{0,1\}^{a(n_i)}$. An arrow from boxes at length n' to boxes at length n labeled $M_i//b$ denotes that $M//1$ copies $M_i//b$ on the corresponding inputs.

Now, assume that our model of computation has a universal machine U that can be mimicked by a nondeterministic polynomial-time machine N. This is the case, for example, for the model of randomized machines with one-sided error: For a randomized machine M_i, string x, and integer $t \geq 0$, we can let $N(\langle M_i, x, 0^t\rangle)$ check whether there exists a random string that makes M_i accept input x in t steps; whenever M_i satisfies the promise on input x and runs in t steps, $N(\langle M_i, x, 0^t\rangle) = M_i(x)$. Suppose also that L is paddable and NP-complete, as satisfiability is. Then, for some length $\tilde{m}_i$ polynomially related to m_i, there exists an efficient translation of queries to U of length $\tilde{m}_i$ into queries to L of length m_i. Since we can assume that $M//1$ satisfies the promise at length m_i, runs in polynomial time, and agrees with L at length m_i, we obtain an efficient safe simulation T of U at length $\tilde{m}_i$.

An efficient safe simulation of U can be used as a substitute for U in the construction from Section 3.1. In that case, there no longer is a need for advice as each advice bit in that construction indicates whether U satisfies the promise on a certain set of inputs — T satisfies the promise everywhere! As a consequence, we no longer have to use different lengths for all the nodes of the copying tree. We still need to assign $2^{a(\ell)}$ strings of length ℓ to each leaf of length ℓ such that these strings are distinct for all leaves.

Suppose the length we assign to a node only depends on its depth in the tree. As before, let us use the notation $n_{i,j}$ to denote the length corresponding to depth $k_i - j$, $1 \leq j \leq k_i$, with $n_i \doteq n_{i,1}$ and $n_i^* \doteq n_{i,k_i}$. The resulting copying process is illustrated for the case $a(n) = 1$ in Figure 2.

The logarithm of the number of distinct strings of length n_i we need can then be expressed as

$$a(n_i) + \sum_{j=2}^{k_i} a(n_{i,j}) = \sum_{j=1}^{k_i} a(n_{i,j}). \tag{4}$$

The question is how large we can make $a(n)$ such that (4) does not exceed n_i.

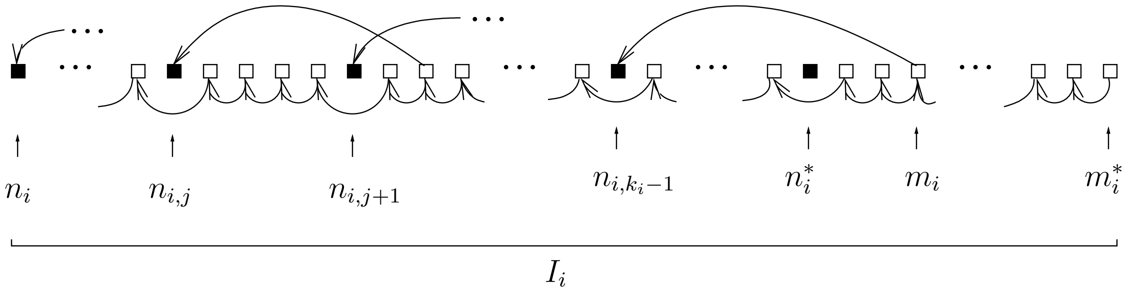

Figure 3: Full construction of M on I_i in Theorem 3, combining simplified copying nodes (black) with efficient safe simulation nodes (white). An arrow from length n' to length n denotes that $M//1$ (above line) or $M_i//b$ (below line) at length n' is used to construct $M//1$ at length n.

If a computation of n^c steps can be complemented within the model in time 2^{n^c} (as in the case of randomized machines with one-sided error), we have the condition $2^{n_i^c} \leq (n_i^*)^{O(1)}$ in order to guarantee that M/α runs in polynomial time at length n_i^*. Since $a(n_i^*) \leq n_i$ follows from our upper bound on (4), we conclude that $a(n_i^*) = O((\log n_i^*)^{1/c})$.

We can actually achieve modulus $a(\log n)^{1/c}$ for any constant a. By setting $n_{i,j+1} = n_{i,j}^d$, $1 \leq j < k_i$, where d is any constant, (4) becomes a linear function in n_i with a coefficient that is a geometric sum $\sum_{j=1}^{k_i} r^j$ and such that the ratio r converges to 0 when d grows. We refer to (8) in Section 5.4 for the details of the computation. By picking d large enough, we can bound (4) by n_i.

We developed our simplification of the copying process from Section 3.1 assuming free access to an efficient safe simulation T of U at all the levels we need it. In reality, we have to build T at all those levels using the recovery procedure. We can apply the recovery as described above to obtain T at length $\tilde{m}_i = n_i^*$ by building L at length m_i. Once we have L at length m_i, we can exploit the paddability properties of L and apply the recovery procedure to obtain L at any smaller length in I_i except the few lengths that are reserved for the simplified copying process. This effectively makes T available at all lengths up to n_i^*, which is (more than) what we need for the simplified copying process. See Figure 3 for an illustration. We refer to Section 5.4 for the remaining details of the proof of Theorem 4.

Finally, we point out that copying using membership proof recovery also leads to a *relativizing* proof of Theorem 3 for the specific case of randomized machines with two-sided error.

Our strong separation result for randomized machines with one-sided error hinges on the efficient simulation of a universal machine by a nondeterministic machine. For randomized machines with two-sided error we do not know an efficient simulation of a universal machine U by a nondeterministic machine but do by a nondeterministic machine N^L with oracle access to satisfiability, L. Moreover, the two-sided error model has the following closure property: Given randomized machines R_i and R'_i that efficiently solve satisfiability with two-sided error on inputs of certain lengths m_i and m'_i, respectively, we can easily construct a randomized machine M that efficiently simulates N^L with two-sided error on inputs of a polynomially related length $\tilde{m}_i$. The property essentially follows from the standard argument that if NP $\subseteq$ BPP then $\Sigma_2^p \subseteq$ BPP. More precisely, suppose wlog. that N^L only makes oracle queries of length exactly m_i on inputs of length $\tilde{m}_i$. Let R''_i be a sufficiently amplified version of R_i such that for most random strings ρ of the appropriate length, $R''_i(q, \rho)$ computes satisfiability correctly for all queries q of length m_i. For any fixed ρ and input x of length $\tilde{m}_i$, $N^{R''_i(\cdot,\rho)}(x)$ defines a nondeterministic computation, which we can efficiently translate into a satisfiability question y, say of length m'_i. Our simulation M of $N^L(x)$ first picks ρ uniformly at random, then computes y, and finally runs R'_i on y.

Combined with copying down satisfiability to lengths m_i and m'_i, we obtain our efficient safe simulation T of U at length $\tilde{m}_i$. By the analysis for the one-sided error case, this leads to a strong separation result with up to $a(\log n)^{1/c}$ bits of advice on the time n^c side. In fact, we can do better by exploiting the efficient closure of the two-sided error model under complementation. We can actually get an efficient safe *complementation* S of U instead of an efficient safe *simulation* T. In that case, we only need to apply S once, namely at the beginning of the interval I_i, and furthermore copy down satisfiability as needed. In terms of Figure 3, there is only one black node, namely at length n_i. The tree in Figure 2 reduces to a single node and (4) to the single term $a(n_i)$. The restriction on the advice length due to the simulation/complementation is relaxed from $a(n) = O((\log n)^{1/c})$ to $a(n) \leq n$. The restriction due to copying down satisfiability remains $a(n) = O(\log n)$.

Note that the above argument does not use instance checkers or any other nonrelativizing ingredient. Thus, we obtain a relativizing proof of Theorem 3 for the case of randomized machines with two-sided error. The argument fails for quantum machines with two-sided error because the simulation of BQP in the polynomial-time hierarchy remains open.

4 Hierarchy Theorem

In this section, we establish our generic hierarchy theorem. We introduce the notion of a semantic model of computation with advice and list the modest properties we need for our hierarchy theorem to apply. We then formally prove Theorems 1 and 2 and state their generalization to superpolynomial time bounds. We refer to Section 3.1 for the intuition behind the proofs.

4.1 Semantic Models

Fix an alphabet Σ containing the symbols 0 and 1. We abstractly view a *model of computation* as consisting of a set $\mathcal{M} \subseteq \Sigma^*$ of "machines" (or "programs"), and a partial computable function $\gamma : \Sigma^* \times \Sigma^* \to \Sigma^*$. For any $M \in \mathcal{M}$ and $x \in \Sigma^*$, $\gamma(M, x)$ determines the output of M on input x (possibly undefined). We also use the shorthand $M(x)$ for $\gamma(M, x)$. A language $L \subseteq \Sigma^*$ is said to be "decided" by M if $M(x) = L(x)$ for each $x \in \Sigma^*$, where $L(x)$ denotes the indicator for the property "$x \in L$", i.e., $L(x) = 1$ if $x \in L$ and $L(x) = 0$ otherwise.

We call a model of computation *syntactic* if $\mathcal{M}$ is computably enumerable. We call the model *semantic* if it is induced by a syntactic model $(\mathcal{M}', \gamma)$ and a predicate $\pi \subseteq \Sigma^* \times \Sigma^*$ such that

$$\mathcal{M} = \{M \in \mathcal{M}' \mid (\forall\, x \in \Sigma^*)\, \pi(M, x)\}.$$

The predicate π can be thought of as a condition on or *promise* about the behavior of M on input x. A machine $M \in \mathcal{M}'$ has to satisfy the promise on each input x in order to fall within the computation model $\mathcal{M}$. Note that we could abstract away the predicate π at this point and just consider the model as defined by $\mathcal{M}$ and γ. However, the predicate π will play a role once we introduce advice. We also note that according to our definition every model of computation is semantic – simply set $\mathcal{M}' = \Sigma^*$ and let $\pi(M, x)$ indicate whether $M \in \mathcal{M}$. However, the resulting promise π may not be "natural." The semantic models of interest are all induced by standard syntactic models and "natural" promises.

Examples of syntactic models include deterministic, nondeterministic, and randomized machines, as well as alternating machines of any fixed signature. Every syntactic model is also "naturally" semantic but not vice versa. For example, randomized machines with two-sided error form a semantic non-syntactic model $\mathcal{M}$. There does not exist a computable enumeration of $\mathcal{M}$ but the model $\mathcal{M}'$ of all randomized machines is syntactic and we can obtain $\mathcal{M}$ as those machines of $\mathcal{M}'$ that satisfy the promise of two-sided error. Other examples of semantic non-syntactic models include randomized machines with one-sided or zero-sided error, quantum machines with two-, one-, or zero-sided error, unambiguous machines, symmetric alternation, Arthur-Merlin games of any signature, etc.

We point out that similar formalizations of the intuitive difference between syntactic and semantic computation have been proposed before in the literature [Pap94, FST05]. However, the earlier attempts all seem to associate these notions with complexity classes rather than models of computation. For example, BPP (the class of languages decidable by polynomial-time randomized machines with two-sided error) is considered a semantic non-syntactic class, whereas P is considered syntactic. This leads to inconsistencies since BPP may coincide with P. Our approach based on machines rather than languages does not suffer from that pitfall.

We assume there is an underlying notion of *time*. Whenever $\gamma(M, x)$ is defined, M halts and produces its output after a finite number of steps, denoted $t_M(x)$. We say that M runs in time t at length n if $t_M(x) \leq t$ for each $x \in \Sigma^n$, and that M runs in time $t(n)$ if M runs in time $t(n)$ at each length n. In the case of semantic models we assume that the notion of running time extends to every machine M in $\mathcal{M}'$, i.e., that the semantic model inherits its notion of time from its syntactic mother model.

An *advice* sequence α of length $a(n)$ is an infinite sequence of strings $\alpha_0, \alpha_1, \alpha_2, \ldots$, one for each length n, such that $|\alpha_n| = a(n)$ for each n. We define the behavior of a machine $M \in \mathcal{M}'$ with advice α, denoted M/α, on a given input x as equal to the behavior of M on input $\langle x, \alpha_{|x|}\rangle$, where $\langle \cdot, \cdot \rangle$ denotes a standard pairing function. In particular, M/α satisfies the promise on input x iff $\pi(M, \langle x, \alpha_{|x|}\rangle)$ holds, and $M/\alpha(x) = M(\langle x, \alpha_{|x|}\rangle)$. Whenever we talk about a property of M/α at length n (like satisfying the promise, running time, etc.), we refer to that property on all inputs of the form $\langle x, \alpha_n\rangle$ where x is a string of length n. Note that the behavior of M/α at length n depends on the component α_n but not on the other components of α. We use the shorthand $M//\alpha_n$ to denote that behavior.

We consider M/α to fall within the model iff $M \in \mathcal{M}'$ and M/α satisfies the promise at each length. We point out that, apart from the predicate π, the choice of the encapsulating syntactic model $\mathcal{M}'$ and the actual advice string α play a role. This differs from the Karp-Lipton notion of computation with advice [KL82], which essentially only considers those machines $M \in \mathcal{M}'$ that robustly satisfy the promise, i.e., the machines in $\mathcal{M}$. More precisely, M/α falls within their model iff M/β falls within our model for each advice sequence β of the same length as α. We point out that under the Karp-Lipton notion of advice, establishing time hierarchies with advice is as hard as without advice. For the notion of advice we use such an equivalence is not known.

4.2 Reasonable Semantic Models

We now introduce the additional requirements a semantic model of computation has to satisfy for our hierarchy theorem to apply. The first one deals with the existence of an efficient universal machine.

Definition 1 *A* universal machine *is a machine $U \in \mathcal{M}'$ such that for each $M \in \mathcal{M}'$, $x \in \Sigma^*$, and $t \geq t_M(x)$, U satisfies the promise on input $\langle M, x, 0^t \rangle$ whenever M satisfies the promise on input x, and if so, $U(\langle M, x, 0^t \rangle) = M(x)$. We call U* efficient *if it runs in polynomial time.*

The second condition states that the model can be complemented within the model in exponential time. We phrase the condition in terms of the universal machine U.

Definition 2 *We say that U can be complemented within the model in exponential time if there exists a machine S that runs in time $2^{n^{O(1)}}$, satisfies the promise on every input, and such that $S(x) = \neg U(x)$ for every input $x \in \Sigma^*$ on which U satisfies the promise.*

The final property states that the model is closed under deterministic transducers. By the latter, we mean deterministic machines D that, on input x, output either an answer $a(x)$, or else a query $q(x)$. Note that a transducer that always outputs an answer is equivalent to a standard Turing machine, and that a transducer that always outputs a query is equivalent to a many-one reduction. For any $M' \in \mathcal{M}'$, we use the following notation:

$$D^{M'}(x) = \begin{cases} a(x) & \text{if } D \text{ outputs an answer on input } x \\ M'(q(x)) & \text{otherwise} \end{cases}$$

and

$$t_{D^{M'}}(x) = \begin{cases} t_D(x) & \text{if } D \text{ outputs an answer on input } x \\ t_D(x) + t_{M'}(q(x)) & \text{otherwise.} \end{cases}$$

We are now ready to formally state the closure property we need.

Definition 3 *A semantic model is* closed under deterministic transducers *if for each deterministic transducer D and each machine $M' \in \mathcal{M}'$, there exists a machine $M \in \mathcal{M}'$ such that the following holds for all inputs x: If $D(x)$ outputs an answer or if M' satisfies the promise on input $q(x)$, then M satisfies the promise on input x, and $M(x) = D^{M'}(x)$. We say that the closure is efficient if M runs in time $t_{D^{M'}}(x)$ on input x.*

Our hierarchy theorem applies to any semantic model with the above three properties.

Definition 4 *A semantic model of computation is called* reasonable *if it has an efficient universal machine that can be complemented deterministically in exponential time and if it is efficiently closed under deterministic transducers.*

All the concrete models mentioned in this paper are reasonable semantic models.

We point out that for the proof of Theorem 1, we can relax the efficiency requirement in Definition 3 to time $(t_{D^{M'}}(x))^{O(1)}$ instead of time $t_{D^{M'}}(x)$. However, for the strong hierarchy of Theorem 2, we seem to need the efficiently requirement as stated in Definition 3.

4.3 Proof of Theorem 1

Assume a reasonable semantic model of computation. Let $\mathcal{M}$ be the set of the machines belonging to the model. This set is contained in some other set $\mathcal{M}'$ that has a computable enumeration $(M_i)_{i=1}^{\infty}$. This reasonable semantic model has an efficient universal machine U which runs in time n^u for some constant u (Definition 1) and has a safe complementation S within the model running in time 2^{n^s} for some constant s (Definition 2). Without loss of generality, we assume that c is a positive integer.

A disjoint interval $I_i = [n_i, n_i^*]$ of input lengths is reserved for every machine M_i. Interval I_i contains the subintervals $I_{i,j} = [l_{i,j}, r_{i,j}]$, $1 \le j \le k_i$, where $l_{i,1} = n_i$, $r_{i,j} < l_{i,j+1}$ and $r_{i,k_i} = n_i^*$. We set

$$r_{i,j} = l_{i,j} + (2^a)^{(k_i - j)} - 1 \qquad\qquad k_i = \lceil \log n_i \rceil .$$

Thus, for every $n \in I_{i,j}$ we have $n = l_{i,j} + \Delta_n$, $0 \le \Delta_n < (2^a)^{(k_i-j)}$. We can think of $I_{i,j}$ as the nodes at level $k_i - j$ of a full 2^a-ary tree with root at n_i^*. Let us fix $l_{i,j}$, $1 \le j \le k_i$, such that

$$l_{i,j} = {n_i}^{d^{(j-1)}} \qquad\qquad d = \max(\lceil 4^{a \cdot c \cdot s} \rceil, 2a).$$

It remains to fix the starting input lengths n_i of the intervals I_i taking the following into account. For any number n we want to efficiently compute a number i such that $n \in I_i$ and the description of the machine M_i that corresponds to interval I_i. Since the enumeration $(M_i)_{i=1}^{\infty}$ of machines in $\mathcal{M}'$ can be very ineffective, we allow the intervals I_i to be sparsely distributed over input lengths, and we let $n_i = \max(n_{i-1}^* + 1, m)$ where m is such that the description of machine M_i is produced after m steps of the enumerating procedure. As for the starting length n_1, some of the inequalities in the proof below require that every input length n of interest (that is, belonging to some interval) is greater than some constant. We choose n_1 larger than all these constants. Notice that now, given a number n, we can compute in linear time the numbers i and j, if any, such that $n \in I_{i,j}$ and produce the description of machine M_i.

To guarantee the disjointness of the subintervals $I_{i,j}$ we need to check that $r_{i,j} < l_{i,j+1}$ for any i and any $1 \le j < k_i$. If n_1 is big enough, we have

$$(2^a)^{(k_i-j)} \le (2^a)^{(k_i-1)} \le (2^a)^{\log n_i} \le {n_i}^a$$
$$r_{i,j} = {n_i}^{d^{(j-1)}} + (2^a)^{(k_i-j)} - 1 < {n_i}^{d^{(j-1)}} + {n_i}^a < {n_i}^{d^j} = l_{i,j+1}.$$

Let $x_{i,b,n} = 10^{n-a-1}b$ where b is some string of length a. This works provided $n_1 > a$ as then any input length $n \in I_{i,j}$ is greater than a.

Given an input x of length n, machine M/α does the following.

1. Compute numbers i and j such that $n \in I_{i,j}$. If no such numbers exist, output 0 and halt.
2. If $j < k_i$ and $\alpha_n = 1$ then
   (a) Compute Δ_n such that $n = l_{i,j} + \Delta_n$.
   (b) Let $n' = l_{i,j+1} + \lfloor \Delta_n / 2^a \rfloor$ and let $b = \Delta_n \bmod 2^a$.
   (c) Call U on $\langle M_i // b, 0^{n'-n}x, 0^{(n')^c} \rangle$.
3. If $j = k_i$ and $x = 0^{n-m}x_{i,b,m}$ for some $m \in I_{i,1}$ then
   (a) Call S on $\langle M_i // b, x_{i,b,m}, 0^{m^c} \rangle$.
4. Output 0.

M uses its advice α_n at length n only if n belongs to some subinterval $I_{i,j}$, $1 \leq j < k_i$. For such an input length n, let $\alpha_n = 1$ if $M_i//b$ satisfies the promise at length n' and runs in time $(n')^c$ (see the above algorithm for definitions of n' and b). Otherwise, let $\alpha_n = 0$.

Let us verify that the resulting machine M and advice α are such that M/α:

(A) falls within the model,

(B) runs in polynomial time, and

(C) disagrees with any M_i/β for any advice sequence β of modulus a for which M_i/β falls within the model and runs in time n^c.

Note that we can translate a query y of length m to S into the query $\langle S, y, 0^t\rangle$ to U with $t = 2^{m^s}$. Using that translation, M becomes a deterministic transducer to machine $U \in \mathcal{M}'$. The possible queries to U occur in steps 2(c) and 3(a) of the algorithm. Step 2(c) makes the query $\langle M_i//b, 0^{n'-n}x, 0^{(n')^c}\rangle$ to U. By the choice of the advice α, that step is only executed if $M_i//b$ satisfies the promise at length n' and runs in time $(n')^c$. As for step 3(a), by Definition 2, S satisfies the promise on every input. It follows from Definition 1 that machine U satisfies the promise on every query the transducer M makes. Thus, by Definition 3, M/α falls within the model.

The length of the query $\langle M_i//b, 0^{n'-n}x, 0^{(n')^c}\rangle$ to U in step 2(c) is polynomial in n since M_i is produced in time linear in n and $n' \leq n^d$. Step 3(a) runs S on input $\langle M_i//b, x_{i,b,m}, 0^{m^c}\rangle$ for some $m \in I_{i,1}$ and is only executed if the input to M is of length n_i^*. If n_1 is big enough, we have

$$|\langle M_i//b, x_{i,b,m}, 0^{m^c}\rangle| \leq 2m^c$$
$$m \leq n_i + (2^a)^{k_i-1} - 1 < n_i + (n_i)^a \leq 2{n_i}^a$$
$$n_i^{2\cdot a\cdot c\cdot s} \leq n_i^{\log d} = d^{\log n_i} \leq d^{k_i} \leq \log(n_i^{d^{k_i}}) = \log((n_i^*)^d),$$

and step 3(a) using the simulation by U takes time at most

$$(2 \cdot 2^{(2m^c)^s})^u \leq 2^{2^{s+1}(2\cdot n_i^a)^{c\cdot s}u} < 2^{n_i^{2a\cdot c\cdot s}} \leq (n_i^*)^d.$$

The efficiency requirement in Definition 3 then implies that M runs in polynomial time.

For property (C), consider an arbitrary machine M_i with an advice sequence β of modulus a such that M_i/β falls within the model and runs in time n^c. Let us assume that M_i/β agrees with M/α at each length. Then we can prove by induction on j from k_i down to 1 that there exists an input length $n \in I_{i,j}$ such that M/α at length n copies M/α at length n_i^*, i.e.,

$$(\forall\, x \in \{0,1\}^n)\ M/\alpha\,(x) = M/\alpha\,(0^{n_i^*-n}x).$$

The case when $j = k_i$ holds trivially. For any $j < k_i$, by the induction hypothesis, there is an input length $n' \in I_{i,j+1}$ such that M/α at length n' copies M/α at length n_i^*. Then consider $n = l_{i,j} + \Delta_{n'} \cdot 2^a + \beta_{n'}$. We have that $n \in I_{i,j}$ and $n' = l_{i,j+1} + \lfloor \Delta_n/2^a \rfloor$. By the specification of M and by the choice of the advice sequence α, M/α at length n copies M/α at length n' and, consequently, copies M/α at length n_i^*.

Hence, for some $n \in I_{i,1}$, M/α at length n copies M/α at length n_i^*. At the same time, M_i/β at length n fails to copy M/α at length n_i^* since $M/\alpha\,(0^{n_i^*-n}\,x_{i,\beta_n,n}) = \neg M_i/\beta\,(x_{i,\beta_n,n})$ whenever M_i/β satisfies the promise at length n and runs in time n^c. Therefore, M_i/β does not agree with M/α at length n, which contradicts our assumption.

This finishes the proof of Theorem 1.

Let us point out that we do not really need the strong form of efficiency stated in Definition 3 for the above proof. The place where it plays a role is in our argument for property (B); requiring the running time of M to be $(t_{D^{M'}}(x))^{O(1)}$ suffices for that argument. For the next proof we do seem to need the stronger notion of efficiency.

4.4 Proof of Theorem 2

The proof of Theorem 2 follows from Theorem 1 by successive padding.

Assume that the statement of the theorem does not hold. Then every language computable in time n^d with a bits of advice is also computable in time n^c with a bits of advice. In what follows, we will employ a translation argument to prove by induction on $k \geq 0$ that every language computable in time $n^{c \cdot r^k}$ with a bits of advice is also computable in time n^c with a bits of advice, where r is some constant larger than 1. Since $c \cdot r^k$ grows unboundedly, we obtain a contradiction to Theorem 1.

We choose r to be a rational in the range $1 < r \leq (d/c)^{1/3}$. The fact that r is rational will make sure that we can perform all the arithmetic needed sufficiently efficiently, e.g., computing $\left\lfloor n^{c \cdot r^k} \right\rfloor$. The upper bound on r guarantees that the cases of the inductive statement with $k \leq 3$ are implied by the assumption we made at the beginning of the proof. This is because $n^{c \cdot r^k} \leq n^d$ for values of $k \leq 3$. Thus, $k \leq 3$ forms the base for the induction.

In order to prove the induction step from k to $k+1$ for $k \geq 3$, consider an arbitrary language L computable in time $n^{c \cdot r^{(k+1)}}$ with a bits of advice. We will prove that L is also computable in time $n^{c \cdot r^k}$ with a bits of advice and thus, by the induction hypothesis, in time n^c with a bits of advice.

The language L is recognized by some machine M with advice α of modulus a such that M/α falls within the model and runs in time $n^{c \cdot r^{(k+1)}}$. In order to apply the speed up provided by the induction hypothesis, we construct the following padded version L' of L:

$$L' = \{0^{\sigma(|x|)-|x|-1}\,1\,x : x \in L\} \quad \text{where} \quad \sigma(n) = \left\lceil n^{r^2} \right\rceil .$$

We claim that we can compute L' in time $m^{c \cdot r^k}$ with a bits of advice on inputs of length m. To see this, let us construct a machine M' with advice α' such that M'/α' recognizes L'. The advice α' is defined so that for any $m = \sigma(n)$ in the range of σ, $\alpha'_m = \alpha'_{\sigma(n)} = \alpha_n$. Given an input y of length m, M' first checks whether y is of the form $y = 0^{\sigma(|x|)-|x|-1}\,1\,x$; if so, it recovers the string x, say of length n. M' then runs $M//\alpha'_m = M//\alpha_n$ on input x. By Definition 3, M'/α' falls within the model and runs in time

$$O(m) + n^{c \cdot r^{k+1}} \leq O(m) + m^{c \cdot r^{k-1}} \leq m^{c \cdot r^k} - m$$

for sufficiently large m. By applying Definition 3 to a deterministic transducer that has the answers for small y's hardwired and queries M' at y otherwise, we can assume without loss of generality that M'/α' runs in time $m^{c \cdot r^k}$ for all lengths m. So, L' is computable in time $m^{c \cdot r^k}$ with a bits of advice and therefore, by the induction hypothesis, also in time m^c with a bits of advice. That is, L' is recognized by some machine N' with advice sequence β' of modulus a such that N'/β' falls within the model and runs in time n^c.

Next, we lift the computation N'/β' for L' up to a computation N/β for L. We define β such that $\beta_n = \beta'_{\sigma(n)}$. Given an input x of length n, N constructs the string $y = 0^{\sigma(n)-n-1}\,1\,x$ and runs

$N'//\beta_n = N'//\beta'_m$ on input y, where $m = |y| = \sigma(n)$. Again, by Definition 3, N/β falls within the model and runs in time

$$O(m) + m^c = O(n^{c \cdot r^2}) \leq n^{c \cdot r^3} \leq n^{c \cdot r^k} - n$$

for sufficiently large n. By a similar application of Definition 3 as above, we can assume without loss of generality that the running time of N/β is bounded by $n^{c \cdot r^k}$ for all lengths n. Applying the inductive hypothesis for k one more time, we have that L can be computed in time n^c with a bits of advice. This finishes the inductive step and thereby the proof of Theorem 2.

We point out that if the efficiency requirement in Definition 3 is relaxed from time $t_{DM'}(x)$ to time $(t_{DM'}(x))^e$ for some constant e, then the above proof yields the statement of Theorem 2 but only for values of $d > e \cdot c$ instead of $d > c$.

4.5 General Time Bounds

Theorems 1 and 2 generalize in a straightforward way to any time bound that satisfies some mild conditions.

Theorem 5 *For any reasonable semantic model of computation, any constant a, and any monotone constructible time bound $t(n)$ such that*

$$t(2^{n^{O(1)}}) \geq 2^{(t(n))^{O(1)}} \text{ and } t(n^{O(1)}) \geq (t(n^{O(1)}))^{O(1)}, \tag{5}$$

there exists a language computable in time $t(n^{O(1)})$ with one bit of advice but not in time $t(n)$ with a bits of advice.

The first part of condition (5) should be read as stating that for every positive constant α there exists a positive constant β such that $t(2^{n^\beta}) \geq 2^{(t(n))^\alpha}$. Similarly, the second part of condition (5) states that for every positive constant α there exists a positive constant β such that $t(n^\beta) \geq (t(n^\alpha))^\alpha$.

Theorem 6 *For any reasonable semantic model of computation, any constant $\delta > 1$, and any monotone constructible time bound $t(n)$ satisfying (5), there exists a language computable in time $t(n^\delta)$ with a bits of advice but not in time $t(n)$ with a bits of advice.*

The instantiation of the above two theorems for time bounds of the form $t(n) = n^c$ yields Theorems 1 and 2, respectively. The instantation for time bounds of the form $t(n) = 2^{n^c}$ gives the corresponding results at the exponential level.

5 Separation Theorems

In this section, we establish our separation theorems. We review some preliminaries about instance checkers and introduce the notion of a randomized semantic model of computation with advice. We specify the properties we need for our generic separation theorem (Theorem 3) to apply and then formally prove it. Finally, we establish our separation theorem for randomized machines with one-sided error (Theorem 4). We refer to Section 3.2 for the intuition behind the proofs.

5.1 Instance Checkers and Randomized Semantic Models

An instance checker for a language L is a polynomial-time randomized oracle machine C that can output 0, 1, or "I don't know" such that for any $x \in \Sigma^*$:

(Completeness) $\Pr[C^L(x) = L(x)] = 1$.

(Soundness) For any oracle P, $\Pr[C^P(x) = \overline{L(x)}] \leq 2^{-|x|}$.

We will use an instance checker for a language L with the properties given by the following lemma.

Lemma 1 *There exists a paddable language L that is complete for exponential time and has an instance checker C such that all queries C makes on inputs of length n are of length $f \cdot n$ for some constant $f \geq 1$.*

Proof Sketch. The proof follows from the probabilistically checkable proofs for deterministic time t that are computable in time $t^{O(1)}$ [AS98] and their connection to instance checkers [BK89]. See [Bar02] for more details. □

We do not know whether Lemma 1 holds with the additional restriction that $f = 1$. If so, the formal proof of Theorem 3 in Section 5 can be somewhat simplified.

We will actually run the instance checker of Lemma 1 with an "oracle" P that isn't a fixed language but rather the outcome of a randomized process, i.e., $P(y)$ is the outcome of a 0/1 coin flip with a bias depending on y. By a standard averaging argument[4], the soundness property of the instance checker also holds for such "oracles" P. Perfect completeness typically does not carry over to this setting. However, provided P has bounded error, we can bring the completeness to level $1 - 2^{-|x|}$ or higher by answering each query as the majority vote of a linear number of independent runs.

In order to apply an instance checker C in a semantic model of computation $(\mathcal{M}', \gamma, \pi)$, we need to augment the notion we introduced in Section 4.1. Intuitively, we would like to run C with an "oracle" P that is the result of running a machine $M' \in \mathcal{M}'$ on the queries y of the instance checker. For that to make sense and interact well with the properties of the instance checker, we need to associate a random variable $\underline{M}'(y)$ with the behavior of M' on input y. We call a model equipped with such random variables a *randomized model.* Natural examples for $\underline{M}'(y)$ include the acceptance indicator for randomized machines or for Arthur-Merlin games under an optimal strategy for Merlin.

5.2 Reasonable Randomized Semantic Models

Once we have such an underlying random process, for any randomized oracle machine D, we can define the random variable $D^{M'}(x)$ as the outcome of a run of D where each query y is answered according to a sample of $\underline{M}'(y)$. We require that we can efficiently simulate such a process in our model of computation and that the simulation be sound whenever $D^{M'}$ has two-sided error on input x. More precisely, we stipulate the following.

Definition 5 *A randomized semantic model of computation is* closed under randomized reductions with two-sided error *if for every randomized oracle machine D and every machine $M' \in \mathcal{M}'$, there exists a machine $M \in \mathcal{M}'$ such that the following holds for any string x: If $D^{M'}$ has two-sided error*

[4] W.l.o.g., we are assuming that no run of the instance checker C makes the same query more than once.

on input x, then M satisfies the promise on input x and $M(x)$ equals the majority outcome of $D^{M'}$ on input x. We say that the closure is efficient if M runs in time $(t_D(n) \cdot \max_{0 \leq m \leq t_D(n)} t_{M'}(m))^{O(1)}$.

Another condition we need is that the model has an efficient universal machine U (see Definition 1) which can be simulated deterministically in exponential time.

Definition 6 *We say that U can be deterministically simulated in exponential time if there exists a deterministic machine T which runs in time $2^{n^{O(1)}}$ and such that $T(x) = U(x)$ for each $x \in \Sigma^*$ on which U satisfies the promise.*

Our generic separation theorem applies to any reasonable randomized semantic model defined as follows.

Definition 7 *A randomized semantic model of computation is called* reasonable *if it has an efficient universal machine that can be simulated deterministically in exponential time and if it is efficiently closed under randomized reductions with two-sided error.*

Reasonable randomized semantic models include randomized and quantum machines with two-sided error.

5.3 Proof of Theorem 3

Assume a reasonable randomized semantic model of computation. Let $\mathcal{M}$ be the set of the machines belonging to the model. This set is contained in some other set $\mathcal{M}'$ that has a computable enumeration $(M_i)_{i=1}^{\infty}$. Let U denote an efficient universal machine for the model, as in Definition 1. Let the modulus $a(n)$ be $a \log n$ for some constant a. Without loss of generality, we assume that a and c are positive integers.

Let L be a paddable language that is complete for deterministic exponential time and has an instance checker C such that all queries C makes on inputs of length m are of length $f \cdot m$ for some constant $f \geq 1$. Such a language exists by virtue of Lemma 1.

We use the instance checker C to construct an efficient randomized reduction D to U that will allow M/α to copy L at length m provided M_i/β "appropriately" copies L at length $f \cdot m$ for some advice sequence β of modulus $a(n)$. The critical point is that D is not given access to β.

Lemma 2 *There exists a polynomial-time randomized reduction D with the following property for any integers $m \geq 0$ and $n \geq f \cdot m$ and any machine M_i: If there exists a string $b \in \{0,1\}^{a(n)}$ such that $M_i//b$ satisfies the promise at length n and runs in time n^c, and L at length $f \cdot m$ is a copy of $M_i//b$ at length n, i.e.,*

$$(\forall\, z \in \{0,1\}^{f \cdot m})\; M_i//b\,(0^{n-f\cdot m} z) = L(z),$$

then

$$(\forall\, y \in \{0,1\}^{m})\; \Pr[D^U(\langle M_i, y, 0^n \rangle) = L(y)] > 1 - 2^{-n}.$$

Proof. The idea is to execute M_i at length n with any possible advice string b of length $a(n)$ and to use the instance checker C to verify the answers. By making modifications to C as discussed after Lemma 1, we obtain a polynomial-time randomized oracle machine $C'(\cdot, \cdot)$ which outputs 0, 1, or "I don't know" and such that for any y of length m, $C'(y, 0^n)$ only makes queries of length $f \cdot m$ and the following holds for any $n \geq f \cdot m$:

(Completeness) For any randomized oracle P which solves L with two-sided error,

$$\Pr[C'^P(y, 0^n) = L(y)] > 1 - \frac{1}{n^a \cdot 2^n}.$$

(Soundness) For any randomized oracle P,

$$\Pr[C'^P(y, 0^n) = \neg L(y)] < \frac{1}{n^a \cdot 2^n}.$$

Given input $\langle M_i, y, 0^n \rangle$, reduction D^U does the following:

1. For any advice string b of length $a(n)$
   (a) Simulate $C'(y, 0^n)$ and when C makes a query z, answer it with $U(\langle M_i//b, 0^{n-|z|}z, 0^{n^c}\rangle)$.
   (b) If $C'(y, 0^n) \in \{0, 1\}$, output that value and halt.
2. Output 0.

The number of advice strings we try is $2^{a(n)} = n^a$. Instance checker C runs in time polynomial in n and the length of y, and every query z which C asks is transformed into a query to U in polynomial time. Therefore, procedure D runs in polynomial time.

Let b^* be the value of b given in the statement of the lemma. When trying the value $b = b^*$ in the algorithm, D^U accepts the answer of $M_i//b$ with probability at least $1 - (n^{-a} \cdot 2^{-n})$. On the other hand, the probability that D^U accepts an incorrect answer of $M_i//b$ when using some $b \neq b^*$, is less than $(n^a - 1) \cdot (n^{-a} \cdot 2^{-n})$. Therefore, D^U succeeds and outputs $L(y)$ with probability more than $1 - 2^{-n}$. □

Now we turn to the construction of M/α. We reserve a disjoint interval $I_i = [\tilde{n}_i, n_i^*]$ of input lengths for every machine M_i, and will construct a machine M and advice α of modulus 1 such that M/α falls within the model, runs in polynomial time and has the following property: For any advice sequence β of modulus $a(n)$ for which M_i/β falls within the model, runs in time n^c and agrees with M/α at every length $n \in I_i \setminus \{\tilde{n}_i\}$, M/α disagrees with M_i/β at length $\tilde{n}_i$. With that goal in mind, for any string b of length $a(\tilde{n}_i)$, we let M/α on input $x_{i,b} = 0^{\tilde{n}_i - a(\tilde{n}_i)}b$ compute and output $\neg M_i//b\,(x_{i,b})$. Evidently, M/α disagrees with M_i/β at length $\tilde{n}_i$, because for $b = \beta_{\tilde{n}_i}$, we have $M/\alpha\,(x_{i,b}) = \neg M_i/\beta\,(x_{i,b})$. The only hurdle on the way to the time hierarchy is that the computation of $M_i//b\,(x_{i,b})$ must be done in time polynomial in $\tilde{n}_i$.

The value of $M_i//b\,(x_{i,b})$ can be computed by means of the universal machine as $U(q_{i,b})$ where $q_{i,b} = \langle M_i//b, x_{i,b}, 0^{\tilde{n}_i^c}\rangle$. By Definition 6, there exists a deterministic procedure T that simulates the universal machine U in exponential time, and therefore also a deterministic exponential-time machine S that simulates $\neg U$. Since L is complete for exponential time, there is a polynomial-time many-one reduction R that translates every query $\neg U(q_{i,b})$ into some query $L(z_{i,b})$. Since L is paddable, we can set up R to produce queries $z_{i,b}$ of length exactly $n_i = (|M_i| \cdot \tilde{n}_i)^p$ for some constant $p > 1$. It remains to make M/α efficiently compute L at length n_i.

As an exponential-time language, L is computable in time 2^{n^e} on a deterministic machine for some constant e. Let each interval I_i contain input lengths $n_{i,j}$, $1 \leq j \leq k_i$, such that

$$n_{i,j+1} = (n_{i,j})^d \qquad d = (2f)^e \qquad k_i = \lceil \log n_i \rceil.$$

Let $n_i \doteq n_{i,1}$ and $n_i^* \doteq n_{i,k_i}$. These settings enable M/α at length n_i^* to solve L at length $f^{k_i-1} \cdot n_i$ in time polynomial in the input length n_i^*, since

$$(f^{k_i-1} \cdot n_i)^e \leq (f^{\log n_i} \cdot n_i)^e = n_i^{(1+\log f)\cdot e} = n_i^{\log d} = d^{\log n_i} \leq d^{k_i} = \log((n_i^*)^d)$$
$$2^{(f^{k_i-1} \cdot n_i)^e} \leq (n_i^*)^d.$$

It remains to set the boundaries of the intervals $I_i = [\tilde{n}_i, n_i^*]$. As in the proof of Theorem 1, we make the choice of the starting points $\tilde{n}_i$ so that the description of machine M_i can be produced in time linear in $\tilde{n}_i$. The first length $\tilde{n}_1$ is chosen large enough for every inequality in this proof to hold.

Finally, we construct machine M with advice α. Given an input x of length n, M/α does the following.

1. Compute numbers i and j such that $n = n_{i,j}$ or $n = \tilde{n}_i$. If no such numbers exist, output 0 and halt.
2. If $n = \tilde{n}_i$ for some i, x is of the form $0^{\tilde{n}_i - a(\tilde{n}_i)}b$, and $\alpha_n = 1$ then
   (a) Let $q_{i,b} = \langle M_i//b, 0^{\tilde{n}_i - a(\tilde{n}_i)}b,\ 0^{(\tilde{n}_i)^c}\rangle$.
   (b) Compute $z_{i,b} = R(q_{i,b})$ of length n_i.
   (c) Call $M//1$ on input $z_{i,b}$.
3. If $n = n_{i,j}$ for some i and j, and x is of the form $0^{n-f^{j-1}\cdot n_i}y$ then
   (a) If $n = n_i^*$ then
      i. Evaluate $L(y)$ deterministically, output the result, and halt.
   (b) If $\alpha_n = 1$ then
      i. Call D^U on input $\langle M_i, y, 0^{n_{i,j+1}}\rangle$.
4. Output 0.

For any $n = n_{i,j}$, $1 \leq j < k_i$, let $\alpha_n = 1$ iff for some advice sequence β of modulus $a(n)$, M_i/β satisfies the promise at length $n_{i,j+1}$ and runs in time $n_{i,j+1}^c$ and if L at length $f^j \cdot n_i$ is a copy of M_i/β at length $n_{i,j+1}$. For $n = \tilde{n}_i$, let $\alpha_n = 1$ iff $M//1$ agrees with L at length n_i. Let $\alpha_n = 0$ in all other cases.

Let us verify that the machine M and advice sequence α we constructed are such that M/α:

(A) falls within the model,

(B) runs in polynomial time, and

(C) disagrees with any M_i/β for any advice sequence β of modulus $a(n)$ for which M_i/β falls within the model and runs in time n^c.

Note that step 2(c) can be expanded into step 3(b)i at length n_i. After doing that, machine M/α becomes a randomized reduction to machine U. In order to prove that M/α falls within the model and runs in polynomial time, we use the closure of the model under randomized reductions with two-sided error (Definition 5). We consider several cases.

In all cases where M/α acts deterministically and does not call U, it follows from Definition 5 that M/α satisfies the promise since deterministic behavior trivially satisfies the condition of two-sided error. In all these cases, M/α also runs in polynomial time. For step 3(a)i, this is because M/α can deterministically solve $f^{k_i-1}\cdot n_i$-long instances of L in time polynomial in n_i^*, as we argued above.

There are two remaining cases. In step 3(b)i at length $n_{i,j}$, $1 \le j < k_i$, M/α invokes D^U on input $\langle M_i, y, 0^{n_{i,j+1}}\rangle$ with $|y| = f^{j-1}\cdot n_i$. This only happens if $\alpha_{n_{i,j}} = 1$, i.e., if for some advice sequence β, M_i/β satisfies the promise at length $n_{i,j+1}$ and runs in time $n_{i,j+1}^c$ and if L at length $f^j \cdot n_i$ is a copy of M_i/β at length $n_{i,j+1}$. In that case, Lemma 2 guarantees that D^U satisfies the condition of two-sided error. By Definition 5, M/α then satisfies the promise. Also, M/α runs in polynomial time since $n_{i,j+1} = (n_{i,j})^d$.

Finally, in step 2(c) at length $\tilde{n}_i$, M/α invokes itself on an input of length n_i. Both the promise and the running time then follow from the previous case.

Now assume that M_i/β falls within the model, runs in time n^c, and agrees with M/α for some advice sequence β of modulus $a(n)$. Then for each $1 \le j \le k_i$, L at length $f^{j-1}\cdot n_i$ is a copy of M/α at length $n_{i,j}$, i.e.,

$$(\forall\, y \in \{0,1\}^{f^{j-1}\cdot n_i})\; M/\alpha\,(0^{n_{i,j}-|y|}y) = L(y). \tag{6}$$

This can be proved by induction on j from k_i down to 1. At length $n_{i,k_i} = n_i^*$, the construction of M/α explicitly satisfies (6). Let us prove the transition from $j+1$ to j. By the induction hypothesis, as M_i/β agrees with M/α, L at length $f^j\cdot n_i$ is a copy of M_i/β at length $n_{i,j+1}$. Thus, $\alpha_{n_{i,j}} = 1$ and at length $n_{i,j}$, M/α executes step 3(b)i. By Lemma 2, this implies that L at length $f^{j-1}\cdot n_i$ is a copy of M/α at length $n_{i,j}$.

For $j = 1$, we obtain that M/α agrees with L at length $n_i = n_{i,1}$. Thus, $\alpha_{\tilde{n}_i} = 1$ and, on input $x_{i,b}$ of length $\tilde{n}_i$, M/α computes $L(z_{i,b}) = \neg M_i/\beta\,(x_{i,b})$. Consequently, M/α disagrees with M_i/β at length $\tilde{n}_i$, which contradicts our hypothesis.

5.4 Proof of Theorem 4

Before presenting the proof of Theorem 4, let us recall the techniques used in the proofs of Theorems 1 and 3. The main idea of the separation result for semantic models with two-sided error (Theorem 3) is to copy an exponential-time complete language using an instance checker. This allows M/α to compute M_i/β efficiently (under the assumption that M_i/β agrees with M/α) and "do the opposite" while always keeping the promise. The main idea of the hierarchy for general semantic models (Theorem 1) is quite different. Informally speaking, it is a tree-like delayed diagonalization.

Returning to randomized computations with one-sided error, we face the fact that we do not know how to "do the opposite" efficiently. However, computations with one-sided error have some nice properties, which we want to employ to construct the machine M/α that disagrees with M_i/β for any advice sequence β of super-constant modulus. As we mentioned earlier, tree-like delayed diagonalization fails in the case of super-constant modulus. So, both techniques we used before individually fail for randomized computations with one-sided error.

The solution is to combine both techniques, namely to use delayed diagonalization, which enables M/α to "do the opposite," and to employ copying using membership proofs, which allows M/α to simulate M/β efficiently for every step of the delayed diagonalization.

Let $(M_i)_{i=1}^{\infty}$ denote a standard enumeration of all randomized machines and let $a(n) = a \log n$ where a is a constant. Without loss of generality, we assume that a and c are positive integers.

We let L denote a nicely paddable version of the satisfiability problem, namely the language of all strings of the form $0^{\ell}1\phi$, where ℓ is any nonnegative integer and ϕ is a satisfiable propositional formula. Note that if M/α at length n copies L at length $n' > n$ then M/α actually computes L at length n. Our definition of L also allows us to assume that substituting a logical value for a variable of a formula does not change its length.

We start by constructing a randomized machine D that will help M/α to copy NP-complete behavior. Note that the machine D in the next lemma does not receive any advice.

Lemma 3 *There exists a randomized polynomial-time machine D with the following property for any integers $n, m \geq 0$, randomized machine M_i and any $b \in \{0,1\}^{a(n)}$: If $M_i//b$ satisfies the promise at length n and runs in time n^c and if L at length m is a copy of $M_i//b$ at length n, i.e.,*

$$(\forall \psi \in \{0,1\}^m)\ M_i//b\,(0^{n-m}\psi) = L(\psi),$$

then

$$(\forall \phi \in \{0,1\}^m) \begin{cases} \Pr[D(\langle M_i,\, \phi,\, 0^n \rangle) = 1] > 1/2 & \text{if } \phi \in L \\ \Pr[D(\langle M_i,\, \phi,\, 0^n \rangle) = 1] = 0 & \text{if } \phi \notin L. \end{cases}$$

Proof. Machine D simply tries all possible advice strings b of length $a(n)$ and employs the self-reducibility of L to check the answer of $M_i//b$. Let $\phi|_{x_i=v}$ denote the substitution of the logical value v for variable x_i in formula $\phi(x_1, \ldots, x_k)$. On input $\langle M_i,\, \phi,\, 0^n \rangle$, D acts as follows.

1. Let k be the number of variables in $\phi(x_1, \ldots, x_k)$.
2. For each $b \in \{0,1\}^{a(n)}$ do
   (a) Let $\phi_0 = \phi$.
   (b) For each j from 1 to k do
      i. For $\lceil \log 2n \rceil$ times, simulate $M_i//b\,(0^{n-|\phi|}\phi_{j-1}|_{x_j=0})$ for n^c steps.
      ii. If at least one of the answers is 1, then let $\phi_j = \phi_{j-1}|_{x_j=0}$.
      iii. Otherwise, let $\phi_j = \phi_{j-1}|_{x_j=1}$.
   (c) If ϕ evaluates to 1, then output 1 and halt.
3. Output 0.

Basically, machine D executes a polynomial number of self-reductions, each time using a polynomial number of runs of M_i to decide whether to substitute a value 0 or 1 for variable x_j in the formula. Note that the simulation of n^c steps of a randomized machine M_i with one-sided error can be accomplished in time polynomial in n and $|M_i|$. Thus, D runs in time polynomial in its input length.

Let b^* denote the value of b given in the statement of the lemma. Consider the iteration of the above algorithm with $b = b^*$. In step 2(b)i, if $\phi_{j-1}|_{x_j=0}$ is satisfiable, D discovers this with probability greater than $1 - (1/2)^{\log 2n} = 1 - 1/2n$ and sets $\phi_j = \phi_{j-1}|_{x_j=0}$. On the other hand, if $\phi_{j-1}|_{x_j=0}$ is not satisfiable, then D always sets $\phi_j = \phi_{j-1}|_{x_j=1}$. As a result, provided ϕ_{j-1} is satisfiable, D produces a satisfiable ϕ_j with probability greater than $1 - 1/2n$. Therefore, for any satisfiable formula ϕ, machine D produces a satisfied ϕ_k and outputs 1 when trying $b = b^*$ with probability greater than $1 - k \cdot 1/2n \geq 1/2$. Given an unsatisfiable ϕ, D always produces an unsatisfied ϕ_k and therefore always outputs 0. $\square$

Now we turn to the construction of machine M/α that witnesses the statement of Theorem 4 for the given values of a and c. We reserve a disjoint interval $I_i = [n_i, m_i^*]$ of input lengths for each machine M_i. We will implement the following scheme of delayed diagonalization against M_i/β:

$$\begin{aligned} M(x_{i,\beta})/\alpha = M_i(0^{n_{i,2}-n_i}x_{i,\beta})/\beta = M(0^{n_{i,2}-n_i}x_{i,\beta})/\alpha = M_i(0^{n_{i,3}-n_i}x_{i,\beta})/\beta = \dots \\ \dots = M(0^{n_{i,k_i-1}-n_i}x_{i,\beta})/\alpha = M_i(0^{n_i^*-n_i}x_{i,\beta})/\beta = M(0^{n_i^*-n_i}x_{i,\beta})/\alpha = \neg M_i(x_{i,\beta})/\beta. \quad (7) \end{aligned}$$

Let us define the strings $x_{i,\beta}$ on which M/α diagonalizes against M_i/β as follows:

$$x_{i,\beta} = 0^s \circ \beta_{n_{i,1}} \circ \dots \circ \beta_{n_{i,k_i}},$$

where s is such that $|x_{i,\beta}| = n_i$.

We set the input lengths $n_{i,j}$ at which the delayed diagonalization is realized as follows:

$$n_{i,j+1} = (n_{i,j})^d \qquad d = (a+1)^c \qquad k_i = \min\{k : n_{i,k}^d \geq 2^{n_i^c}\},$$

where $n_i \doteq n_{i,1}$ and $n_i^* \doteq n_{i,k_i}$ as usual.

By the choice of k_i and $n_{i,j}$, M/α at length n_i^* is able to simulate n_i^c steps of $\neg M_i/\beta$ at length n_i in time polynomial in n_i^*. Also, we have enough strings at length n_i to represent all the behaviors of M_i/β on input lengths $n_{i,j}$, $1 \leq j \leq k_i$, because

$$n_i^* \leq 2^{n_i^c} \qquad n_{i,k_i-j} \leq (2^{n_i^c})^{d^{-j}}$$

$$\sum_{j=1}^{k_i} a(n_{i,j}) = \sum_{j=1}^{k_i} a \cdot (\log n_{i,j})^{1/c} \leq \sum_{j=0}^{k_i-1} a \cdot n_i \cdot d^{-j/c} < a \cdot n_i \cdot \frac{1}{d^{1/c}-1} = n_i. \quad (8)$$

Therefore, the strings $x_{i,\beta}$ are well-defined.

As mentioned in Section 3.2.2, there exists a universal machine U that can be mimicked by a nondeterministic polynomial-time machine N. Every query to U can therefore be translated in polynomial time into a query to the NP-complete language L. We denote this polynomial-time reduction by R. If M/α can solve L efficiently at the lengths we need, it can also efficiently compute $U(\langle M_i//\beta_{n_i}, 0^{n_{i,j}-n_i}x_{i,\beta}, 0^{(n_{i,j})^c}\rangle)$ and execute the delayed diagonalization scheme (7) in polynomial time.

The paddability properties of L and the polynomial running time of R allow us to assume without loss of generality that $|R(\langle M_i//\beta_{n_i}, 0^{n_{i,j}-n_i}x_{i,\beta}, 0^{n_{i,j}^c}\rangle)| = (|M_i| \cdot n_{i,j})^r$ for some constant r. Thus, the maximum length of an instance of L that M/α needs to evaluate when diagonalizing against machine M_i is $m_i = (|M_i| \cdot n_i^*)^r$. So we let $m_i^* = 2^{m_i}$, and devote all lengths in $I_i = [n_i, m_i^*]$ except those that are already used for the delayed diagonalization, for copying L. The goal is that for each $n \in I_i \setminus \{n_{i,j}\}_{j=1}^{k_i}$, L at length $\min(m_i, n)$ is a copy of M/α at length n. In particular, M/α computes L at any such $n \leq m_i$.

Given an input x of length n, M/α does the following:

1. If $n = n_{i,j}$ for some i and j, then

(a) If $j = k_i$ and x is of the form $0^{n_i^*-n_i}x_{i,\beta}$ then   *% complement M*

i. Extract β_{n_i} from $x_{i,\beta}$.

ii. Deterministically simulate $M_i//\beta_{n_i}(x_{i,\beta})$ for n_i^c steps, output the complement and halt.

(b) Else if $j < k_i$, x is of the form $0^{n-n_i}x_{i,\beta}$, and $\alpha_n = 1$ then   *% copy M*

i. Let $n' = n_{i,j+1}$.

ii. Extract $b = \beta_{n'}$ from $x_{i,\beta}$.

iii. Compute $\phi = R(\langle M_i, 0^{n'-n_i}x_{i,\beta}, 0^{(n')^c}\rangle)$.

iv. Let $m' = |\phi| + 1$ if $|\phi| + 1 \notin \{n_{i,\ell}\}_{\ell=1}^{k_i}$; otherwise, let $m' = |\phi| + 2$.

v. Call D on input $\langle M_i,\, \phi,\, 0^{m'}\rangle$.

2. If $n \in I_i \setminus \{n_{i,j}\}_{j=1}^{k_i}$ then

(a) If $n = m_i^*$ and x is of the form $0^{m_i^*-m_i}\phi$ then   *% compute L*

i. Deterministically compute $L(\phi)$, output that value and halt.

(b) If $n < m_i^*$, x is of the form $0^{n-\min(m_i,\,n)}\phi$, and $\alpha_n = 1$, then   *% copy L*

i. Let $m'' = n + 1$ if $n + 1 \notin \{n_{i,\ell}\}_{\ell=1}^{k_i}$; otherwise, let $m'' = n + 2$.

ii. Call D on input $\langle M_i,\, \phi,\, 0^{m''}\rangle$.

3. Output 0.

Notice that m' and m'' lie in $I_i \setminus \{n_{i,j}\}_{j=1}^{k_i}$.

For any $n = n_{i,j}$, $1 \le j < k_i$, let $\alpha_n = 1$ iff for some advice sequence β of modulus $a(n)$, M_i/β satisfies the promise at length m' (see the algorithm above), runs in time $(m')^c$, and computes L at length m'. For any $n \in I_i \setminus \{n_{i,j}\}_{j=1}^{k_i} \setminus \{m_i^*\}$, let $\alpha_n = 1$ iff for some advice sequence β of modulus $a(n)$, M_i/β satisfies the promise at length m'' (see the algorithm above), runs in time $(m'')^c$, and L at length $\min(m_i, m'')$ is a copy of M_i/β at length m''. In all other cases, let $\alpha_n = 0$.

We defined the advice α in such a way that whenever D is called on some input, Lemma 3 guarantees that D satisfies the condition of one-sided error on that input. Since M/α acts deterministically in all other cases, it follows that M/α falls within the model independently of the behavior of machine M_i/β.

As to time requirements for M/α, note that m' and m'', the lengths for which the polynomial-time procedure D is invoked, are polynomial in the input length n. Also, M/α at length n_i^* simulates M_i/β at length n_i in time polynomial in n_i^*. M/α at length $m_i^* = 2^{m_i}$ computes L at length m_i in time polynomial in m_i^*. Therefore, M/α runs in polynomial time on every input.

Now assume that M_i/β falls within the model, runs in time n^c, and agrees with M/α for some advice sequence β of modulus $a(n)$. In that case, by induction on n from m_i^* down to n_i, we can show that L at length $\min(m_i, n)$ is a copy of M/α at length n for each n in $I_i \setminus \{n_{i,j}\}_{j=1}^{k_i}$. In particular, at all lengths m' in the above algorithm, M/α computes L. This enables M/α to implement the delayed diagonalization scheme (7), which contradicts that M/α and M_i/β agree on $x_{i,\beta}$. This finishes the proof of Theorem 4.

6 Further Research

In this paper, we established a hierarchy theorem that applies to any "reasonable" semantic model of computation with one bit of advice (Theorems 1 and 2). The most pertinent open problem is to eliminate the need for the one bit of advice. Ideally, we would like to do that without further restricting the meaning of "reasonable" but the question remains open for any semantic model which is not known to be equivalent in power to a syntactic one.

For randomized machines with two-sided error, the question whether a hierarchy theorem would require nonrelativizing techniques is still up for debate [FS89, FS97, RV01]. Prior to our work, a hierarchy theorem with one bit of advice was established using nonrelativizing techniques. Our proof shows that the result itself as well as the strong separation stated in Theorem 3 do relativize for this specific model. Whether our generic separation theorem (Theorem 3) relativizes for every model remains open.

Improving the advice bound in our separation results (Theorems 3 and 4) forms another possible direction for further research. As for Theorem 4, one can abstract the properties the model needs for our proof to carry through, just as we did for our other arguments. We refrained from stating Theorem 4 in such generality because randomized machines with one-sided error are the only interesting application we could think of. Are there others?

Finally, one can ask about strong hierarchy theorems, in which the more restricted machines fail to decide the hard language for almost all input lengths (instead of just one or infinitely many). Even with advice, no such hierarchy theorems are known for a non-syntactic model.

Acknowledgments

We thank Dimitri Grigoriev, Edward A. Hirsch, Jeff Kinne, Rahul Santhanam, Madhu Sudan, Thomas Watson, and the anonymous reviewers for discussions and/or helpful comments on an earlier version of the paper.

References

[AS98] S. Arora and S. Safra. Probabilistic checking of proofs: a new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998.

[Bar02] B. Barak. A probabilistic time hierarchy theorem for slightly nonuniform algorithms. In *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques in Computer Science*, volume 2483 of *Lecture Notes in Computer Science*, pages 194–208. Springer-Verlag, 2002.

[BK89] M. Blum and S. Kannan. Designing programs that check their work. In *Proceedings of the 21st ACM Symposium on the Theory of Computing*, pages 86–97. ACM, 1989.

[Coo73] S. Cook. A hierarchy theorem for nondeterministic time complexity. *Journal of Computer and System Sciences*, 7:343–353, 1973.

[FS89] L. Fortnow and M. Sipser. Probabilistic computation and linear time. In *Proceedings of the 21st ACM Symposium on the Theory of Computing*, pages 148–156. ACM, 1989.

[FS97] L. Fortnow and M. Sipser. Retraction of "Probabilistic computation and linear time". In *Proceedings of the 29th ACM Symposium on the Theory of Computing*, page 750. ACM, 1997.

[FS04] L. Fortnow and R. Santhanam. Hierarchy theorems for probabilistic polynomial time. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science*, pages 316–324. IEEE, 2004.

[FST05] L. Fortnow, R. Santhanam, and L. Trevisan. Hierarchies for semantic classes. In *Proceedings of the 37th ACM Symposium on the Theory of Computing*, pages 348–355. ACM, 2005.

[GHP05] D. Grigoriev, E. Hirsch, and K. Pervyshev. Time hierarchies for cryptographic function inversion with advice. Technical Report TR-05-076, Electronic Colloquium on Computational Complexity, 2005.

[GST04] O. Goldreich, M. Sudan, and L. Trevisan. From logarithmic advice to single-bit advice. Technical Report TR-04-093, Electronic Colloquium on Computational Complexity, 2004.

[HS65] J. Hartmanis and R. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[KL82] R. Karp and R. Lipton. Turing machines that take advice. *L'Enseignement Mathématique*, 28(2):191–209, 1982. A preliminary version appeared in STOC 1980.

[Lev73] L. Levin. Universal search problems. *Problems of Information Transmission*, 9(3):265–266, 1973. In Russian.

[Pap94] C. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48:498–532, 1994.

[Per05] K. Pervyshev. Time hierarchies for computations with a bit of advice. Technical Report TR-05-054, Electronic Colloquium on Computational Complexity, 2005.

[RV01] R. Rettinger and R. Verbeek. Monte-Carlo polynomial versus linear time - the truth-table case. In *Proceedings of the 13th International Sympsoium on Fundamentals of Computation Theory*, pages 311–322. Springer-Verlag, 2001.

[SFM78] J. Seiferas, M. Fischer, and A. Meyer. Separating nondeterministic time complexity classes. *Journal of the ACM*, 25:146–167, 1978.

[Ž83] S. Žàk. A Turing machine time hierarchy. *Theoretical Computer Science*, 26:327–333, 1983.