Electronic Colloquium on Computational Complexity, Report No. 134 (2007)

Space Hierarchy Results for Randomized and Other Semantic Models

Jeff Kinne* Dieter van Melkebeek†

December 14, 2007

Abstract

We prove space hierarchy and separation results for randomized and other semantic models of computation with advice. Previous works on hierarchy and separation theorems for such models focused on time as the resource. We obtain tighter results with space as the resource. Our main theorems are the following. Let $s(n)$ be any space-constructible function that is $\Omega(\log n)$ and such that $s(an) = O(s(n))$ for all constants a, and let $s'(n)$ be any function that is $\omega(s(n))$.

> There exists a language computable by *two-sided* error randomized machines using $s'(n)$ space and one bit of advice that is not computable by *two-sided* error randomized machines using $s(n)$ space and $\min(s(n), n)$ bits of advice.
>
> There exists a language computable by *zero-sided* error randomized machines in space $s'(n)$ with one bit of advice that is not computable by *one-sided* error randomized machines using $s(n)$ space and $\min(s(n), n)$ bits of advice.

The condition that $s(an) = O(s(n))$ is a technical condition satisfied by typical space bounds that are at most linear. We also obtain weaker results that apply to generic semantic models of computation.

1 Introduction

A hierarchy theorem states that the power of a machine increases with the amount of resources it can use. Time hierarchy theorems on deterministic Turing machines follow by direct diagonalization: a machine N diagonalizes against every machine M_i running in time t by choosing an input x_i, simulating $M_i(x_i)$ for t steps, and then doing the opposite. Deriving a time hierarchy theorem for nondeterministic machines is more complicated because a nondeterministic machine cannot easily complement another nondeterministic machine (unless NP=coNP). A variety of techniques can be used to overcome this difficulty, including translation arguments and delayed diagonalization [Coo73, SFM78, Žàk83].

In fact, these techniques allow us to prove time hierarchy theorems for just about any *syntactic* model of computation. We call a model syntactic if there exists a computable enumeration of all machines in the model. For example, we can enumerate all nondeterministic Turing machines by

*Department of Computer Sciences, University of Wisconsin-Madison. Email: jkinne@cs.wisc.edu. Supported by NSF Career award CCR-0133693.

†Department of Computer Sciences, University of Wisconsin-Madison. Email: dieter@cs.wisc.edu. Partially supported by NSF Career award CCR-0133693.

ISSN 1433-8092

representing their transition functions as strings and then iterating over all such strings to discover each nondeterministic Turing machine.

Many models of computation of interest are not syntactic, but *semantic*. A semantic model is defined by imposing a promise on a syntactic model. A machine belongs to the model if it is output by the enumeration of the underlying syntactic model and its execution satisfies the promise on every input. Bounded-error randomized Turing machines are an example of a non-syntactic semantic model. There does not exist a computable enumeration consisting of exactly all randomized Turing machines that satisfy the promise of bounded error on every input, but we can enumerate all randomized Turing machines and attempt to select among them those that have bounded error. In general promises make diagonalization problematic because the diagonalizing machine must satisfy the promise everywhere but has insufficient resources to determine whether a given machine from the enumeration against which it tries to diagonalize satisfies the promise on a given input.

Because of these difficulties there has yet to be a single non-trivial proof of a time hierarchy theorem for any non-syntactic model.[1] A recent line of research [Bar02, FS04, GST04, FST05, MP07] has provided progress toward proving time hierarchy results for non-syntactic models, including two-sided error randomized machines. Each of these results applies to semantic models that take advice, where the diagonalizing machine is only guaranteed to satisfy the promise when it is given the correct advice. Many of the results require only one bit of advice, which the diagonalizing machine uses to avoid simulating a machine on an input for which that machine breaks the promise.

As opposed to the setting of time, fairly good space hierarchy theorems are known for certain non-syntactic models. In fact, the following simple translation argument suffices to show that for any constant $c > 1$ there exists a language computable by two-sided error randomized machines using $(s(n))^c$ space that is not computable by such machines using $s(n)$ space [KV87], for any space-constructible $s(n)$ that is $\Omega(\log n)$. Suppose by way of contradiction that every language computable by two-sided error machines in space $(s(n))^c$ is also computable by such machines in space $s(n)$. A padding argument then shows that in that model any language computable in $(s(n))^{c^2}$ space is computable in space $(s(n))^c$ and thus in space $s(n)$. We can iterate this padding argument any constant number of times and show that for any constant d, any language computable by two-sided error machines in space $(s(n))^d$ is also computable by such machines in $s(n)$ space. For $d > 1.5$ we reach a contradiction with the deterministic space hierarchy theorem because randomized two-sided error computations that run in space $s(n)$ can be simulated deterministically in space $(s(n))^{1.5}$ [SZ99]. The same argument applies to non-syntactic models where $s(n)$ space computations can be simulated deterministically in space $(s(n))^d$ for some constant d, including one- and zero-sided error randomized machines, unambiguous machines, etc.

Since we can always reduce the space usage by a constant factor by increasing the work-tape alphabet size, the tightest space hierarchy result one might hope for is to separate space $s'(n)$ from space $s(n)$ for any space-constructible function $s'(n) = \omega(s(n))$. For models like nondeterministic machines, which are known to be closed under complementation in the space-bounded setting [Imm88, Sze88], such tight space hierarchies follow by straightforward diagonalization. For generic syntactic models, tight space hierarchies follow using the same techniques as in the time-bounded setting. Those techniques all require the existence of an efficient universal machine, which presup-

[1] Time hierarchies for a few non-syntactic models follow directly from their equivalence in power to a syntactic model. These hierarchies result from equalities such as PSPACE=IP, BP.$\oplus$P=Σ_2.$\oplus$P, NEXP=MIP=PCP(poly,poly), and NP=PCP($\log n$,1).

poses the model to be syntactic. For that reason they fail for non-syntactic models of computation such as bounded-error machines.

In this paper we obtain space hierarchy results that are tight with respect to space by adapting to the space-bounded setting techniques that have been developed for proving hierarchy results for semantic models in the time-bounded setting. Our results improve upon the space hierarchy results that can be obtained by the simple translation argument.

1.1 Our Results

Space hierarchy results have a number of parameters: (1) the gap needed between the two space bounds, (2) the amount of advice that is needed for the diagonalizing machine N, (3) the amount of advice that can be given to the smaller space machines M_i, and (4) the range of space bounds for which the results hold. We consider (1) and (2) to be of the highest importance. We focus on space hierarchy theorems with an optimal separation in space – where any super-constant gap in space suffices. The ultimate goal for (2) is to remove the advice altogether and obtain uniform hierarchy results. As in the time-bounded setting, we do not achieve this goal but get the next best result – a single bit of advice for N suffices in each of our results. Given that we strive for space hierarchies that are tight with respect to space and require only one bit of advice for the diagonalizing machine, we aim to optimize the final two parameters.

1.1.1 Randomized Models

Our strongest results apply to randomized models. For two-sided error machines, we can handle a large amount of advice and any typical space bound between logarithmic and linear. We point out that the latter is an improvement over results in the time-bounded setting, in the sense that there tightness degrades for all super-polynomial time bounds whereas here the results remain tight for a range of space bounds.

Theorem 1. *Let $s(n)$ be any space-constructible function that is $\Omega(\log n)$ and such that $s(an) = O(s(n))$ for all constants a, and let $s'(n)$ be any function that is $\omega(s(n))$. There exists a language computable by two-sided error randomized machines using $s'(n)$ space and one bit of advice that is not computable by two-sided error randomized machines using $s(n)$ space and $\min(s(n), n)$ bits of advice.*

For $s(n) = \log(n)$, Theorem 1 gives a bounded-error machine using only slightly larger than $\log n$ space that uses one bit of advice and differs from all bounded-error machines using $O(\log n)$ space and $O(\log n)$ bits of advice. The condition that $s(an) = O(s(n))$ for all constants a is a technical condition needed to ensure the construction yields a tight separation in space. The condition is true of all natural space bounds that are at most linear. More generally, our construction works for arbitrary space bounds $s(n)$ and space-constructible $s'(n)$ such that $s'(n) = \omega(s(n + as(n)))$ for all constants a.

Our second result gives a separation result with similar parameters as those of Theorem 1 but for the cases of one- and zero-sided error randomized machines. We point out that the separation result for zero-sided error machines is new to the space-bounded setting as the techniques used to prove stronger separations in the time-bounded setting do not work for zero-sided error machines. In fact, we show a single result that captures space separations for one- and zero-sided error machines – that a zero-sided error machine suffices to diagonalize against one-sided error machines.

Theorem 2. *Let $s(n)$ be any space-constructible function that is $\Omega(\log n)$ and such that $s(an) = O(s(n))$ for all constants a, and let $s'(n)$ be any function that is $\omega(s(n))$. There exists a language computable by zero-sided error randomized machines using $s'(n)$ space and one bit of advice that is not computable by one-sided error randomized machines using $s(n)$ space and $\min(s(n), n)$ bits of advice.*

1.1.2 Generic Semantic Models

The above results take advantage of specific properties of randomized machines that do not hold for arbitrary semantic models. Our next results involve a generic construction of [MP07] that applies to a wide class of semantic models which the authors term *reasonable*. We refer to Section 4.2 for the precise definition; but besides randomized two-, one-, and zero-sided error machines, the notion also encompasses bounded-error quantum machines [Wat03], unambiguous machines [BJLR91], Arthur-Merlin games and interactive proofs [Con93], etc. When applied to the logarithmic space setting, the construction yields the following.

Theorem 3 (follows from [MP07]). *Let $s'(n)$ be any function with $s'(n) = \omega(\log n)$. For any reasonable semantic model of computation, there exists a language computable using $s(n)$ space and one bit of advice that is not computable using $O(\log n)$ space and $O(1)$ bits of advice.*

The performance of the generic construction is poor on the last two parameters we mentioned earlier – it allows few advice bits on the smaller space side and is only tight for $s(n) = O(\log n)$. Either of these parameters can be improved for models that can be simulated deterministically with only a polynomial blowup in space – models for which the simple translation argument works. In fact, there is a trade-off between (a) the amount of advice that can be handled and (b) the range of space bounds for which the result is tight. By maximizing the former we get the following.

Theorem 4. *Fix any reasonable model of computation for which space $O(\log n)$ computations can be simulated deterministically in space $O(\log^d n)$ for some rational constant d. Let $s'(n)$ be any function with $s'(n) = \omega(\log n)$. There exists a language computable using $s'(n)$ space and one bit of advice that is not computable using $O(\log n)$ space and $O(\log^{1/d} n)$ bits of advice.*

In fact, a tight separation in space can be maintained while allowing $O(\log^{1/d} n)$ advice bits for $s(n)$ any poly-logarithmic function, but the separation in space with this many advice bits is no longer tight for larger $s(n)$. By maximizing (b), we obtain a separation result that is tight for typical space bounds between logarithmic and polynomial.

Theorem 5. *Fix any reasonable model of computation for which space s computations can be simulated deterministically in space $O(s^d)$ for some constant d. Let $s(n)$ be a space bound that is $\Omega(\log n)$ and such that $s(n) \leq n^{O(1)}$; let $s'(n)$ be a space bound that is constructible in space $o(s'(n))$ and such that $s'(n+1) = O(s'(n))$. If $s'(n) = \omega(s(n))$ then there is a language computable in space $s'(n)$ with one bit of advice that is not computable in space $s(n)$ with $O(1)$ bits of advice.*

The first two conditions on $s'(n)$ are technical conditions true of typical space bounds in the range of interest – between logarithmic and polynomial. When applied to randomized machines, Theorem 5 gives a tight separation result for slightly higher space bounds than Theorems 1 and 2, but the latter can handle more advice bits.

1.1.3 Generic Promise Models

Our proofs use advice in a critical way to derive hierarchy theorems for languages computable by semantic models. We can obviate the need for advice by considering *promise problems* rather than languages. A promise problem only specifies the behavior of a machine on a subset of the inputs; the machine may behave arbitrarily on inputs outside of this set. For semantic models of computation, one can associate in a natural way a promise problem to each machine in the underlying enumeration. For example, for randomized machines with bounded error, the associated promise problem leaves the behavior unspecified on inputs where the randomized machine violates the bounded-error condition. The ability to ignore problematic inputs allows traditional techniques to demonstrate good space and time hierarchy theorems for the promise problems computable by semantic models. This is a folklore result, but there does not appear to be a correct proof in the literature; we include one in this paper.

Theorem 6 (**folklore**). *Fix a reasonable model of computation. Let $s(n)$ and $s'(n)$ be space bounds with $s(n) = \Omega(\log n)$ and $s'(n)$ space constructible. If $s'(n) = \omega(s(n+1))$ then there is a promise problem computable within the model using space $s'(n)$ that is not computable as a promise problem within the model using space $s(n)$.*

1.2 Our Techniques

Recently, Van Melkebeek and Pervyshev [MP07] showed how to adapt the technique of delayed diagonalization to obtain time hierarchies for any reasonable semantic model of computation with one bit of advice. For any constant a, they exhibit a language that is computable in polynomial time with one bit of advice but not in linear time with a bits of advice. Our results for generic models of computation (Theorems 3, 4, and 5) follow from a space-efficient implementation and a careful analysis of that approach.

Our stronger results for randomized machines follow a different type of argument, which roughly goes as follows. When N diagonalizes against machine M_i, it tries to achieve complementary behavior on inputs of length n_i by reducing the complement of M_i at length n_i to instances of some hard language L of length somewhat larger than n_i, say m_i. N cannot compute L on those instances directly because we do not know how to compute L in small space. We instead use a delayed computation and copying scheme that forces M_i to aid N in the computation of L if M_i agrees with N on inputs larger than m_i. As a result, either M_i differs from N on some inputs larger than m_i, or else N can decide L at length m_i in small space and therefore diagonalize against M_i at length n_i.

The critical component of the copying scheme is the following task. Given a list of randomized machines with the guarantee that at least one of them satisfies the promise and correctly decides L at length m in small space, construct a single randomized machine that satisfies the promise and decides L at length m in small space. We call a procedure accomplishing this task a space-efficient *recovery procedure* for L.

The main technical contributions of this paper are the design of recovery procedures for adequate hard languages L. For Theorem 1 we use the computation tableau language, which is an encoding of bits of the computation tableaux of deterministic machines; we develop a recovery procedure based on the local checkability of computation tableaux. For Theorem 2 we use the configuration reachability language, which is an encoding of pairs of configurations that are connected in a

nondeterministic machine's configuration graph; we develop a recovery procedure from the proof that NL=coNL [Imm88, Sze88].

1.2.1 Relation to Previous Work

Our high-level strategy is most akin to the one used in [MP07]. In the time-bounded setting, [MP07] achieves a strong separation for bounded-error randomized machines using the above construction with satisfiability as the hard language L. Hardness of L follows from the fact that randomized machines can be time-efficiently deterministically simulated using a randomized two-sided error algorithm for satisfiability. The recovery procedure exploits the self-reducibility of satisfiability to obtain satisfying assignments for satisfiable formulae. As the partial assignment must be stored during the construction, this approach uses too much space to be useful in the setting of this paper.

[MP07] also derives a stronger separation for bounded-error quantum machines in the time-bounded setting, with the hard language L being PSPACE-complete. A time-efficient recovery procedure for L follows from the existence of instance checkers for L. The latter transformation of instance checkers into recovery procedures critically relies on large memory space. Instance checkers are only guaranteed to work when given a fixed oracle to test; their properties carry over to testing randomized procedures by treating randomized procedures as probability distributions over oracles. This works in the time-bounded setting because we can ensure consistent answers to the oracle queries by storing the answers of the randomized procedure to all queries the first time they are asked. In the space-bounded setting we do not have the resources to store the answers to all queries, which implies we can no longer treat randomized procedures as probability distributions over oracles. As a result, it is no longer obvious that an instance checker yields a recovery procedure. Note that the other direction is not immediate either because a recovery procedure may rely on the guarantee that the list of machines contains one that correctly computes the language, whereas an instance checker needs to be able to detect misbehavior on a given input. The two notions seem incomparable in the space-bounded setting. As the methods for designing recovery procedures in the time-bounded setting do not carry over to the space-bounded setting, new ingredients are required here.

We point out that some of our results can also be obtained using a different high-level strategy than the one in [MP07], which can be viewed as delayed diagonalization with advice. Some of the results of [MP07] in the time-bounded setting can also be derived by adapting translation arguments to use advice [Bar02, FS04, GST04, FST05]. It is possible to derive our Theorems 1 and 2 following a space-bounded version of the latter strategy. However, the proofs still rely on the recovery procedure as a key technical ingredient and we feel that our proofs are simpler. Moreover, for the case of generic semantic models, our approach yields results that are strictly stronger.

1.3 Organization

Section 3 contains the proofs of our separation results for randomized models (Theorems 1 and 2). Section 4 contains the proofs of our separation results for generic semantic models (Theorems 3, 4 and 5). Section 5 contains a proof of the hierarchy theorem for promise problems of semantic models (Theorem 6).

2 Preliminaries

We assume familiarity with standard definitions for randomized complexity classes, including two-, one-, and zero-sided error machines. For each machine model requiring randomness, we allow the machine one-way access to the randomness and only consider computations where each machine always halts in finite time. The default for bounded-error machines is that the probability of error on every input is bounded by 1/3.

Our separation results apply to machines that take advice. We use α and β to denote infinite sequences of advice strings. Given a machine M, M/β denotes the machine M taking advice β. Namely, on input x, M is given both x and $\beta_{|x|}$ as input. When we are interested in the execution of M/β on inputs of length n, we write M/b where $b = \beta_n$.

We consider semantic models of computation, with an associated computable enumeration $(M_i)_{i=1,2,3,\ldots}$ and an associated promise. A machine falls within the model if it is contained in the enumeration and its behavior satisfies the promise on all inputs.

For a machine M/β^* that takes advice, we only require that M satisfies the promise when given the "correct" advice sequence β^*. We note that this differs from the Karp-Lipton notion of advice of [KL82], where the machine must satisfy the promise no matter which advice string is given. A hierarchy for a semantic model with advice under the stronger Karp-Lipton notion would imply the existence of a hierarchy without advice.

3 Randomized Machines with Bounded Error

In this section we give the constructions for Theorems 1 and 2. We first describe the high-level strategy used for these results. Most portions of the construction are the same for both, so we keep the exposition general. We aim to construct a randomized machine N and advice sequence α witnessing Theorems 1 and 2 for some space bounds $s(n)$ and $s'(n)$. N/α should always satisfy the promise, run in space $s'(n)$, and differ from M_i/β for randomized machines M_i and advice sequences β for which M_i/β *behaves appropriately*, i.e., for which M_i/β satisfies the promise and uses at most $s(n)$ space on all inputs.

As with delayed diagonalization, for each M_i we allocate an interval of input lengths $[n_i, n_i^*]$ on which to diagonalize against M_i. That is, for each machine M_i and advice sequence β such that M_i/β behaves appropriately, there is an $n \in [n_i, n_i^*]$ such that N/α and M_i/β decide differently on at least one input of length n. The construction consists of three main parts: (1) reducing the complement of the computation of M_i on inputs of length n_i to instances of a hard language L of length m_i, (2) performing a delayed computation of L at length m_i on inputs of length n_i^*, and (3) copying this behavior to smaller and smaller inputs down to input length m_i. These will ensure that if M_i/β behaves appropriately, either N/α differs from M_i/β on some input of length larger than m_i, or N/α computes L at length m_i allowing N/α to differ from M_i/b for all possible advice strings b at length n_i. We describe how to achieve (1) for two-sided error machines in section 3.1 and for one- and zero-sided error machines in section 3.2. For now, we assume a hard language L and describe (2) and (3). Figure 1 contains an illustration of the construction; the reader is encouraged to refer to this figure while reading the rest of this section.

Let us first try to develop the construction without assuming any advice for N or for M_i and see why N needs at least one bit of advice. On an input x of length n_i, N reduces the complement of $M_i(x)$ to an instance of L of length m_i. Because N must run in space not much more than

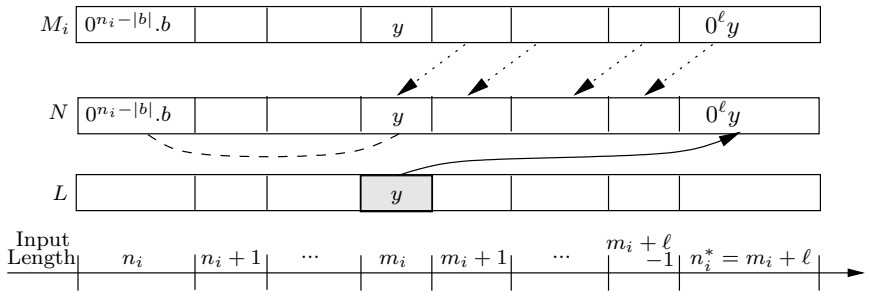

Figure 1: Illustration of the construction for Theorems 1 and 2. The solid arrow indicates that on input $0^\ell y$, N deterministically computes $L(y)$ for each y of length m_i. The dotted arrows indicate that for $\ell' \in [0, \ell-1]$, on input $0^{\ell'}y$ with advice bit 1, N attempts to compute $L(y)$ by using the recovery procedure and making queries to M_i on padded inputs of one larger length. The dashed line indicates that on input $0^{n_i-|b|}b$ with advice bit 1, N complements $M_i(0^{n_i-|b|}b)/b$ by reducing to an instance y of L and simulating $N(y)$.

$s(n)$ and we do not know how to compute the hard languages we use with small space, N cannot directly compute L at length m_i. However, L can be computed at length m_i within the space N is allowed to use on much larger inputs. Let n_i^* be large enough so that L at length m_i can be deterministically computed in space $s'(n_i^*)$. We let N at length n_i^* perform a *delayed computation* of L at length m_i as follows: on inputs of the form $0^\ell y$ where $\ell = n_i^* - m_i$ and $|y| = m_i$, N uses the above deterministic computation of L on input y to ensure that $N(0^\ell y) = L(y)$.

Since N performs a delayed computation of L, M_i must as well – otherwise N already computes a language different than M_i. We would like to bring this delayed computation down to smaller padded inputs. The first attempt at this is the following: on input $0^{\ell-1}y$, N simulates $M_i(0^\ell y)$. If M_i behaves appropriately and performs the initial delayed computation, then $N(0^{\ell-1}y) = M_i(0^\ell y) = L(y)$, meaning that N satisfies the promise and performs the delayed computation of L at length m_i at an input length one smaller than before. However, M_i may not behave appropriately on inputs of the form $0^\ell y$; in particular M_i may fail to satisfy the promise, in which case N would also fail to satisfy the promise by performing the simulation. If M_i does not behave appropriately, N does not need to consider M_i and could simply abstain from the simulation. If M_i behaves appropriately on inputs of the form $0^\ell y$, it still may fail to perform the delayed computation. In that case N has already diagonalized against M_i at input length $m_i + \ell$ and can therefore also abstain from the simulation on inputs of the form $0^{\ell-1}y$.

N has insufficient resources to determine on its own if M_i behaves appropriately and performs the initial delayed computation. Instead, we give N one bit of advice at input length $m_i + \ell - 1$ indicating whether M_i behaves appropriately and performs the initial delayed computation at length $n_i^* = m_i + \ell$. If the advice bit is 0, N acts trivially at this length by always rejecting inputs. If the advice bit is 1, N performs the simulation so $N(0^{\ell-1}y)/\alpha = M_i(0^\ell y) = L(y)$.

If we give N one bit of advice, we should give M_i at least one advice bit as well. Otherwise, the hierarchy result is not fair (and is trivial). Consider how allowing M_i advice effects the construction. If there exists an advice string b such that M_i/b behaves appropriately and $M_i(0^\ell y)/b = L(y)$ for all y with $|y| = m_i$, we set N's advice bit for input length $m_i+\ell-1$ to be 1, meaning N should copy down the delayed computation from length $m_i + \ell$ to length $m_i + \ell - 1$. Note, though, that N does not know for which advice b the machine M_i/b appropriately performs the delayed computation

at length $m_i + \ell$. N has at its disposal a list of machines, M_i with each possible advice string b, with the guarantee that at least one M_i/b behaves appropriately and $M_i(0^\ell y)/b = L(y)$ for all y with $|y| = m_i$. With this list of machines as its primary resource, N wishes to ensure that $N(0^{\ell-1}y)/\alpha = L(y)$ for all y with $|y| = m_i$ while satisfying the promise and using small space.

N can accomplish this task given a space-efficient recovery procedure for L at length m_i: on input $0^{\ell-1}y$, N removes the padding and executes the recovery procedure to determine $L(y)$, for each b simulating $M_i(0^\ell y')/b$ when the recovery procedure makes a query y'. As the space complexity of the recovery procedures we give in sections 3.1 and 3.2 is within a constant factor of a single simulation of M_i, this process uses $O(s(n))$ space. We point out that for Theorem 1, the recovery procedure may have two-sided error, while for Theorem 2, the recovery procedure must have zero-sided error.

Given a recovery procedure for L, N/α correctly performs the delayed computation on inputs of length $m_i + \ell - 1$ if there is an advice string causing M_i to behave appropriately and perform the initial delayed computation at length $m_i + \ell$. We repeat the process on padded inputs of the next smaller size. Namely, N's advice bit for input length $m_i + \ell - 2$ is set to indicate if there is an advice string b such that M_i/b behaves appropriately on inputs of length $m_i + \ell - 1$ and $M_i(0^{\ell-1}y)/b = L(y)$ for all y with $|y| = m_i$. If so, then on inputs of the form $0^{\ell-2}y$, N/α uses the recovery procedure for L to determine the value of $L(y)$, for each b simulating $M_i(0^{\ell-1}y')/b$ when the recovery procedure makes a query y'. By the correctness of the recovery procedure, N/α thus correctly performs the delayed computation on padded inputs of length $m_i + \ell - 2$. If the advice bit is 0, N/α acts trivially at input length $m_i + \ell - 2$ by rejecting immediately.

We repeat the same process on smaller and smaller padded inputs. We reach the conclusion that either there is a largest input length $n \in [m_i + 1, n_i^*]$ where for no advice string b, M_i/b appropriately performs the delayed computation of L at length n; or N/α correctly computes L on inputs of length m_i. If the former is the case, N/α performs the delayed computation at length n whereas for each b either M_i/b does not behave appropriately at length n or it does but does not perform the delayed computation at length n. In either case, N/α has diagonalized against M_i/b for each possible b at length n. N's remaining advice bits for input lengths $[n_i, n-1]$ are set to 0 to indicate that nothing more needs to be done, and N/α immediately rejects inputs in this range. Otherwise N/α correctly computes L on inputs of length m_i. In that case N/α diagonalizes against M_i/b for all advice strings b at length n_i by acting as follows. On input $x_b = 0^{n_i-|b|}b$, N reduces the complement of the computation $M_i(x_b)/b$ to an instance y of L of length m_i and then simulates $N(y)/\alpha$, so $N(x_b)/\alpha = N(y)/\alpha = L(y) = \neg M_i(x_b)/b$.

We have given the major points of the construction, with the notable exception of the recovery procedures. We develop these in the next two sections. We save the resource analysis of the construction for section 3.3.

3.1 Two-sided Error Recovery Procedure – Computation Tableau Language

In this section we develop a space-efficient recovery procedure for the computation tableau language (hereafter written COMP), the hard language used in the construction of Theorem 1.

> COMP = $\{\langle M, x, t, j\rangle \,|M$ is a deterministic Turing machine, and in the t^{th} time step of executing $M(x)$, the j^{th} bit in the machine's configuration is equal to 1$\}$.

Let us see that COMP is in fact "hard" for two-sided error machines. For some input x, we would like to know whether $\Pr[M_i(x) = 1] < \frac{1}{2}$. For a particular random string, whether $M_i(x)$ accepts

or rejects can be decided by looking at a single bit in M_i's configuration after a certain number of steps – by ensuring that M_i enters a unique accepting configuration when it accepts. With the randomness unfixed, we view $M_i(x)$ as defining a Markov chain on the configuration space of the machine. Provided $M_i(x)$ uses at most $s(n)$ space, a deterministic machine running in $2^{O(s)}$ time and space can estimate the state probabilities of this Markov chain to sufficient accuracy and determine whether a particular configuration bit has probability at most $1/2$ of being 1 after t time steps. This deterministic machine and a particular bit of its unique halting configuration define the instance of COMP we would like to solve when given input x.

We now present the recovery procedure for COMP. We wish to compute COMP on inputs of length m in space $O(s(m))$ with bounded error when given a list of randomized machines with the guarantee that at least one of the machines computes COMP on all inputs of length m using $s(m)$ space with bounded error. Let $y = \langle M, x, t, j\rangle$ be an instance of COMP with $|y| = m$ that we wish to compute. Pseudo-code is given in Figure 2, which the reader may find helpful to consult while reading the remainder of this section.

A natural way to determine COMP(y) is to consider each machine in the list one at a time and design a test that determines whether a particular machine computes COMP(y). The test should have the following properties:

(i) if the machine in question correctly computes COMP on all inputs of length m, the test declares success with high probability, and

(ii) if the test declares success with high probability, then the machine in question gives the correct answer of COMP(y) with high probability.

Given such a test, the recovery procedure consists of iterating through each machine in the list in turn. We take the first machine P to pass testing, simulate $P(y)$ some number of times and output the majority answer. Given a testing procedure with properties (i) and (ii), correctness of this procedure follows using standard probability arguments (Chernoff and union bounds) and the assumption that we are guaranteed that at least one machine in the list of machines correctly computes COMP at length m.

The technical heart of the recovery procedure is the testing procedure to determine if a given machine P correctly computes COMP(y) for $y = \langle M, x, t, j\rangle$. This test is based on the local checkability of computation tableaux – the j^{th} bit of the configuration of $M(x)$ in time step t is determined by a constant number of bits from the configuration in time step $t-1$. For each bit (t, j) of the tableau, this gives a local consistency check – make sure that the value P claims for $\langle M, x, t, j\rangle$ is consistent with the values P claims for each of the bits of the tableau that this bit depends on. We implement this intuition as follows.

1. For each possible t' and j', simulate $P(\langle M, x, t', j'\rangle)$ a large number of times and fail the test if the acceptance ratio lies in the range $[3/8, 5/8]$.

2. For each possible t' and j', do the following. Let j'_1, ..., j'_k be the bits of the configuration in time step $t'-1$ that bit j' in time step t' depends on. Simulate each of $P(\langle M, x, t', j'\rangle)$, $P(\langle M, x, t'-1, j'_1\rangle)$, ..., $P(\langle M, x, t'-1, j'_k\rangle)$ a large number of times. If the majority values of these simulations are not consistent with the transition function of M, then fail the test. For example, if the bit in column j' should not change from time $t'-1$ to time t', but P has claimed different values for these bits, fail the test.

Input: $y = \langle M, x, t, j \rangle$ of length m; machines $P_1, P_2, ..., P_q$
Output: COMP(y)

(1) **foreach** $d = 1..q$ *Try using P_d to compute COMP(y)*
(2) **foreach** t' and j' *Bounded-error checks*
(3) $(A, R) \leftarrow$(#accept, #reject) runs of $2^{O(s(m))}$ simulations of $P_d(\langle M, x, t', j' \rangle)$
(4) **if** $\frac{A}{A+R} \in [3/8, 5/8]$ **then** Try next value of d (line 1) *P_d fails*
(5) **foreach** j' *Check base case – start configuration*
(6) $A \leftarrow$ majority of $2^{O(s(m))}$ simulations of $P_d(\langle M, x, j', 0 \rangle)$
(7) **if** $A \neq j'^{th}$ bit of start configuration
(8) **then** Try next value of d (line 1) *P_d fails*
(9) **foreach** $t' > 0$ and j' *Local consistency checks*
(10) bit j' in time step t' depends on bits $j'_1, j'_2, ..., j'_k$ in time step $t' - 1$
(11) **foreach** $c = 1, 2, ..., k$
(12) $A_{j'_c, t'-1} \leftarrow$ majority of $2^{O(s(m))}$ simulations of $P_d(\langle M, x, j'_c, t' - 1 \rangle)$
(13) $A_{j',t'} \leftarrow$ majority of $2^{O(s(m))}$ simulations of $P_d(\langle M, x, j', t' \rangle)$
(14) **if** $A_{j',t'}, A_{j'_1,t'-1}, A_{j'_2,t'-1}, ..., A_{j'_k,t'-1}$ violate transition function of M
(15) **then** Try next value of d (line 1) *P_d fails*
P_d passed all tests
(16) Output majority of $2^{O(s(m))}$ simulations of $P_d(\langle M, x, j, t \rangle)$ and halt
(17) Reject *No machines passed testing*

Figure 2: Pseudo-code for the two-sided error recovery procedure for the computation tableau language. The list of machines is guaranteed to contain at least one computing COMP at length m with two-sided error in space $s(m)$. Lines 2, 5, and 9 loop over all t' and j' valid for M using $2^{O(s(m))}$ time and space, and indices t, j, t', and j' are padded so that all instances of COMP of interest are of length m.

The first test, given by lines 2-4 in Figure 2, checks that P has error bounded away from 1/2 on input $\langle M, x, t, j \rangle$ and on all other bits of the computation tableau of $M(x)$. This allows us to amplify the error probability of P to exponentially small in $2^{s(m)}$. For some constants $0 < \gamma < \delta < 1/2$, the first test has the following properties: (A) If P passes the test with non-negligible probability then for any t' and j', the random variable $P(\langle M, x, t', j' \rangle)$ deviates from its majority value with probability less than δ, and (B) if the latter is the case with δ replaced by γ then P passes the test with overwhelming probability. The second test, given by lines 5-15 in Figure 2, verifies the local consistency of the computation tableau claimed by P. Note that if P computes COMP correctly at length m then P passes each consistency test with high probability, and if P passes each consistency test with high probability then P must compute the correct value for COMP(y). This along with the two properties of the first test guarantee that we can choose a large enough number of trials for the second test so that properties (i) and (ii) from above are satisfied.

Consider the space usage of the recovery procedure, given in pseudo-code in Figure 2. The main tasks are the following: (a) cycle over all machines in the list of machines, and (b) for each t' and j' determine the bits of the tableau that bit (t', j') depends on and for each of these run $2^{O(s(m))}$ simulations of P. The first requirement depends on the representation of the list of machines. For our application, we will be cycling over all advice strings for input length m, and this takes $O(s(m))$

space provided advice strings for M_i are of length at most $s(m)$. The second requirement takes an additional $O(s(m))$ space by the fact that we only need to simulate P while it uses $s(m)$ space and the fact that the computation tableau bits that bit (t', j') depends on are constantly many and can be computed very efficiently.

Finally, we note that a result corresponding to Theorem 1 also applies to space-bounded quantum machines [Wat03]. The key properties needed for this proof to apply to a model are: (1) the behavior of unbounded-error $s(n)$ space bounded machines is computable in deterministic time and space $2^{O(s(n))}$, and (2) taking the majority of $2^{O(s(n))}$ many trials can be done in $O(s(n))$ space and the model is closed under $O(s(n))$ space function composition. Space-bounded quantum machines with two-sided error satisfy both of these properties.

3.2 Zero-sided error Recovery Procedure – Configuration Reachability

In this section we develop a space-efficient recovery procedure for the configuration reachability language (hereafter written CONFIG), the hard language used in the construction of Theorem 2.

> CONFIG = $\{\langle M, x, c_1, c_2, t\rangle \,|M$ is a nondeterministic Turing machine, and on input x, if M is in configuration c_1, then configuration c_2 is reachable within t time steps$\}$.

We point out that CONFIG is "hard" for one-sided error machines since a one-sided error machine can also be viewed as a nondeterministic machine. That is, if we want to know whether $\Pr[M_i(x) = 1] < \frac{1}{2}$ for M_i a one-sided error machine that uses $s(n)$ space, we can query the CONFIG instance $\left\langle M_i, x, c_1, c_2, 2^{O(s(|x|))}\right\rangle$ where c_1 is the unique start configuration, and c_2 is the unique accepting configuration.

We now present the recovery procedure for CONFIG. We wish to compute CONFIG on inputs of length m with *zero-sided error* and in space $O(s(m))$ when given a list of randomized machines with the guarantee that at least one of the machines computes CONFIG on all inputs of length m using $s(m)$ space with *one-sided error*. Let $y = \langle M, x, c_1, c_2, t\rangle$ be an instance of CONFIG with $|y| = m$ that we wish to compute. Pseudo-code is given in Figure 3, which the reader may find helpful to consult while reading the remainder of this section.

As we need to compute CONFIG with zero-sided error, we can only output a value of "yes" or "no" if we are sure this is correct. The outer loop of our recovery procedure is the following: cycle through each machine in the list of machines, and for each execute a search procedure that attempts to verify whether configuration c_2 is reachable from configuration c_1. The search procedure may output "yes", "no", or "fail", and should have the following properties:

(i) if the machine in question correctly computes CONFIG at length m, the search procedure comes to a definite answer ("yes" or "no") with high probability, and

(ii) when the search procedure comes to a definite answer, it is always correct, no matter what the behavior of the machine in question.

We cycle through all machines in the list, and if the search procedure ever outputs "yes" or "no", we halt and output that response. If the search procedure fails for all machines in the list, we output "fail". Given a search procedure with properties (i) and (ii), the correctness of the recovery procedure follows from the fact that we are guaranteed that one of the machines in the list of machines correctly computes CONFIG at length m.

Input: $y = \langle M, x, c_1, c_2, t\rangle$ of length m; machines $P_1, P_2, ..., P_q$
Output: CONFIG(y)

(1) **if** $c_1 = c_2$ **then** Output "yes" and halt — *Trivial cases*
(2) **else if** $t = 0$ **then** Output "no" and halt
(3) **foreach** $d = 1..q$ — *Try using P_d to compute CONFIG(y)*
(4) $k_0 \leftarrow 1$ — *Number of configurations w/in distance 0 of c_1*
(5) **for** $\ell = 1$ **to** t — *Compute k_ℓ given $k_{\ell-1}$*
(6) $k_\ell \leftarrow 0$
(7) **foreach** configuration c — *Is c w/in distance ℓ of c_1?*
(8) $k'_{\ell-1} \leftarrow 0$ — *Re-experience all configurations w/in distance $\ell - 1$*
(9) **foreach** configuration c'
(10) **if** $Verify(\langle M, x, c_1, c', \ell - 1\rangle, P_d) =$ "yes" — *c' w/in distance $\ell - 1$ of c_1*
(11) **if** c reachable from c' in one time step by M on input x — *c w/in distance ℓ of c_1*
(12) **if** $c = c_2$ **then** Output "yes" and halt
(13) **else** $k_\ell \leftarrow k_\ell + 1$, and Try next c (line 7)
(14) **else**
(15) $k'_{\ell-1} \leftarrow k'_{\ell-1} + 1$
(16) **if** $k'_{\ell-1} \neq k_{\ell-1}$ — *Failed to re-experience all configurations w/in distance $\ell - 1$*
(17) **if** $d < q$ **then** Try next d (line 3) — *P_d fails*
(18) **else** Output "fail" and halt — *All machines have failed*
(19) Output "no" and halt — *k_t computed correctly and c_2 not found*

Figure 3: Pseudo-code for the zero-sided error recovery procedure for the configuration reachability language. The list of machines is guaranteed to contain at least one computing CONFIG at length m with one-sided error in space $s(m)$. Configurations c_1, c_2, and c' and time values t and $\ell - 1$ are padded so that all instances of CONFIG of interest are of length m. The code for *Verify* used on line 10 is given in Figure 4.

The technical heart of the recovery procedure is a search procedure with properties (i) and (ii). Let P be a randomized machine under consideration, and $y = \langle M, x, c_1, c_2, t\rangle$ an input of length m we wish to compute. Briefly, the main idea is to mimic the proof that NL=coNL to verify reachability and un-reachability, replacing nondeterministic guesses with simulations of P. If P computes CONFIG at length m correctly, there is a high probability that we have correct answers to all nondeterministic guesses, meaning property (i) is satisfied. Property (ii) follows from the fact that the algorithm can discover when incorrect nondeterministic guesses have been made. For completeness, we explain how the nondeterministic algorithm of [Imm88, Sze88] is used in our setting. The search procedure works as follows.

1. Let k_0 be the number of configurations reachable from c_1 within 0 steps, i.e., $k_0 = 1$.

2. For each value $\ell = 1, 2, ..., t$, compute the number k_ℓ of configurations reachable within ℓ steps of c_1, using only the fact that we have remembered the value $k_{\ell-1}$ that was computed in the previous iteration.

Verify
Input: $y = \langle M, x, c_0, c', t\rangle$ with $|y| = m$; machine P
Output: "yes" if verified that c' reachable from c_0 by M on input x in $\leq t$ time steps, "fail" otherwise

(1) **if** $c_0 = c'$ **then** Output "yes" and halt — *Trivial cases*
(2) **else if** $t = 0$ **then** Output "fail and halt
(3) $c \leftarrow c_0$ — *Current configuration on path from c_0 to c'*
(4) **for** $j = t - 1$ **down to** 0 — *Try to move w/in distance j of c'*
(5) **foreach** configuration c'' — *Is c'' neighbor of c and one step closer to c'?*
(6) **if** c'' reachable from c in one time step by M on input x — *c'' neighbor of c*
(7) **if** $c'' = c'$ — *Have already reached c'*
(8) Output "yes" and halt
(9) **else if** any of $2^{O(s(m))}$ simulations of $P(\langle M, x, c'', c', j\rangle)$ outputs 1
(10) $c \leftarrow c''$ and try next j (line 4) — *Now c is one step closer*
(11) Output "fail" and halt — *Unable to move one step closer to c'*
(12) Output "fail" — *After t steps, have not reached c'*

Figure 4: Pseudo-code for the verification subroutine used in the zero-sided error recovery procedure of Figure 3. If configuration c' is within distance t of configuration c_0 and P appropriately computes CONFIG at length m, then with high probability a path is verified and "yes" is returned. "Yes" is only returned when a path of length at most t has been verified. Configurations c_0, c', and c'', as well as time values t and j are padded so that all queries to CONFIG of interest are of length m.

3. While computing k_t, experience all of these configurations to see if c_2 is among them.

Consider the portion of the second step where we must compute k_ℓ given that we have already computed $k_{\ell-1}$. We accomplish this, lines 6-18 of Figure 3, by cycling through all configurations c and for each one re-experiencing all configurations reachable from c_1 within $\ell - 1$ steps and verifying whether c can be reached in at most one step from at least one of them. To re-experience configurations reachable within distance $\ell - 1$, we try all possible configurations and query P to verify a nondeterministic path to each. The verification of a nondeterministic path is given in Figure 4. To check if c is reachable within one step of a given configuration, we use the transition function of M. If we fail to re-experience all $k_{\ell-1}$ configurations or if P gives information inconsistent with the transition function of M at any point we consider the search for reachability/un-reachability failed with machine P.

An examination of the algorithm, given in pseudo-code in Figure 3, reveals that it has property (ii) from above: if the procedure reaches a "yes" or "no" conclusion for reachability, it must be correct. Further, by using a large enough number of trials each time we simulate P, we can ensure that we get correct answers on every simulation of P with high probability if P correctly computes CONFIG at length m. This implies property (i) from above.

Consider the space usage of the recovery procedure. A critical component is to be able to cycle over all configurations and determine whether two configurations are "adjacent". As the instances of CONFIG we are interested in correspond to a machine which uses $s(n)$ space, these two tasks can be accomplished in $O(s(m))$ space. The remaining tasks of the recovery procedure take $O(s(m))$ space for similar reasons as given for the recovery procedure for the computation tableau language

in the previous section.

3.3 Analysis

In this section we explain how we come to the parameters given in the statements of Theorems 1 and 2. First, consider the space usage of the construction. The recovery procedures use $O(s(m))$ space when dealing with inputs of size m, and the additional tasks of the diagonalizing machine N also take $O(s(m))$ space. For input lengths n where N is responsible for copying down the delayed computation of the hard language L, N executes the recovery procedure using M_i on padded inputs of one larger length. Thus for such input lengths, the space usage of N is $O(s(n+1))$. For input length n_i, N produces an instance y of the hard language corresponding to complementary behavior of M_i on inputs of length n_i and then simulates $N(y)$. For two-sided error machines, we reduce to the computation tableau language COMP. When M_i is allowed $s(n)$ space, the resulting instance of COMP is of size $n+O(s(n))$. For one- and zero-sided error machines, we reduce to configuration reachability, and the resulting instance is also of size $n + O(s(n))$. In both cases, the space usage of N on inputs of length n_i is $O(s(n_i + O(s(n_i))))$. We have chosen COMP and CONFIG as hard languages over other natural candidates (such as the circuit value problem for Theorem 1 and st-connectivity for Theorem 2) because COMP and CONFIG minimize the blowup in input size incurred by using the reductions.

The constant hidden in the big-O notation depends on things such as the alphabet size of M_i. If $s'(n) = \omega(s(n + as(n)))$ for all constants a, N operating in space $s'(n)$ has enough space to diagonalize against each M_i for large enough n. To ensure the asymptotic behavior has taken effect, we have N perform the construction against each machine M_i infinitely often. We set N's advice bit to zero on the entire interval of input lengths if N does not yet have sufficient space. Note that this use of advice obviates the need for $s'(n)$ to be space constructible.

Now consider the amount of advice that the smaller space machines can be given. As long as the advice is at most $s(n)$, the recovery procedure can efficiently cycle through all candidate machines (M_i with each possible advice string). Also, to complement M_i for each advice string at length n_i, we need at least one input for each advice string of length n_i. Thus, the amount of advice that can be allowed is $\min(s(n), n)$.

Finally, we point out that a slightly weaker version of Theorem 2 – namely a separation between the two different models of zero-sided error and *one-sided* error machines – would follow even if we had only demonstrated a zero-sided error machine that diagonalizes against *zero-sided* error machines. Suppose otherwise, namely that for appropriate choices of s' and s there is a zero-sided error machine N using space $s'(n)$ and one bit of advice that computes a language different than any zero-sided error machine using $s(n)$ space and $\min(s(n), n)$ bits of advice, but that all languages decided by zero-sided error machines using $s'(n)$ space and one bit of advice can be decided by one-sided error machines using $s(n)$ space and $a(n)$ bits of advice, for some function $a(n)$. In particular, both the language decided by N/α and its complement can be decided by one-sided error machines using $s(n)$ space and $a(n)$ bits of advice. Consider the following algorithm for computing the same language as that of N/α: (1) execute the one-sided error algorithm for deciding N/α which uses $s(n)$ space and $a(n)$ bits of advice, and output "yes" if this algorithm outputs "yes", (2) execute the one-sided error algorithm for deciding the complement of N/α which uses $s(n)$ space and $a(n)$ bits of advice, and output "no" if this algorithm outputs "yes", (3) otherwise output "fail". Given the correct advice strings for the algorithms in (1) and (2), this is a zero-sided error algorithm for

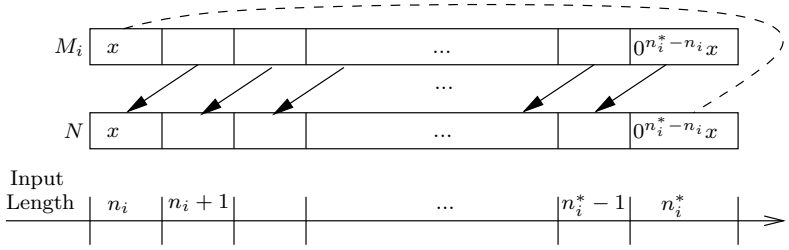

Figure 5: Illustration of delayed diagonalization on a *syntactic* model of computation. The solid arrows indicate that on inputs of the form $0^j x$, N simulates $M_i(0^{j+1}x)$. The dashed line indicates that on input $0^{n_i^*-n_i}x$, N outputs the complement of $M_i(x)$.

deciding N/α; it uses $s(n)$ space and $2a(n)$ bits of advice. This contradicts the assumed hardness of N/α against zero-sided error machines provided $2a(n) \leq \min(s(n), n)$, and we conclude that there is a language computable by zero-sided error algorithms using $s'(n)$ space and one bit of advice that is not computable by one-sided error algorithms using $s(n)$ space and $\frac{1}{2}\min(s(n), n)$ bits of advice. Note that the notion of advice we use – a zero-sided error algorithm is only required to maintain zero-sided error when given the correct advice string – is critical for this argument to hold. Also note that the maximum amount of advice that can be handled with this argument is a factor of two smaller than that given by Theorem 2.

4 Separation Results for Generic Semantic Models

In this section, we prove our separation results for generic semantic models (Theorems 3, 4, and 5). The basic construction is the same for each, with only the analysis differing. We start with the basic construction having some specific semantic models in mind. We analyze the construction in section 4.1 and distill the precise properties required of a generic semantic model in section 4.2.

As the basic construction is an adaptation of delayed diagonalization to handle advice, we first review delayed diagonalization on *syntactic* models. We wish to demonstrate a machine N using slightly more than $s(n)$ space which differs from all machines that use $s(n)$ space. For each machine M_i, N allocates an interval of input lengths $[n_i, n_i^*]$ on which to diagonalize against M_i. The construction consists of two main parts: (1) a delayed complementation at length n_i^* of M_i's behavior at length n_i, and (2) a scheme to copy this behavior down to smaller and smaller padded input lengths all the way to n_i. For (1), we choose n_i^* large enough so that N has sufficient space at length n_i^* to complement the behavior of M_i at length n_i. N performs a *delayed complementation* by ensuring that $N(0^{n_i^*-n_i}x) = \neg M_i(x)$ for x with $|x| = n_i$. For (2), on inputs of the form $0^j x$ with $|x| = n_i$ and $0 \leq j < n_i^* - n_i$, N simulates $M_i(0^{j+1}x)$ while M_i uses at most $s(n)$ space, outputs a value if M_i does, and outright rejects if M_i uses more than $s(n)$ space. Suppose that M_i is a machine which uses at most $s(n)$ space and computes the same language as N on all input lengths in $[n_i, n_i^*]$. This assumption and N's definition imply the following set of equalities for every input x of length n_i:

$$M_i(x) = N(x) = M_i(0x) = N(0x) = M_i(0^2x) = ... = M_i(0^{n_i^*-n_i}x) = N(0^{n_i^*-n_i}x) = \neg M_i(x).$$

As $M_i(x)$ must take some definite value, we have reached a contradiction. Either M_i differs from N on some input of length in $[n_i, n_i^*]$, or M_i uses more than $s(n)$ space. An illustration of delayed

diagonalization is given in Figure 5.

Consider the case of a *semantic* model of computation, where N must use not much more than $s(n)$ space, satisfy the promise on all inputs, and differ from each machine M_i which *behaves appropriately*, i.e., which satisfies the promise and uses at most $s(n)$ space on all inputs. We keep a few specific semantic models in mind during the development and analysis of the construction – Arthur-Merlin games for Theorem 3, and unambiguous machines for the stronger separations of Theorems 4 and 5. We use certain closure properties of these models in the analysis; we refer to section 4.2 for the precise properties required of a generic semantic model. The delayed diagonalization construction given in the previous paragraph fails for such non-syntactic models: it may be the case that M_i breaks the promise on inputs of the form $0^j x$, and N would also break the promise by performing the simulations described above. However, if M_i breaks the promise on some input, then N does not need to consider M_i and may simply abstain from working against M_i. We give N one bit of advice at each input length to indicate if performing the simulations at that length would cause N to break the promise. If the advice bit is 1, then N/α performs the simulation. If the advice bit is 0, N/α abstains by immediately rejecting.

As N is allowed one bit of advice, M_i should also be allowed at least one advice bit. With M_i allowed one bit of advice, N now has two different machines at each input length that it is concerned with – $M_i/0$ and $M_i/1$. N should perform a given simulation if at least one of these behaves appropriately and copies N's behavior. This can be done by giving N two advice bits – one each to indicate whether each of $M_i/0$ and $M_i/1$ behaves appropriately and copies N's behavior on inputs of one larger length. In general, if M_i is allowed $a(n)$ bits of advice, N would require $2^{a(n+1)}$ advice bits to specify whether M_i with each advice string behaves appropriately and copies N's behavior on inputs of one larger length. The construction of section 3 avoided this problem by considering a particular behavior that M_i might have – computing a hard language – and using this behavior to handle M_i with many advice strings at once. This entailed a recovery procedure for the hard language, a process that does not apply to generic semantic models. In this section, we use a different approach that does apply to generic semantic models, which can be thought of as a copying scheme that allows N to spread the $2^{a(n+1)}$ advice bits needed to appropriately simulate M_i at a given length over many input lengths.

Consider the simulations of M_i at length n_i^* which N is responsible for copying to smaller padded inputs. We would like to give N one advice bit for each of M_i's possible advice strings at length n_i^*, indicating for each whether M_i with that advice string behaves appropriately. We spread these advice bits across multiple input lengths. That is, for each of M_i's possible advice strings b at length n_i^*, we allocate a distinct slightly smaller input length from which N is responsible for simulating M_i/b at length n_i^*. For the input length responsible for advice string b, N's advice bit is set to indicate if M_i/b behaves appropriately at length n_i^*. If the advice bit is 1, N/α performs the simulation of M_i/b at length n_i^*. If the advice bit is 0, N abstains by immediately rejecting. Now N/α satisfies the promise on all inputs, and for each advice string that causes M_i to appropriately copy N's behavior at length n_i^*, N/α copies that behavior to a slightly smaller input length.

As with delayed diagonalization on syntactic models, we repeat the same process to copy the behavior at length n_i^* to smaller and smaller inputs. This is best visualized by a tree of input lengths with n_i^* being the root node. The tree node corresponding to n_i^* has one child input length for each possible advice string at length n_i^* as described above. Each of these input lengths is also considered a node of the tree of input lengths with as many children as different advice strings at that length. This is repeated until reaching a level of leaf nodes. The tree of input lengths is

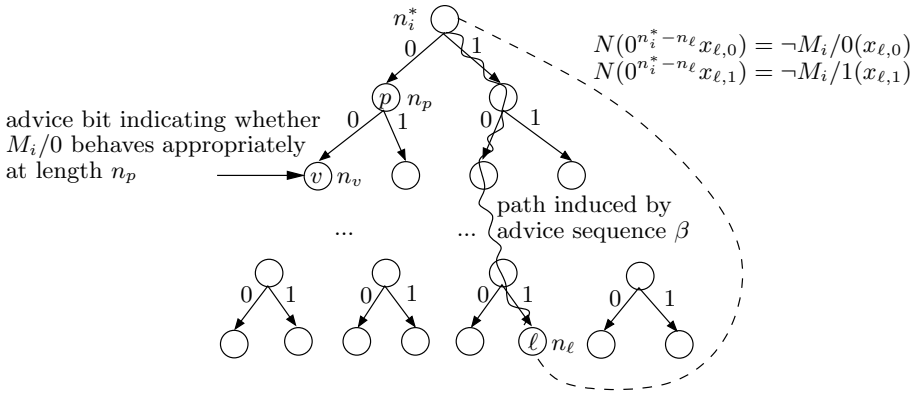

Figure 6: Illustration of N's execution for generic semantic models, shown for the case where M_i receives 1 bit of advice. Solid lines indicate that on the smaller input, N simulates M_i on padded inputs of the larger length, using the advice bit specified on the arrow. The dashed line indicates that on padded inputs of length n_i^*, N complements the behavior of M_i on inputs corresponding to the leaves of the tree of input lengths.

illustrated in Figure 6. We now give more details on the construction.

First consider an internal node corresponding to some input length n_p. This node must have a child node for all possible advice strings at length n_p. Each of these child nodes is responsible for simulating M_i on inputs of length n_p using a different advice string. Let n_v be a child node of node n_p that is responsible for simulating M_i with advice string b. The advice string b can be efficiently computed from the input length n_v – we describe an encoding scheme with this property in the next section. N's advice bit at length n_v indicates whether M_i/b behaves appropriately at length n_p. If the advice bit is 1, then on inputs x of length n_v, N simulates $M_i(0^{n_p-n_v}x)/b$; otherwise, N abstains and rejects all inputs of length n_v.

Consider an input length n_ℓ that corresponds to a leaf node ℓ in the tree. It is the responsibility of the root node of the tree to complement the behavior of M_i on inputs of length n_ℓ for all possible advice strings for input length n_ℓ. The complementation is realized using inputs $x_{\ell,b}$ of length n_ℓ for each possible advice string b at length n_ℓ. The inputs are chosen in such a way that they are distinct for all leaf nodes ℓ and advice strings b and such that they remain distinct when they are padded with zeros to length n_i^*. In particular, we set $x_{\ell,b} = 10^{n_\ell-1-|b|}b$, and $N(0^{n_i^*-n_\ell}x_{\ell,b})$ complements $M_i(x_{\ell,b})/b$. Note that n_i^* must be large enough so that space $s(n_i^*)$ suffices for N to *safely complement* the behavior of M_i on all leaf nodes. By the latter we mean that N satisfies the promise no matter M_i's behavior and that N complements the behavior of M_i on leaf nodes for which M_i behaves appropriately. A safe complementation in general incurs a blowup in space, even for models such as two-sided error machines which are closed under complementation, because N must avoid breaking the promise when working against a machine M_i which does. One way to achieve this is for N at length n_i^* to deterministically simulate M_i at the leaf nodes and flip the result. For Arthur-Merlin games this can be accomplished with an exponential blowup in space, and for unambiguous machines a quadratic blowup is sufficient.

On all input lengths in $[n_i, n_i^*]$ that are not used in the tree of input lengths, N acts trivially by rejecting all inputs of that length.

We claim that N/α constructed in this way satisfies the promise on all inputs and differs

from M_i/β for all machines M_i and advice sequences β for which M_i/β behaves appropriately. N/α satisfies the promise on all inputs by setting the advice bits appropriately on all nodes of the tree. Suppose there is an advice sequence β causing M_i to compute the same language as N while satisfying the promise on all inputs and using $s(n)$ space. The construction of the tree guarantees that there is a chain of inputs present in the tree for this advice sequence from the root node down to a leaf node. If we assume M_i/β computes the same language as N on all these inputs, then the complementary behavior initiated at the root node is copied down all the way to the leaf node, which is impossible. More precisely, let h be the height of the tree and $n_i^* = n_{i,h} > n_{i,h-1} > n_{i,h-2} > \ldots > n_{i,0} = n_\ell$ denote the path from the root of the tree to the leaf ℓ induced by β. By construction, we have for $b = \beta_{n_\ell}$ that

$$\begin{aligned}\neg M_i(x_{\ell,b})/b = N(0^{n_{i,h}-n_\ell}x_{\ell,b})/\alpha = M_i(0^{n_{i,h}-n_\ell}x_{\ell,b})/\beta_{n_{i,h}} = \\ N(0^{n_{i,h-1}-n_\ell}x_{\ell,b})/\alpha = M_i(0^{n_{i,h-1}-n_\ell}x_{\ell,b})/\beta_{n_{i,h-1}} = \ldots = \\ N(0^{n_{i,1}-n_\ell}x_{\ell,b})/\alpha = M_i(0^{n_{i,1}-n_\ell}x_{\ell,b})/\beta_{n_{i,1}} = N(x_{\ell,b})/\alpha = M_i(x_{\ell,b})/b,\end{aligned}$$

which is a contradiction. We conclude that N/α succeeds in differing from each machine M_i which satisfies the promise and uses at most $s(n)$ space on all inputs. It remains to show that N needs space not much more than $s(n)$ and determine the amount of advice the construction can handle.

4.1 Analysis

In this section, we give remaining details of the construction of the copying tree, ensuring N/α uses small space and determining the amount of advice bits that can be given M_i, proving Theorems 3, 4, and 5.

For clarity we focus on the case where $s(n) = \log n$ for now; we consider larger space bounds at the end of this section. Let $a(n)$ denote the amount of advice we allow M_i, and let $\sigma(n)$ be the smallest value such that $\log n$ space computations can be complemented within the model using $\sigma(n)$ space. To ensure that N/α requires not much more than $\log n$ space, we must balance two competing requirements – that n_i^* is large enough to be able to efficiently complement the behavior of the leaf nodes, and that each node in the tree is close enough to its parent node to be able to simulate it efficiently.

Each node in the tree corresponds to some input length in the interval $[n_i, n_i^*]$, where n_i^* corresponds to the root of the tree. We separate the tree into consecutive levels. We call the bottom-most level of leaf nodes "level 0", its parent nodes "level 1", and so on. Let h denote the number of non-leaf levels in the tree, so the root node at input length n_i^* is at level h.

To ensure the simulations take $O(\log n)$ space, we impose the restriction that a node n_v's parent n_p can correspond to an input length that is only polynomially larger: N incurs only a constant factor overhead in simulating M_i, and if M_i uses space at most $\log n$ and $n_p \leq n_v^c$ for some constant c, then the simulation requires $O(\log n_p) = O(\log(n_v^c)) = O(\log n_v)$ space. To ensure each node is separated by its parent's input length by at most a polynomial amount, we embed each level of the tree within an interval that is polynomially long. That is, for each $j = 0, 1, ..., h-1$, we embed level j of the tree in the interval $[n_i^{c^j}, n_i^{c^{j+1}})$ for some constant c to be chosen later. Because each internal node must have as many children as possible advice strings at that length, each internal node in the tree would have a different degree. We simplify the construction and analysis by rounding up the amount of advice given to M_i to ensure that all nodes in the same level have the same degree. That is, all nodes in level j have degree $2^{a(n_i^{c^{j+1}})}$.

For completeness, we give the encoding scheme that identifies which input lengths in the tree correspond to a given node's children. Consider an input length n that is an internal node at level j in the tree, so $n = n_i^{c^j} + \Delta$ for some $\Delta < n_i^{c^{j+1}} - n_i^{c^j}$. We must specify which input lengths in level $j-1$ correspond to n's children for each advice string of length $a(n_i^{c^{j+1}})$. We use the most obvious encoding scheme, filling in the children for level j nodes from left to right within level $j-1$. That is, n's child corresponding to advice string b is at input length $n_i^{c^{j-1}} + 2^{a(n_i^{c^{j+1}})} \cdot \Delta + b$. This encoding scheme allows N to efficiently determine where any given input length falls within the tree, so N can efficiently determine which padded input and with which advice string it is to simulate M_i.

The above encoding scheme can only be realized if the interval $[n_i^{c^j}, n_i^{c^{j+1}})$ contains as many input lengths as there are nodes in level j of the tree, for each $j = 0, 1, 2, ..., h-1$. The bottom-most level contains the largest number of nodes and has the smallest number of input lengths to work with, so the tree can be embedded into $[n_i, n_i^*]$ exactly when the bottom-most level fits within the interval $[n_i, n_i^c)$. Because we have rounded up the degrees of the nodes, we get a simple expression for the number of leaf nodes in the tree: $2^{a(n_i^{c^t})} \prod_{j=2}^{t} 2^{a(n_i^{c^j})}$. By taking logarithms, there are enough input lengths in level 0 for these nodes exactly when

$$a(n_i^{c^h}) + \sum_{j=2}^{h} a(n_i^{c^j}) \leq \log(n_i^c - n_i). \tag{1}$$

Now consider the space usage of the construction. We have already guaranteed the simulations represented by the tree can be performed using $O(\log n)$ space. We must also ensure that the root node operates in $O(\log n_i^*)$ space. Because the root must complement all leaf nodes, the root node runs in $O(\log n_i^*)$ space if

$$\log n_i^* = \Omega(\sigma(n_i^c)). \tag{2}$$

If we can simultaneously satisfy both (1) and (2), we ensure the construction can be implemented correctly and in space $s'(n)$ for any $s'(n) = \omega(\log n)$. We now finish the analysis separately for two cases.

1. For some semantic models, such as Arthur-Merlin games, the most efficient safe complementation known within the model incurs an exponential blowup in space. We handle such models using Theorem 3.

2. For some semantic models, such as unambiguous machines, a safe complementation within the model is known with only a polynomial blowup in space. We handle these models using Theorem 4.

Complementation with Exponential Blowup (Theorem 3)

We first complete the analysis for the more general setting where there is a safe complementation within the model with an exponential blowup in space, which is typically achieved by using a deterministic simulation of the model and flipping the result. We now assume a semantic model where $\log n$ space computations can be complemented within the model in space $O(n^{d'})$ for some constant d'. In this case, (2) becomes

$$\log n_i^* = \log n_i^{c^h} = \Omega(n_i^{cd'}). \tag{3}$$

In other words, $n_i^* = 2^{O(n_i^{cd'})}$, and we must set $h = \lceil \log(\frac{n_i^{cd'}}{\log n_i}) / \log c \rceil = \Omega(\log n_i)$ to ensure (3). To fit the leaves of a tree that has depth $\Omega(\log n_i)$ within the interval $[n_i, n_i^c)$, the degree at each node can be at most some constant. Let $a(n) = k$ for some constant k. Then (1) becomes

$$k + \sum_{j=2}^{h} k = h \cdot k \leq \log(n_i^c - n_i). \tag{4}$$

As the right-hand side grows faster with c than the left-hand side, we can pick c sufficiently large so that both (3) and (4) are satisfied. The construction works for any constant k, and we have shown that N/α uses $O(\log n)$ space where the constant depends on M_i and k. Standard techniques similar to those used in section 3.3 suffice to conclude Theorem 3 for the case of semantic models such as Arthur-Merlin games. Section 4.2 contains a definition of the precise properties needed of a semantic model for Theorem 3 to hold.

Complementation with Polynomial Blowup (Theorem 4)

We now complete the analysis for semantic models where there is a safe complementation within the model with only a polynomial blowup in space. We assume now that M_i's behavior at length n while using space $\log n$ can be complemented within the model using $\sigma(n) = O(\log^d n)$ space. For example, $d = 2$ for unambiguous machines. Thus (2) becomes $\log n_i^* = \Omega(\log^d(n_i^c))$, or equivalently, $n_i^* = 2^{\Omega(\log^d(n_i^c))}$. Now consider the first term of (1). Plugging in the above equality for n_i^* tells us that we must at least satisfy $a(2^{\Omega(\log^d(n_i^c))}) < \log(n_i^c)$ if we are to satisfy (1). This imposes an upper bound on $a(n)$ of $O(\log^{1/d} n)$.

In fact we can achieve $a(n) = \Theta(\log^{1/d} n)$ while still satisfying both (1) and (2), as follows. Let $a(n) = k \log^{1/d} n$ for some integer $k > 0$. Substituting into (1) yields

$$k \log^{1/d}(n_i^{c^h}) + k \sum_{j=2}^{h} \log^{1/d}(n_i^{c^j}) \leq \log(n_i^c - n_i). \tag{5}$$

For technical reasons, we aim to satisfy (2) by ensuring

$$c^3 \log n_i^* = c^3 \log(n_i^{c^h}) \geq \log^d(n_i^c), \tag{6}$$

which we satisfy by setting $h = \lceil (\log(c^{d-3} \log^{d-1} n_i) / \log c \rceil$.

Using the fact that $h \leq \frac{\log(c^{d-3} \log^{d-1}(n_i))}{\log c} + 1$, we bound the first term of the left-hand side of inequality (5).

$$k \log^{1/d}(n_i^{c^h}) = k(c^h \log n_i)^{1/d} \leq k(c^{d-2} \log^d n_i)^{1/d} = kc^{(d-2)/d} \log n_i.$$

Assuming we pick c large enough such that $c^{1/d} - 1 \geq 1$, we now bound the second term.

$$\begin{aligned} k \textstyle\sum_{j=2}^{h} \log^{1/d}(n_i^{c^j}) = \;& k \tfrac{c^{2/d}(c^{(h-1)/d}-1)}{c^{1/d}-1} \log^{1/d} n_i && \leq kc^{2/d}(c^{h-1})^{1/d} \log^{1/d} n_i \\ \leq \;& kc^{2/d}(c^{d-3} \log^{d-1} n_i)^{1/d} \log^{1/d} n_i && = kc^{(d-1)/d} \log n_i. \end{aligned}$$

Adding up these two values satisfies inequality (5) for large enough c.

We have shown that the space usage of N/α is $O(\log n)$ where the constant depends on M_i and k. Using standard techniques similar to those used in section 3.3, $s'(n)$ space is sufficient for

N/α for any $s'(n) = \omega(\log n)$, completing the proof of Theorem 4 for semantic models such as unambiguous machines. Section 4.2 contains a definition of the precise properties required of a model for Theorem 4 to hold.

Larger Space Bounds (Theorem 5)

So far we have only considered the case with $s(n) = \log n$, where we have shown separation results that are tight with respect to space – that $s'(n)$ space suffices to differ from $s(n)$ space machines for any $s'(n) = \omega(s(n))$. Tightness with respect to space follows from satisfying: (1) each node of the copying tree is close enough to its parent so the simulations incur only a constant overhead in space, and (2) nodes are far enough apart so the height of the tree required to allow the root node to complement leaf nodes does not result in more leaf nodes than input lengths allocated in the bottom-most level of the copying tree. In the general setting where safe complementation requires an exponential blowup in space, these cannot be simultaneously met for super-logarithmic space bounds – our construction still works but gives a result that is not tight with respect to space for $s(n) = \omega(\log n)$.

In the setting where safe complementation incurs only a polynomial blowup in space, we have more wiggle room and can derive a tight separation for space bounds up to any polynomial. In fact, an examination of the analysis for Theorem 4 shows the construction as given remains tight with respect to space for $s(n)$ any poly-logarithmic function. For larger space bounds the construction as given is not tight. To handle these the main idea is to place nodes of the copying tree closer to their parent nodes to satisfy (1); this can be achieved for space bounds up to polynomial without breaking (2).

We now prove Theorem 5. Fix a semantic model where M_i's behavior while it uses $s(n)$ space can be complemented within the model using space $O(s(n)^d)$. We first consider space bounds of the form $s(n) = n^r$ for r a rational constant. We would like to demonstrate a language computable within the model using $s'(n)$ space and one bit of advice that is not computable using $s(n)$ space and $O(1)$ bits of advice, for any $s'(n) = \omega(s(n))$. As alluded to above, we accomplish this by modifying the generic construction so that each level of the copying tree is embedded within a smaller interval of input lengths: we embed level j of the copying tree within input lengths $[c^j n_i, c^{j+1} n_i)$ where c is a constant we may choose. This ensures that for each n_v, $n_p \leq c \cdot n_v$ and performing the simulation of M_i on inputs of length n_p uses space $O(n_p^r) \leq O((c \cdot n_v)^r) = O(c^r n_v^r) = O(n_v^r) = O(s(n_v))$. Let h be the height of the copying tree. To ensure the root node has sufficient space to complement the leaf nodes, it must be that

$$(c^h n_i)^r = \Omega(((c \cdot n_i)^r)^d),$$

which we achieve by setting $h = \lceil \log(n_i^{d-1}) / \log c \rceil$. If M_i is allowed k advice bits the total number of leaf nodes is $2^{h \cdot k} = n_i^{k(d-1)/\log c}$, which must be smaller than $c \cdot n_i - n_i$ to ensure the leaf nodes fit within the range of input lengths we have allocated for them. We can choose c large enough to ensure this holds, which combined with standard techniques similar to those of section 3.3 gives Theorem 5 for space bounds of the form $s(n) = n^r$.

Having derived Theorem 5 for polynomial space bounds, we derive the result for smaller space bounds using a simple padding argument.

Let $s(n)$ and $s'(n)$ be space bounds such that $s'(n) = o(n^r)$ for a rational constant r, $s'(n) = \omega(s(n))$, and $s(n) = \Omega(\log n)$. Suppose by way of contradiction that all languages computable within the model with $s'(n)$ space, and therefore also $O(s'(n))$ space, and one bit of advice are

computable with $s(n)$ space and $O(1)$ bits of advice. We use this assumption to violate the separation already proved for the space bound n^r. The argument above gives a language computable with n^r space and one bit of advice that is not computable with $o(n^r)$ space and $O(1)$ bits of advice. Let L be such a language.

Let $m(n)$ be the smallest integer such that $s'(m(n)) \geq n^r$, and define the language $L_{pad} = \{0^{m(n)-1-n}\#x | x \in L\}$. Assuming $s'(n)$ is space-constructible L_{pad} can be decided in space $O(s'(n))$ with one bit of advice, which by assumption means L_{pad} is also computable in space $s(n)$ with $O(1)$ bits of advice by some machine M. Consider the following method for computing L: on input x of length n, form the input $y = 0^{m(n)-1-n}\#x$, simulate $M(y)$ and output the result. The simulation of $M(y)$ takes $s(m(n)) = o(s'(m(n)))$ space and requires $O(1)$ bits of advice. Assuming $s'(n)$ is constructible in space $o(s'(n))$, computing the padded input also takes $o(s'(m(n)))$ space. Assuming $s'(n+1) = O(s'(n))$ for all n, $s'(m(n)) = O(n^r)$. Thus we have reached the contradiction that L can be decided with space $o(n^r)$ and $O(1)$ bits of advice, meaning our original assumption was false and Theorem 5 is proved.

4.2 Generic Semantic Models

Consider the properties of the machine model used in the above analysis of Theorems 3, 4, and 5. First, N can simulate any other machine M_i with only a constant factor overhead in space. This is needed to ensure that N needs only slightly more space than M_i. Second, N can efficiently perform certain deterministic tasks – e.g., for an input of length n, N performs arithmetic to determine which interval of inputs $[n_i, n_i^*]$ and which node within the copying tree n corresponds to. Third, the analysis of section 4.1 was broken up into two cases depending on the efficiency with which complementation is possible within the model. Semantic and syntactic models can in general be complemented with an exponential blowup in space, and then Theorem 3 applies. The following is a precise definition of the properties required for Theorem 3 to hold.

Definition 1. *Let $(M_i)_{i=1,2,3,\ldots}$ be a computable enumeration associated with a semantic model of computation. The semantic model is called* reasonable *if it satisfies the following conditions:*

1. *There exists an efficient universal machine U such that for each $i \geq 1$, $x \in \{0,1\}^*$, and $s \geq s_{M_i}(x)$, U satisfies the promise on input $\langle M_i, x, 0^s \rangle$ whenever M_i satisfies the promise on input x, and if so, $U(\langle M_i, x, 0^s \rangle) = M_i(x)$. U must run in space $O(s + \log(|x| + |M_i|))$.*

2. *Let D be a deterministic transducer, i.e. a deterministic machine D that executes and either outputs an answer $a(x)$ or a query $q(x)$ to some machine M. For each such D and machine M_i, there must exist a machine $M_{i'}$ such that on each input x: if $D(x)$ outputs an answer $a(x)$, then $M_{i'}(x) = a(x)$ and satisfies the promise; and if $D(x)$ outputs a query $q(x)$ on which M_i satisfies the promise, then $M_{i'}(x) = M_i(q(x))$ and satisfies the promise. In addition, the space usage of $M_{i'}(x)$ must be $O(s_D(x))$ when $D(x)$ outputs an answer, and must be $O(s_D(x) + s_{M_i}(q(x)))$ when $D(x)$ outputs a query $q(x)$.*

   If this holds, we say the model is efficiently closed under deterministic transducers.

3. *The efficient universal machine U can be safely complemented with an exponential blowup in space. That is, there is a machine S that satisfies the promise on every input, and such that $S(x) = \neg U(x)$ for every input $x \in \{0,1\}^*$ on which U satisfies the promise. $S(x)$ runs in space $2^{O(s(x))}$, where $s(x)$ is the space used by $U(x)$.*

As pointed out in the introduction, Definition 1 includes a wide class of semantic models, and in particular includes models such as Arthur-Merlin games for which the simple translation argument of [KV87] does not apply.

Theorems 4 and 5 apply to any reasonable semantic model that satisfies the first two conditions of Definition 1 and can be complemented more efficiently, as follows.

Definition 2. *Fix a semantic model that is reasonable according to Definition 1. We call the model* easily complementable *if there exists a safe complementation S of $U(\langle M_i, x, 0^s\rangle)$ that runs in space $O(s^d(n) + \log(n + |M_i|))$ for some rational constant $d \geq 1$.*

Note that due to the space-bounded derandomization of [SZ99], randomized two-sided, one-sided, and zero-sided error machines are easily complementable models of computation with $d = 3/2$. Nondeterministic and unambiguous machines are easily complementable with $d = 2$ due to Savitch's Theorem. We point out that Arthur-Merlin games are unlikely to be easily complementable as a deterministic simulation of Arthur-Merlin games with polynomial overhead in space would imply that NC lies in DSPACE($\log^d n$) for some constant d [FL93].

5 Promise Problems

We have proved space hierarchy theorems for semantic models by allowing one bit of advice. Hierarchy theorems can also be proved for semantic models by relaxing the requirement that a machine must satisfy the promise on all inputs. Formally, for a model of computation defined by an underlying computable enumeration $(M_i)_{i=1,2,3,\ldots}$ and a promise, we define a promise problem within the model as a pair of sets (Π_Y, Π_N) that has the following properties: Π_Y are the set of "yes" instances while Π_N are the set of "no" instances, $\Pi_Y \cap \Pi_N = \emptyset$, and there is a machine $M \in (M_i)_{i=1,2,3,\ldots}$ that computes the correct value and satisfies the promise on all inputs in $\Pi_Y \cup \Pi_N$. Notice that M can behave arbitrarily for $x \notin \Pi_Y \cup \Pi_N$. We would like to use this property to prove a tight space hierarchy for the promise problems of semantic models

An examination of our proofs in the above sections shows that they yield hierarchies for the promise problems of semantic models. However, we shall see that promise hierarchy theorems can be proved without much of the complexity of our arguments – delayed diagonalization suffices to derive Theorem 6.

For concreteness, consider two-sided error randomized machines. A first attempt at proving the hierarchy is to use direct diagonalization. Namely, construct a diagonalizing machine that enumerates all probabilistic machines M_i, chooses a certain input x_i for machine M_i, and simulates $M_i(x_i)$ and does the opposite. But suppose $M_i(x_i)$ does not have bounded error. Then any promise associated with M_i must ignore input x_i, and the promise for our diagonalizing machine must as well since it simulates $M_i(x_i)$. As x_i has the same status with respect to both promise problems, we have not diagonalized against M_i after all.

Another complication arises when considering promise problems. In the context of two-sided error for a randomized machine, the natural promise problem for that machine is to set $\Pi_Y = \{x | \Pr[M(x) = 1] \geq 2/3\}$ and $\Pi_N = \{x | \Pr[M(x) = 1] \leq 1/3\}$. However, there are many other valid promise problems that M decides by ignoring certain inputs even though M has bounded error on these. The diagonalizing machine we construct must work against each M_i in such a way that the promise problem we specify is different than any valid promise problem over M_i. We solve both this problem and the above by using delayed diagonalization, as follows.

Let N be the machine we build to diagonalize against promise problems computable by two-sided error space $s(n)$ machines. For each probabilistic machine M_i, we allocate an interval of input lengths $[n_i, n_i^*]$ on which to diagonalize against M_i. The first part of the construction is a delayed complementation, which is achieved on input $0^{n_i^*}$. Let n_i^* be large enough so that N can deterministically compute the acceptance probability of $M_i(0^{n_i})$ using space $s(n_i^*)$. $N(0^{n_i^*})$ should do the opposite of $M_i(0^{n_i})$. This is ensured by placing $0^{n_i^*}$ within the promise of N and having $N(0^{n_i^*})$ output 1 with probability 1 if $\Pr[M_i(0^{n_i}) = 1] < \frac{1}{2}$, and output 0 with probability 1 otherwise. Notice that regardless of the status of $M_i(0^{n_i})$ in terms of a promise problem (either n_i is in Π_Y, Π_N, or neither), $N(0^{n_i^*})$ does something different.

The second part of the construction copies down the complementary behavior to smaller and smaller padded inputs. On input 0^{n_i+j} for $0 \leq j < n_i^* - n_i$, N simulates $M_i(0^{n_i+j+1})$ while it uses at most $s(n)$ space, and we define N's promise to be the natural one on each of these inputs – the input is within the promise (either Π_Y or Π_N) when its probability of acceptance is either at least $2/3$ or at most $1/3$. On inputs other than those of the form 0^{n_i+j}, N rejects and halts immediately (these inputs are not used in the diagonalization).

Suppose there is a promise problem defined on M_i with M_i using at most $s(n)$ space which computes the same promise problem as N on all inputs in the interval $[n_i, n_i^*]$. Because $0^{n_i^*}$ is in the promise of N, this is also true for M_i. $N(0^{n_i^*-1})$ by construction simulates $M_i(0^{n_i})$, and an input has been defined to be in the promise of N iff N has bounded error on the input. So $0^{n_i^*-1}$ is in the promise of N, and therefore must also be in the promise of M_i. If we continue this argument through the entire interval, we conclude that each 0^{n_i+j} is contained within the promise of both N and M_i for $j = 0, 1, ..., n_i^* - n_i$. By the assumption that M_i computes the same promise as N, the fact that each input is in the promise of M_i and N, and the construction of N to simulate M_i, we have the following set of equalities:

$$\begin{aligned} M_i(0^{n_i}) = &\quad N(0^{n_i}) = M(0^{n_i+1}) = N(0^{n_i+1}) = M_i(0^{n_i+2}) \\ = &\quad ... = M_i(0^{n_i^*-1}) = N(0^{n_i^*-1}) = M_i(0^{n_i^*}) = N(0^{n_i^*}). \end{aligned}$$

However, we have constructed $N(0^{n_i^*})$ so that it explicitly differs from $M_i(0^{n_i})$: if 0^{n_i} is in the promise of M_i, then N flips the output; otherwise 0^{n_i} is not in the promise of M_i even though $0^{n_i^*}$ is in the promise of N. In either case, $N(0^{n_i^*}) \neq M_i(0^{n_i})$ where $\neq$ means the promise problem is different on each. We have reached a contradiction, so there can be no promise problem defined on M_i that corresponds to the natural promise problem of N. Further, standard techniques guarantee that $s'(n)$ space is sufficient for N to carry out this construction against all probabilistic machines M_i, for any $s'(n)$ with $s'(n) = \omega(s(n+1))$. Namely, equip N with a mechanism to ensure it never uses more than $s'(n)$ space, and use an enumeration of probabilistic machines where each machine appears infinitely often to ensure that for each machine M', at least once while working against M' the asymptotic behavior of s' and s has taken effect so that N successfully completes the construction against M'.

The above proof requires only a basic set of properties and holds for any reasonable semantic model as defined in section 4.2. The proof also gives time hierarchies for promise classes, yielding the following.

Theorem 7 (folklore). *Fix a reasonable model of computation. Let $t(n)$ and $t'(n)$ be time bounds with $t(n) = \Omega(n)$ and $t'(n)$ time constructible. If $t'(n) = \omega(t(n+1) \cdot \log t(n+1))$ then there is a promise problem computable within the model using time $t'(n)$ that is not computable as a promise problem within the model using time $t(n)$.*

Acknowledgments

We thank Scott Diehl for many useful discussions, in particular pertaining to the proof of Theorem 2. We also thank the anonymous reviewers for their time and suggestions.

References

[Bar02] Boaz Barak. A probabilistic-time hierarchy theorem for slightly non-uniform algorithms. In *Workshop on Randomization and Approximation Techniques in Computer Science*, 2002.

[BJLR91] Gerhard Buntrock, Birgit Jenner, Klaus-Jorn Lange, and Peter Rossmanith. Unambiguity and fewness for logarithmic space. In *Fundamentals of Computation Theory*, pages 168–179, 1991.

[Con93] Anne Condon. The complexity of space bounded interactive proof systems. In Steven Homer, Uwe Schöning, and Klaus Ambos-Spies, editors, *Complexity Theory: Current Research*, pages 147–190. Cambridge University Press, 1993.

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

[FL93] Lance Fortnow and Carsten Lund. Interactive proof systems and alternating time-space complexity. *Theoretical Computer Science*, 113(1):55–73, 1993.

[FS04] Lance Fortnow and Rahul Santhanam. Hierarchy theorems for probabilistic polynomial time. In *IEEE Symposium on Foundations of Computer Science*, pages 316–324, 2004.

[FST05] Lance Fortnow, Rahul Santhanam, and Luca Trevisan. Hierarchies for semantic classes. In *ACM Symposium on the Theory of Computing*, pages 348–355, 2005.

[GST04] Oded Goldreich, Madhu Sudan, and Luca Trevisan. From logarithmic advice to singe-bit advice. Technical Report TR-04-093, Electronic Colloquium on Computational Complexity, 2004.

[Imm88] Neil Immerman. Nondeterministic space is closed under complementation. *SIAM Journal of Computing*, 17(5):935–938, 1988.

[KL82] Richard Karp and Richard Lipton. Turing machines that take advice. *L'Enseignement Mathématique*, 28(2):191–209, 1982.

[KV87] Marek Karpinski and Rutger Verbeek. Randomness, provability, and the separation of Monte Carlo time and space. In *Computation Theory and Logic*, pages 189–207, 1987.

[MP07] Dieter van Melkebeek and Konstantin Pervyshev. A generic time hierarchy for semantic models with one bit of advice. *Computational Complexity*, 16:139–179, 2007.

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

[SZ99] Michael Saks and Shiyu Zhou. $BP_HSPACE(S) \subseteq DSPACE(S^{3/2})$. *Journal of Computer and System Sciences*, 58:376–403, 1999.

[Sze88] Róbert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.

[Wat03] John Watrous. On the complexity of simulating space-bounded quantum computations. *Computational Complexity*, 12:48–84, 2003.

[Žàk83] Stanislav Žàk. A Turing machine time hierarchy. *Theoretical Computer Science*, 26:327–333, 1983.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de/