Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 147 (2010)

Time-Space Efficient Simulations of Quantum Computations

Dieter van Melkebeek* Thomas Watson†

January 12, 2012

Abstract

We give two time- and space-efficient simulations of quantum computations with intermediate measurements, one by classical randomized computations with unbounded error and the other by quantum computations that use an arbitrary fixed universal set of gates. Specifically, our simulations show that every language solvable by a bounded-error quantum algorithm running in time t and space s is also solvable by an unbounded-error randomized algorithm running in time $O(t \cdot \log t)$ and space $O(s + \log t)$, as well as by a bounded-error quantum algorithm restricted to use an arbitrary universal set and running in time $O(t \cdot \operatorname{polylog} t)$ and space $O(s + \log t)$, provided the universal set is closed under adjoint. We also develop a quantum model that is particularly suitable for the study of general computations with simultaneous time and space bounds.

As an application of our randomized simulation, we obtain the first nontrivial lower bound for general quantum algorithms solving problems related to satisfiability. Our bound applies to MajSAT and MajMajSAT, which are the problems of determining the truth value of a given Boolean formula whose variables are fully quantified by one or two majority quantifiers, respectively. We prove that for every real d and every positive real δ there exists a real $c > 1$ such that either

- MajMajSAT does not have a bounded-error quantum algorithm running in time $O(n^c)$, or
- MajSAT does not have a bounded-error quantum algorithm running in time $O(n^d)$ and space $O(n^{1-\delta})$.

In particular, MajMajSAT does not have a bounded-error quantum algorithm running in time $O(n^{1+o(1)})$ and space $O(n^{1-\delta})$ for any $\delta > 0$. Our lower bounds hold for any reasonable uniform model of quantum computation, in particular for the model we develop.

*Department of Computer Sciences, University of Wisconsin – Madison. Partially supported by NSF awards CCF-0728809 and CCF-1017597, and by the Humboldt Foundation.

†Computer Science Division, University of California, Berkeley. Part of the research was performed while at the University of Wisconsin – Madison. Partially supported by a Computer Science Department Summer Research Fellowship and a Barry M. Goldwater Scholarship while at the University of Wisconsin – Madison; supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-0946797 and by the National Science Foundation under Grant No. CCF-1017403 while at the University of California, Berkeley.

ISSN 1433-8092

1 Introduction

Motivated by an application to time-space lower bounds, we establish two efficient simulations of quantum computations with simultaneous time and space bounds. Our first result shows how to simulate quantum computations with intermediate measurements by classical randomized computations with unbounded error in a way that is both time- and space-efficient. For bounded-error quantum computations our simulation only incurs a logarithmic factor overhead in time and a constant factor overhead in space. Modulo some minor technicalities, this simulation subsumes and improves all previously known simulations of bounded-error quantum computations by unbounded-error randomized computations.

Theorem 1 (Randomized Simulation). *Every language solvable by a bounded-error quantum algorithm running in time $t \geq \log n$ and space $s \geq \log n$ with algebraic transition amplitudes is also solvable by an unbounded-error randomized algorithm running in time $O(t \cdot \log t)$ and space $O(s + \log t)$, provided t and s are constructible by a deterministic algorithm with the latter time and space bounds.*

In fact, Theorem 1 holds more generally for transition amplitudes that satisfy a certain mild approximability condition (see Theorem 4 and Theorem 5 in Section 3.1, and Definition 1 and Definition 2 in Section 2.3.2).

Our second simulation deals with the quantum compiling problem: given a quantum computer implementation that has a fixed universal library of unitary gates, and given a quantum algorithm with an arbitrary library specified by the algorithm designer, compile the algorithm into a form that can run on the available computer. We show how to do this in a way that is both time- and space-efficient as long as the universal set is closed under adjoint. For bounded-error quantum computations our simulation only incurs a polylogarithmic factor overhead in time and a constant factor overhead in space. This is the first rigorous result on quantum compiling in a model of computation with finite-precision arithmetic, and it strengthens the well-known Solovay-Kitaev Theorem [25] by reducing the space bound to the natural barrier imposed by the numerical nature of the algorithm.

Theorem 2 (Quantum Simulation). *For every universal set S of unitary gates with algebraic entries that is closed under adjoint, every language solvable by a bounded-error quantum algorithm running in time t and space s with algebraic transition amplitudes is also solvable by a bounded-error quantum algorithm running in time $O(t \cdot \mathrm{polylog}\, t)$ and space $O(s + \log t)$ whose library of gates is S, provided t is constructible by a deterministic algorithm with the latter time and space bounds.*

Like Theorem 1, Theorem 2 holds more generally under a mild approximability condition on the transition amplitudes and the entries of the gates in S (see Theorem 6 in Section 4.1 and Definition 1 in Section 2.3.2).

For such fine-grained simulations to be meaningful, we need a model of quantum computation that allows us to accurately measure time and space complexity simultaneously. Another contribution of this paper is the development of such a model. The existing models give rise to various issues. For example, intermediate measurements are needed for time-space efficient simulations of randomized computations by quantum computations. Several of the known models only allow measurements at the end of the computation but not during the computation. As another example, the known time-space lower bounds for classical algorithms (see [37] for a survey) hold for models with random access to the input and memory. This makes the lower bounds more meaningful as they do not exploit artifacts due to sequential access. Extending the standard quantum Turing machine model [8] to accommodate random access leads to complications that make the model inconvenient to work with. In Section 2 we discuss these and other issues, survey the known models from the literature, and present a model that addresses all the issues we raise and is capable of efficiently simulating all currently envisioned realizations of quantum computing.

We point out that our arguments for Theorem 1 and Theorem 2 are very robust with respect to the details of the model. However, our results are more meaningful for models that accurately reflect time and space, which our model does.

The starting point for the construction in Theorem 1 is a simulation due to Adleman et al. [1] (as streamlined by Fortnow and Rogers [19]), which is the only previously known time-efficient simulation. That simulation does not deal with intermediate measurements in a space-efficient way, and it incurs a polylogarithmic factor overhead in running time. We show how to handle intermediate measurements with only a constant factor overhead in space, moreover using only a logarithmic factor overhead in time. Reducing the space involves modifying the techniques of [19] to ensure that each bit of the sequence of coin flips is only referenced once (and thus no space needs to be used to remember it). We reduce the time by directly handling nonunitary approximations to the library gates, rather than appealing to unitary approximations (which incur a polylogarithmic factor time overhead). Also, our construction handles computations in which the sequence of local quantum operations can depend on previous measurement outcomes; such computations are seemingly more powerful than uniform quantum circuits. This result is developed in Section 3, where we also give a detailed comparison with earlier simulations.

The main component in the proof of Theorem 2 is a strengthening of the classic Solovay-Kitaev Theorem [25]. The latter theorem shows how to approximate any unitary quantum gate within ϵ using a sequence of at most $\mathrm{polylog}(1/\epsilon)$ gates from an arbitrary universal set (provided the set is closed under adjoint). To prove Theorem 2, we need a deterministic algorithm for computing such a sequence in time $\mathrm{polylog}(1/\epsilon)$ and space $O(\log(1/\epsilon))$, in a standard finite-precision model of computation in which every bit counts toward the complexity. If we allow space $\mathrm{polylog}(1/\epsilon)$ then such an algorithm can be gleaned from the proof of the Solovay-Kitaev Theorem by analyzing the precision needed for the numerical calculations. More ideas are needed to bring the space complexity down to the natural barrier of $O(\log(1/\epsilon))$ while maintaining a $\mathrm{polylog}(1/\epsilon)$ running time. Our improvement has two main components. First, we modify the algorithm's numerical core to combat the accumulation of error that is inherent in the conditioning. We make use of a known result from matrix theory. We also use an idea due to Nagy [30] which allows us to bypass the need for certain numerical calculations that would require too much precision when dealing with multi-qubit gates. Second, we modify the algorithm's overall architecture to save space. The simulation in Theorem 2 is then mostly a matter of applying our space-efficient algorithm to each unitary gate in the original quantum computation.[1] In Section 4 we explain the ideas behind our improvement of the Solovay-Kitaev Theorem, give the formal proof, and derive Theorem 2.

As an application of Theorem 1, we obtain the first nontrivial time-space lower bounds for quantum algorithms solving problems related to satisfiability, the seminal NP-complete problem. We show how to transfer the time-space lower bounds of Allender et al. [2] from classical unbounded-

[1] In particular, the space overhead in the simulation is only in classical bits, not qubits, since the simulation essentially just overlays some deterministic computation on the original quantum computation.

error randomized algorithms to bounded-error quantum algorithms. The lower bounds apply to analogues of satisfiability in the first two levels of the counting hierarchy, namely MajSAT and MajMajSAT (see [4] for an introduction to the counting hierarchy). MajSAT, short for majority-satisfiability, denotes the problem of deciding whether the majority of the assignments to a given Boolean formula satisfy the formula. Similarly, an instance of MajMajSAT asks whether a given Boolean formula depending on two sets of variables y and z has the property that for at least half of the assignments to y, at least half of the assignments to z satisfy the formula.

Theorem 3. *For every real d and every positive real δ there exists a real $c > 1$ such that either*

- MajMajSAT *does not have a bounded-error quantum algorithm running in time $O(n^c)$ with algebraic transition amplitudes, or*
- MajSAT *does not have a bounded-error quantum algorithm running in time $O(n^d)$ and space $O(n^{1-\delta})$ with algebraic transition amplitudes.*

In particular, Theorem 3 implies the following time-space lower bound for MajMajSAT.

Corollary 1. MajMajSAT *does not have a bounded-error quantum algorithm running in time $O(n^{1+o(1)})$ and space $O(n^{1-\delta})$ with algebraic transition amplitudes, for any $\delta > 0$.*

The quantitative strength of our lower bounds for MajSAT and MajMajSAT derives from [2]; thanks to the efficiency of Theorem 1, the translation does not induce any weakening. In contrast, none of the previously known simulations of bounded-error quantum computations by unbounded-error randomized computations are strong enough to yield nontrivial quantum lower bounds for problems related to satisfiability. In Section 5 we provide some more background on time-space lower bounds, derive Theorem 3, and present some directions for further research.

2 Models of Quantum Computation

In this section we develop the model that we use for the exposition of our arguments. Although we consider the development of such a model as a significant contribution of our paper, the crux of our main results can be understood at an abstract level. As such, a reader who would like to quickly get to the heart of our paper can skim Section 2.3 and then continue with Section 3 or with Section 4.

In Section 2.1 we discuss the issues that arise in choosing a model of quantum computation that accurately reflects time and space complexity. Section 2.2 describes how previously studied models fit into our taxonomy, and it can be skipped without loss of continuity. We motivate and precisely define our chosen model in Section 2.3.

2.1 Issues

Our model should capture the notion of a quantum algorithm as viewed by the computer science and physics communities and allow us to accurately measure the resources of time and space. For example, the model should allow us to express important quantum algorithms such as Shor's [35] and Grover's [21] in a way that is natural and faithfully represents their complexities. This forms the overarching issue in choosing a model. Below we discuss eight specific aspects of quantum computation models and describe how the corresponding issues are handled in the classical setting.

Sublinear space bounds. Many algorithms have the property that the amount of work space needed is less than the size of the input. Models such as one-tape Turing machines do not allow us to accurately measure the space usage of such algorithms because they charge for the space required to store the input. In the deterministic and randomized settings, sublinear space bounds are accommodated by considering Turing machines with a read-only input tape that does not count toward the space bound and read-write work tapes that do. In the quantum setting, we need a model with an analogous capability.

Random access to the input and memory. In order to accurately reflect the complexity of computational problems, our model should include a mechanism for random access, i.e., the ability to access any part of the input or memory in a negligible amount of time (say, linear in the length of the address). For example, there is a trivial algorithm for the language of palindromes running in quasilinear time and logarithmic space on standard models with random access, but to decide palindromes on a traditional sequential-access Turing machine with one head per tape, the time-space product needs to be at least quadratic. The latter result does not reflect the complexity of deciding palindromes, but rather exploits the fact that sequential-access machines may have to waste a lot of time moving their tape heads back and forth. Classical Turing machines can be augmented with a mechanism to support random access; our quantum model should also have such a mechanism.

Intermediate measurements. Unlike the previous two issues, intermediate measurements are specific to the quantum setting. In time-bounded quantum computations, it is customary to assume that all measurements occur at the end. This is because intermediate measurements can be postponed by introducing ancilla qubits to store (unitarily) what would be the result of the measurement, thus preventing computation paths with different measurement outcomes from interfering with each other. However, this has a high cost in space — a computation running in time t may make up to t measurements, so the space overhead could be as large as t, which could be exponential in the original space bound. This suggests that postponing measurements might be inherently costly in space (although we are not aware of any formal evidence for this). Hence, to handle small space bounds our model should allow intermediate measurements. This is crucial for our model to meet the expectation of being at least as strong as randomized algorithms with comparable efficiency parameters; the standard way to "flip a coin" in the quantum setting is to apply a Hadamard gate to a qubit in a basis state and then measure it. Also, many quantum algorithms are naturally described using intermediate measurements.

We also need to decide which measurements to allow. Projective measurements in the computational basis are the most natural choice. Should we allow projective measurements in other bases? How about fully general measurements (see Section 2.2.3 in [31]), where the measurement operators need not be projections? General measurements can be performed by introducing ancilla qubits (at a cost in space), performing a change of basis (at a cost in time), and doing a projective measurement in the computational basis, one qubit at a time. It is reasonable to charge the complexity of these operations to the algorithm designer, so we are satisfied with allowing only single-qubit measurements in the computational basis.

Obliviousness to the computation history. Computations proceed by applying a sequence of local operations to data. We call a computation *nonoblivious* if at each step, which local operation to use and which operands to apply it to may depend on the computation history. A generic deterministic Turing machine computation is nonoblivious. We can view each state as defining an operation on a fixed number of tape cells, where the operands are given by the tape head locations. In each step, the outcome of the applied operation affects the next state and tape head locations, so both the operation and the operands can depend on the computation history. In contrast, a classical circuit computation is oblivious because neither the operation (gate) nor the operands (wires connected to the gate inputs) depend on the computation history (values carried on the wires).

In the randomized and quantum settings, the notion of a computation history becomes more complicated because there can be many computation paths. In the randomized setting, applying a randomized operation to a configuration may split it into a distribution over configurations, and the randomized Turing machine model allows the next state and tape head locations to depend on which computation path was taken. In the quantum setting, applying a quantum operation to a basis state may split it into a superposition over several basis states, and general nonoblivious behavior would allow the next operation and operands to depend on which computation path was taken. However, it is unclear whether such behavior is physically realizable, as currently envisioned technologies all select quantum operations classically. An intermediate notion of nonobliviousness, where the operations and operands may depend on previous measurement outcomes but not on the quantum computation path, does seem physically realistic (see for example [42, section 2.5.2] and the references within).

Classical control. There is a wide spectrum of degrees of interaction between a quantum computation and its classical control. On the one hand, one can imagine a quantum computation that is entirely “self-sufficient”, other than the interaction needed to provide the input and observe the output. On the other hand, one can imagine a quantum computation that is guided classically every step of the way. Self-sufficiency is inherent to computations that are nonoblivious to the quantum computation path, whereas measurements are inherently classically controlled operations. Incorporating intermediate measurements into computations that are nonoblivious to the quantum computation path would require some sort of global coordination among the quantum computation paths to determine when a measurement should take place.

Syntax. Our model should be syntactic, meaning that identifying valid programs in the model is decidable. If we are interested in bounded-error computations, then we cannot hope to decidably distinguish programs satisfying the bounded-error promise from those that do not. However, we should be able to distinguish programs that are correctly formatted (according to the postulates of quantum mechanics) from those that are not. Allowing nonobliviousness to the quantum computation path complicates this syntax check. If different components of the superposition can undergo different unitary operations then the overall operation is not automatically unitary, due to interference. Extra conditions on the transition function are needed to guarantee unitarity.

Complexity of the transition amplitudes. Care should be taken in specifying the allowable transition amplitudes. In the randomized setting, it is possible to solve undecidable languages by encoding the characteristic sequences of these languages in the transition probabilities. This problem is usually handled by using a certain universal set of elementary randomized operations, e.g., an unbiased coin flip. In the quantum setting, the same problem arises with unrestricted amplitudes. Again, one can solve the problem by restricting the elementary quantum operations to a universal set. However, unlike in the randomized setting, there is no single standard universal set like the unbiased coin flip with which all quantum algorithms are easy to describe. Algorithm designers should be allowed to use arbitrary local operations provided they do not smuggle hard-to-compute information into the amplitudes.

Absolute halting. In order to measure time complexity, we should use a model that naturally allows any algorithm to halt absolutely within some time bound t. In the randomized setting, one can design algorithms whose running times are random variables and may actually run forever. We can handle such algorithms by clocking them, so that they are forced to halt within some fixed number of time steps. Our quantum model should provide a similar mechanism.

2.2 Earlier Models

Now that we have spelled out the relevant issues and criteria, we consider several previously studied models as candidates.

2.2.1 Models with Quantum Control

Bernstein and Vazirani [8] laid the foundations for studying quantum complexity theory using quantum Turing machines. Their model uses a single tape and therefore cannot handle sublinear space bounds. Like classical one-tape Turing machines, their model is sequential-access. It does not allow intermediate measurements. On the other hand, their model is fully nonoblivious: the transition function produces a superposition over basis configurations, and the state and tape head location may be different for different components of the superposition. Their model represents the self-sufficient extreme of the classical control spectrum. In their paper, Bernstein and Vazirani prove that their model is syntactic by giving a few orthogonality constraints on the entries of the transition function table that are necessary and sufficient for the overall evolution to be unitary. These conditions are somewhat unintuitive, and can be traced back to the possibility of nonobliviousness to the quantum computation path. Bernstein and Vazirani restrict the transition amplitudes by requiring that the first k bits of each amplitude are computable deterministically in time $\mathrm{poly}(k)$. Their model is nontrivial to clock; they require that the transition function be designed in such a way that the machine always halts, meaning that it reaches a superposition in which all non-halting basis configurations have zero amplitude. Bernstein and Vazirani detail how to design such mechanisms.

In [38] Watrous considers a model similar to Bernstein and Vazirani's, but with one read-write work tape and a read-only input tape not counting toward the space bound. The model naturally allows for sublinear space bounds, but it is still sequential-access. It allows intermediate measurements but only for the halting mechanism: a special register is measured after each time step, with the outcome indicating "halt and output 1", "halt and output 0", or "continue". The model is nonoblivious like the Bernstein-Vazirani model. It has more classical interaction due to the halting mechanism, but this is arguably not "classical control". The syntax conditions on the transition function are similar to those for the Bernstein-Vazirani model. The results in [38] require the transition amplitudes to be rational, which is somewhat unappealing since one may often wish to use Hadamard gates, which have irrational amplitudes. Similar to the Bernstein-Vazirani model, the model is nontrivial to clock. In fact, the results in [38] rely on counting an infinite computation as a rejection.

The paper [43] describes a model similar to the one from [38] but without any restriction on the transition amplitudes. Another model is also described in [43] where the tape head movements are classical but the finite control is still quantum.

The main issue with the above models for our purposes is their sequential-access nature. It is possible to handle this problem by imposing a random-access mechanism. However, the conditions on the entries of the transition function table characterizing unitary evolution become more complicated and unintuitive, making the model inconvenient to work with. Again, the culprit is the nonobliviousness to the quantum computation path. Since this behavior does not appear to be physically realizable in the foreseeable future anyway, the complications arising from it are in some sense unjustified.

2.2.2 Models with Classical Control

In [39] Watrous considers a different model of space-bounded quantum computation. This model is essentially a classical Turing machine with an additional quantum work tape and a fixed-size quantum register. Sublinear space bounds are handled by charging for the space of the classical work tape and the quantum work tape but not the input tape. All three tape heads move sequentially. This model handles intermediate measurements. It is oblivious to the quantum computation path; the state and tape head locations cannot be in superposition with the contents of the quantum work tape. However, the computation is nonoblivious to the classical computation history, including the measurement outcomes. The finite control is classical; in each step it selects a quantum operation and applies it to the combination of the qubit under the quantum work tape head together with the fixed-size register. The register is needed because there is only one head on the quantum work tape, but a quantum operation needs to act on multiple qubits to create entanglement. The allowed operations come from the so-called quantum operations formalism (see [31, chapter 8]), which encompasses unitary operations and general measurements, as well as interaction with an external environment. Each quantum operation produces an output from a finite alphabet — the measurement outcome in the case of a measurement. This outcome influences the next (classical) transition. This model is syntactic just like classical Turing machines, with the additional step of testing that each quantum operation satisfies the definition of a valid quantum operation. For his constructions, Watrous needs the transition amplitudes to be algebraic. This model is trivial to clock, since all the control is done classically and thus the machine can halt in a fixed number of steps, just as in the classical setting.

Perdrix and Jorrand [32] study a model they dub a “classically-controlled quantum Turing machine”. Their model is like a classical multitape Turing machine, but where the cells are quantum; in each step, classical machinery dictates the quantum operation that is applied to the cells under the tape heads. The model handles neither sublinear-space algorithms nor random access to memory. The authors allow intermediate measurements, and they consider two variations: one where the local operations come from the quantum operations formalism, and one where only projective measurements are allowed. The model is oblivious to the quantum computation path but nonoblivious to the intermediate measurement outcomes. Like Watrous’s model from [39], the control is classical, so syntax and absolute halting are not a problem. The issue of the complexity of the transition amplitudes is not addressed in [32].

These two models are convenient to work with since the essence of the quantum aspects of a computation are isolated into local operations that are chosen classically and applied to a simple quantum register. This reflects the currently envisioned realizations of quantum computers (see for example [28] and the references within for superconductor-based technologies, [10] for trapped ion technologies, [33] for quantum optics technologies, and the references in the survey [29] for quantum dot technologies; see also the classic article [16]). Watrous's model from [39] is the most suitable as a starting point for the exposition of our results, but we need to make some modifications to it in order to address the following issues.

We want our model to have random access to emphasize the fact that our time-space lower bound does not exploit any model artifacts due to sequential access. We can make the model from [39] random-access by allowing each of the tape heads to jump in unit time to a location whose address we have classically computed, just as can be done for deterministic and randomized Turing machines.

The quantum operations used in the model from [39] are more general than we need to consider. The quantum operations formalism models the evolution of open systems, which is of information-theoretic rather than computational concern [31]. We choose to restrict the set of allowed operations to unitary operations and projective measurements in the computational basis. This is without loss of generality since an operation from the quantum operations formalism can be simulated by introducing an additional "environment" system with a constant number of qubits, performing a unitary operation, and then measuring the environment qubits in the computational basis [31]. Using nonobliviousness to measurement outcomes, we can then reset the environment qubits for use in simulating the next quantum operation.

Algorithms like Grover's require quantum access to the input, i.e., an operation that allows different basis states in a superposition to access different bits of the input simultaneously. On inputs of length n, this is done with a query gate that effects the transformation $|i\rangle|b\rangle \mapsto |i\rangle|b \oplus x_i\rangle$ where $i \in \{0,1\}^{\lceil \log_2 n \rceil}$, $b \in \{0,1\}$, and x_i is the ith bit of the input. The model from [39] does not have such an operation and thus cannot express algorithms like Grover's. While this operation seems no more physically realistic than nonobliviousness to the quantum computation path if we view the input as stored in a classical memory, it does make sense when the input is actually the output of another computation. For these reasons, we include such an operation in our model.

Finally, our restriction on the transition amplitudes is similar to the one assumed by Bernstein and Vazirani, and is more general than algebraic numbers.

2.3 Our Model

For concreteness, we now describe and motivate the particular model we use for the exposition of our arguments. Our model addresses all the issues listed in Section 2.1 and is an adaptation of Watrous's model from [39], as described at the end of Section 2.2.

In a nutshell, our model of a quantum algorithm running in time t and space s consists of a classically controlled machine that applies t operations to s bits and qubits, and which supports random access to the input and memory and can be influenced by intermediate measurement outcomes. We more formally define our model in Section 2.3.1 and then discuss the complexity measures for our model in Section 2.3.2.

2.3.1 Model Definition

We define a quantum algorithm as follows. There are three semi-infinite tapes: the input tape, the classical work tape, and the quantum work tape. Each cell on the input tape holds one bit or a blank symbol. Each cell on the classical work tape holds one bit. Each cell on the quantum work tape holds one qubit. The input tape contains the input, a string in $\{0,1\}^n$, followed by blanks, and the classical and quantum work tapes are initialized to all 0's. There are a fixed number of tape heads, each of which is restricted to one of the three tapes. There may be multiple heads moving independently on the same tape.

The finite control, the head movements on all tapes, and the operations on the classical work tape are all classical; each operation on the quantum work tape can be either a unitary operator or a single-qubit projective measurement in the computational basis. In each step of the computation, the finite control of the algorithm is in one of a finite number of states. Each state has an associated classical function, which is applied jointly to the contents of the cells under the heads on the classical work tape, and an associated quantum operation, which is applied jointly to the contents of the cells under the heads on the quantum work tape. The next state of the finite control and the head movements are determined by the current state, the contents of the cells under the input tape heads and classical work tape heads at the beginning of the computation step, and the measurement outcome if the quantum operation was a measurement.

Each head moves left one cell, moves right one cell, stays where it is, or jumps to a new location at a precomputed address. The latter type of move is classical random access. We also allow "quantum random access" to the input by optionally performing a query that effects the transformation $|i\rangle|b\rangle \mapsto |i\rangle|b \oplus x_i\rangle$ on a contiguous block of qubits on the quantum work tape, where $i \in \{0,1\}^*$ is an index into the input, $b \in \{0,1\}$, and x_i is the ith bit of the input of length n or 0 if $i > n$. To specify addresses for classical random access, as well as the two endpoint addresses of the block for quantum random access, the algorithm can write addresses on special one-way sequential-access write-only index tapes, which get erased after each time they are used.

Among the states of the finite control are an "accept" state and a "reject" state, which cause the algorithm to halt. Although not needed in this paper, the algorithm can be augmented with a one-way sequential-access write-only classical output tape in order to compute non-Boolean functions.

Let us motivate our model definition. In terms of physical computing systems, the input tape corresponds to an external input source, the classical work tape corresponds to classical memory, and the quantum work tape corresponds to quantum memory. The bits and qubits under the heads correspond to the data being operated on in the CPU.

We use multiple heads on each tape since many algorithms are naturally expressed using elementary operations that involve more than one bit or qubit. Creating entanglement requires multiple-qubit operations and hence multiple quantum work tape heads. The index tapes are needed for random access since addresses have non-constant length and thus cannot fit under the tape heads all at once. A minor issue arises with our multiple head approach: an operation on a work tape may not be well-defined if two of the heads are over the same cell. Rather than requiring programs to avoid this situation, which would make the model non-syntactic, we can just assume that no operation is performed on the violating work tape when this situation arises.

2.3.2 Complexity Measures

We say that a quantum algorithm M runs in time $t(n)$ if for all input lengths n, all inputs x of length n, and all computation paths of M on input x, M halts in at most $t(n)$ steps. We say that a quantum algorithm M runs in space $s(n)$ if for all input lengths n, all inputs x of length n, and all computation paths of M on input x, the largest address of a classical work tape head or quantum work tape head during the computation of M on input x is at most $s(n)$. Either the time or the space may be infinite. Note that we consider *all* computation paths, even ones that occur with probability 0 due to destructive interference.

The above definition of space usage allows the space to be exponential in the running time, since in time t an algorithm can write an address that is exponential in t and move a head to that location using the random-access mechanism. However, the space usage can be reduced to at most the running time with at most a polylogarithmic factor increase in the latter by compressing the data and using an appropriate data structure to store (old address, new address) pairs. (See Section 2.3.1 of [37] for a similar construction.)

We say that a quantum algorithm M solves a language L with error $\epsilon(n)$ if M has finite running time and for all input lengths n and all inputs x of length n, $\Pr\big(M(x) \neq L(x)\big) \leq \epsilon(n)$. We say the error is bounded if $\epsilon(n) \leq 1/3$ for all n and unbounded if $\epsilon(n) < 1/2$ for all n.

For a unitary operator U, we let $\mathcal{A}(U)$ denote the set of absolute values of both the real and imaginary parts of each matrix entry of U in the computational basis. For a set S of unitary operators, we let $\mathcal{A}(S) = \bigcup_{U \in S} \mathcal{A}(U)$. Each quantum algorithm M has a *library* of q-qubit unitary operators it can use (other than measurement gates and quantum query gates, which every quantum algorithm can use), where q is the number of quantum work tape heads; we define $\mathcal{A}(M)$ to be $\mathcal{A}(S)$ where S is the set of library gates of M. We use the following definitions to limit the complexity of the numbers in $\mathcal{A}(M)$. (The first definition is used for both Theorem 1 and Theorem 2, while the second definition is only used for Theorem 1.)

Definition 1. *We say a deterministic algorithm A is a (t, s)-*approximator *for $r \in [0, 1]$ if given a positive integer precision parameter p, A runs in time $t(p)$ and space $s(p)$ and outputs a nonnegative integer $\widehat{r}$ in binary such that $\left|r - \widehat{r}/2^p\right| \leq 1/2^p$.*

Definition 2. *We say a nondeterministic algorithm G is a (t, s)-*generator *for $r \in [0, 1]$ if given a positive integer precision parameter p, G runs in time $t(p)$ and space $s(p)$ and has $\widehat{r}$ accepting computation paths such that $\left|r - \widehat{r}/2^p\right| \leq 1/2^p$.*

For a complex matrix A, we let $\|A\|$ denote the operator norm $\|A\| = \sup_{v \neq 0} \|Av\|/\|v\|$ where $\|\cdot\|$ on the right side denotes the 2-norm for vectors. Throughout this paper, we freely use the fact that for all $d \times d$ complex matrices A and B, if $\|A - B\| \leq \epsilon$ then each of the $2d^2$ real numbers comprising A is within ϵ of the corresponding real number in B, and conversely, if each of the $2d^2$ real numbers comprising A is within ϵ of the corresponding real number in B, then $\|A - B\| \leq \sqrt{2}d\epsilon$. We weaken the latter bound to $2d\epsilon$ for notational simplicity throughout this paper.

As evidence in support of our model of choice, we note that the following results hold in our model.

- Every language solvable by a bounded-error randomized algorithm M running in time t and space s is also solvable by a bounded-error quantum algorithm running in time $O(t)$ and space $s+1$ that directly simulates M and produces unbiased coin flips by applying a Hadamard gate to one qubit and then measuring it. This qubit can be reused to generate as many random bits as needed.

- Grover's algorithm [21] shows that OR (the problem of computing the disjunction of the n input bits) is solvable by a bounded-error quantum algorithm running in time $O(n^{1/2} \cdot \operatorname{polylog} n)$ and space $O(\log n)$.

- Shor's algorithm [35] shows that a nontrivial factor of an integer of bit length n can be computed in time $O(n^3 \cdot \operatorname{polylog} n)$ and space $O(n)$ with error probability at most $1/3$.

3 Randomized Simulation

In this section we prove Theorem 1. In Section 3.1 we state our simulation result in full generality and give several instantiations. We discuss the relationship of our result and its proof to previous simulations in Section 3.2. In Section 3.3 we prove our general simulation result, and we conclude in Section 3.4 with some remarks on the proof.

3.1 General Result and Instantiations

We now state our general randomized simulation result.

Theorem 4. *Suppose language L is solvable by a quantum algorithm M running in time $t \geq \log n$ and space $s \geq \log n$ with error $\epsilon < 1/2$ having q quantum work tape heads, and such that each number in $\mathcal{A}(M)$ has a (t', s')-approximator. Then L is also solvable by an unbounded-error randomized algorithm running in time $O\big(t \cdot p + t'(p)\big)$ and space $O\big(s + \log t + p + s'(p)\big)$ for any integer function $p \geq \log_2 \frac{5t \cdot 2^{q+1}}{1/2-\epsilon}$, provided t, s, and p are constructible by a deterministic algorithm with the latter time and space bounds.*

The function p in Theorem 4 represents the precision parameter on which the simulation invokes the approximators for the numbers in $\mathcal{A}(M)$. The parameters t, s, ϵ, and p are all functions of the input length n, and the big O's are with respect to $n \to \infty$. Regarding the conditions in Theorem 4 (and Theorem 1), we assume t and s are at least logarithmic so that they dominate any logarithmic terms arising from indexed access to the input. The constructibility constraints are satisfied by all "ordinary" functions, which is sufficient for obtaining our lower bound, Theorem 3. Theorem 4 is a corollary of the following even more general simulation.

Theorem 5 (Main Randomized Simulation). *Suppose language L is solvable by a quantum algorithm M running in time $t \geq \log n$ and space $s \geq \log n$ with error $\epsilon < 1/2$ having q quantum work tape heads, and such that each number in $\mathcal{A}(M)$ has a (t', s')-generator. Then L is also solvable by an unbounded-error randomized algorithm running in time $O\big(t \cdot t'(p)\big)$ and space $O\big(s + \log t + s'(p)\big)$ for any integer function $p \geq \log_2 \frac{5t \cdot 2^{q+1}}{1/2-\epsilon}$, provided t, s, p, and $t'(p)$ are constructible by a deterministic algorithm with the latter time and space bounds. Moreover, the randomized algorithm uses the (t', s')-generators only as black boxes.*

The proof of Theorem 5 is given in Section 3.3. We now show how Theorem 5 implies Theorem 4 and Theorem 1.

Proof of Theorem 4. The randomized algorithm for L first runs the approximator for each number $r \in \mathcal{A}(M)$ and stores the result $\widehat{r}$, then runs the simulation from Theorem 5 using $\big(O(p), O(p)\big)$-generators that nondeterministically guess a $(p+1)$-bit nonnegative integer and accept iff it is less than $\widehat{r}$. Note that the numbers in $\mathcal{A}(M)$ might not have actual $\big(O(p), O(p)\big)$-generators according to Definition 2, but we can simulate oracle access to such generators using the precomputed values $\widehat{r}$. Since the simulation from Theorem 5 only invokes the generators as black boxes, we can simulate each call to a generator in time and space $O(p)$, and so the simulation from Theorem 5 takes time

$O(t \cdot p)$ and space $O\big(s + \log t + p\big)$. The additional $O(t'(p))$ time overhead and $O(s'(p))$ space overhead comes from running the approximators to get the numbers $\widehat{r}$ once, at the beginning of the computation. □

Proof of Theorem 1. If M has algebraic transition amplitudes, then each number in $\mathcal{A}(M)$ has a $\big(\mathrm{poly}(p), O(p)\big)$-approximator (by Newton's method, with a little care taken to ensure the linear space bound). Theorem 1 follows by setting $\epsilon = 1/3$ and $p = \big\lceil \log_2(30t \cdot 2^{q+1})\big\rceil$ and applying Theorem 4. □

Theorem 1 deals with the standard setting of bounded error. To handle unbounded error, applying Theorem 4 or Theorem 5 requires an upper bound on the precision p and hence a bound on how close the error ϵ can be to $1/2$. In the case of rational transition amplitudes, a simple bound yields the following corollary to Theorem 5.

Corollary 2. *Every language solvable by an unbounded-error quantum algorithm running in time $t \geq \log n$ and space $s \geq \log n$ with rational transition amplitudes is also solvable by an unbounded-error randomized algorithm running in time $O(t^2)$ and space $O(s + \log t)$, provided t and s are constructible by a deterministic algorithm with the latter time and space bounds.*

Proof. The acceptance probability of any quantum algorithm running in time t on a fixed input equals a multivariate polynomial of total degree at most $2t$ with integer coefficients and whose variables are evaluated at the numbers in $\mathcal{A}(M)$.[2] Thus in the case of rational transition amplitudes, the acceptance probability can be expressed as a rational number where the denominator is the product of the denominators of the numbers in $\mathcal{A}(M)$ raised to the power $2t$. This implies that if $\epsilon < 1/2$, then in fact $\epsilon \leq 1/2 - 2^{-O(t)}$. Using the fact that each rational number in $[0, 1]$ has a trivial $\big(O(p), O(\log p)\big)$-generator, Theorem 5 yields the result. □

It is natural to conjecture that whenever a quantum algorithm (with arbitrary transition amplitudes) solves a language in time t with error $\epsilon < 1/2$, then in fact $\epsilon \leq 1/2 - 2^{-O(t)}$. However, Arnold has shown this to be false [5]. In fact, he shows that ϵ cannot be bounded away from $1/2$ by *any* function of t; moreover, this holds even when the transition amplitudes are approximable and the function of t is sufficiently constructible. Partial positive results are known for the case of algebraic amplitudes [39].

3.2 Intuition and Relationship to Previous Work

There are three previously known simulations of quantum algorithms by unbounded-error randomized algorithms: a time-efficient one [1, 19] and two space-efficient ones [38, 39]. Modulo some minor technicalities, our simulation subsumes and improves all three for bounded-error quantum algorithms, and it subsumes and improves the simulations of [1, 19] and [38] for unbounded-error quantum algorithms. We now give a detailed comparison of our results and our proof techniques with these previous simulations.

The inclusion BQP ⊆ PP was first proved by Adleman et al. [1], and the proof was later rephrased using counting complexity tools by Fortnow and Rogers [19]. Under the assumption that the elementary quantum operations have been discretized in some unitary way, this simulation

[2] This can be seen, for example, using the postponed measurement framework discussed in Section 3.3.2.

incurs a constant factor overhead in running time. However, the fastest general method for unitary discretization is the Solovay-Kitaev algorithm [25, 11], which incurs a $\operatorname{polylog} t$ factor overhead in running time for bounded-error quantum computations. Also, the simulation of [1, 19] does not preserve the space bound when intermediate measurements are allowed. Theorem 4 subsumes and improves the result of [1, 19] by improving both the time and space overheads. Strictly speaking, the simulation of [1, 19] actually assumes a quantum model that is nonoblivious to the quantum computation path (see Section 2) and is thus not immediately captured by the statement of Theorem 4; however, with some technical work our proof should extend to such models (see the remark in Section 3.4).

The proof of Theorem 5 uses the technique in [19] as a starting point. The basic idea of the latter simulation is to write the final amplitude of a basis state as a simple linear combination of #P functions, where each #P function counts the number of quantum computation paths leading to that state with a certain path amplitude. Assuming the elementary quantum operations come from an appropriate discrete universal set, simple algebraic manipulations can be used to express the probability of acceptance as the difference between two #P functions, up to a simple common scaling factor. This leads to a time- and space-efficient simulation by an unbounded-error randomized algorithm, assuming there is only a final measurement and no intermediate measurements.

Intermediate measurements can be eliminated by instead copying the measurement results into ancilla qubits. This could blow up the space bound to t, which could be exponential in s. We handle intermediate measurements directly by first adapting the approach from the previous paragraph to capture the probability of observing any particular sequence of measurement outcomes. The acceptance probability can then be expressed as a sum over all sequences of measurement outcomes that lead to acceptance, where each term is the scaled difference of two #P functions. We can combine those terms into a single one using the closure of #P under uniform exponential sums. However, the usual way of doing this — nondeterministically guess and store a sequence and then run the computation corresponding to that sequence — is too space-inefficient. To address this problem, we note that the crux of the construction corresponds to multiplying two #P functions on related inputs. The standard approach runs the two computations in sequence, accepting iff both accept. We argue that we can run these two computations *in parallel* and keep them in sync so that they access each bit of the guessed sequence at the same time, allowing us to reference each bit only once. We can then guess each bit when needed during the final simulation and overwrite it with the next guess bit, allowing us to meet the space constraint.

Since the standard Solovay-Kitaev algorithm for unitary discretization (adapted to work with finite precision) takes time and space $\operatorname{polylog} t$ for bounded-error quantum computations, this approach leads to a simulation running in time $O(t \cdot \operatorname{polylog} t)$ and space $O(s + \operatorname{polylog} t)$. The space bound could be reduced to $O(s+\log t)$ using our space-efficient simulation with a universal set (Theorem 2), but to also bring the running time down to $O(t \cdot \log t)$ we eschew the Solovay-Kitaev algorithm altogether and directly use *nonunitary* discretizations, which can be obtained by simply approximating each matrix entry with $O(\log t)$ bits. Then using the technique from the previous paragraph, we can *approximate* the probability of each sequence of measurement outcomes using a scaled difference of #P functions. A bit of care is needed to ensure that the total error in the probability estimate, over these exponentially many classical computation paths, is small enough.

Watrous [38] shows how to simulate unbounded-error quantum algorithms running in space s with rational amplitudes by unbounded-error randomized algorithms running in space $O(s)$. His technique is genuinely different than ours; he first gives an $O(s)$-space reduction to the problem of comparing the determinants of two $2^{O(s)} \times 2^{O(s)}$ integer matrices to see which is larger, and then uses a logarithmic space unbounded-error randomized algorithm for this problem due to Allender and Ogihara [3].

In [38], Watrous allows nonobliviousness to the quantum computation path, but as with [1, 19], this is not a genuine obstacle. Another technicality is that he allows $t \not\leq 2^{O(s)}$ while still achieving an $O(s)$ space simulation. In fact, he allows space-bounded quantum algorithms to run forever, counting an infinite computation as a rejection, which does not correspond to a realistic computational process. However, in our general model, if a quantum algorithm runs in time $t \not\leq 2^{O(s)}$ then for infinitely many input lengths, it cannot even keep track of its own running time, and for $t = \operatorname{poly} n$ it cannot even write down an address into the input and so it must exploit sequential access in order to do something nontrivial. In the normal case when $t \leq 2^{O(s)}$, the result from [38] is subsumed and improved by Corollary 2 since the latter has a running time of $O(t^2)$ whereas the result from [38] has no bound on the running time of the simulation.

In another paper, Watrous [39] shows again how to simulate unbounded-error quantum algorithms running in space s by unbounded-error randomized algorithms running in space $O(s)$, assuming a different model of quantum computation which allows algebraic transition amplitudes. He first shows how to approximate arbitrary algebraic numbers using ratios of GapL functions, then he space-efficiently reduces the quantum computation to the problem of computing the sign of a particular entry in a certain infinite series of matrices, and finally he gives a logarithmic space unbounded-error randomized algorithm for the latter problem. As in [38] he allows $t \not\leq 2^{O(s)}$, whereas all known natural algorithms satisfy $t \leq 2^{O(s)}$. He uses the quantum operations formalism with algebraic amplitudes, but this can be simulated in our model (see the discussion in Section 2.2.2). In the normal case when $t \leq 2^{O(s)}$, the result from [39], restricted to bounded-error quantum algorithms, is subsumed and improved by Theorem 1 since the latter has a running time of $O(t \cdot \log t)$ whereas the result from [39] has no bound on the running time of the simulation. It remains an open problem to match the space bound of [39] for the simulation of unbounded-error quantum computations using our technique.

3.3 Proof of Theorem 5

Our simulation relies on a description of the computation in terms of mixed states and uses classical algorithms to guess paths down a computation tree. An equivalent alternative formulation involves describing the computation in terms of density operators and using classical algorithms to guess paths through products of matrices. We elaborate on the latter in the remarks in Section 3.4.

Suppose language L is solvable by a quantum algorithm M running in time $t \geq \log n$ and space $s \geq \log n$ with error $\epsilon < 1/2$ having q quantum work tape heads, and such that each number in $\mathcal{A}(M)$ has a (t', s')-generator. Let $p \geq \log_2 \frac{5t \cdot 2^{q+1}}{1/2-\epsilon}$ be an integer function. Basically, this bound on the precision parameter p comes from the need to approximate M's acceptance probability within $1/2 - \epsilon$, which necessitates approximating the unitary operator applied in each computation step within $\frac{1/2-\epsilon}{5t}$, which necessitates approximating the numbers in $\mathcal{A}(M)$ within $\frac{1/2-\epsilon}{5t \cdot 2^{q+1}}$.

We fix an arbitrary input x, and for simplicity of notation we let $p = p(|x|)$ and assume that on input x, M uses exactly s qubits and always applies exactly t quantum gates, exactly m of which are measurements, regardless of the observed sequence of measurement outcomes. This can be achieved by padding the computation with gates that do not affect the accept/reject decision of M. Note that whether a particular run of M on input x accepts is uniquely determined by the observed sequence of measurement outcomes $\mu \in \{0,1\}^m$. This reflects our model's obliviousness to the quantum computation path but nonobliviousness to the intermediate measurement outcomes.

In Section 3.3.1 we describe a certain tree representing the computation of M on input x, state a key structural property of this tree, and give the final simulation. Then in Section 3.3.2 we prove the key structural property.

3.3.1 Algorithm Construction

PP can be characterized as the class of languages consisting of inputs for which the difference of two #P functions exceeds a certain polynomial-time computable threshold. Thus, we would like to approximate the acceptance probability of M on input x as the ratio of the difference of two #P functions over some polynomial-time computable scaling factor. To facilitate the argument, we model the computation of M on input x as a tree, analogous to the usual computation trees one associates with randomized or nondeterministic computations. We would then like to approximate the final amplitude of a basis state with a linear combination of #P functions, where each #P function counts the number of root-to-leaf paths that lead to that basis state and have a particular path amplitude. (The coefficients in this linear combination are the path amplitudes, which are the products of the transition amplitudes along the paths.) To accomplish this, we discretize the elementary quantum operations using rational numbers, which are obtained via the (t', s')-generators. These rational numbers have a common denominator that can be factored out and absorbed into the scaling factor, leaving Gaussian integers (i.e., complex numbers with integer real and imaginary parts), which can be expressed using #P functions.

We formally define the computation tree for our fixed input x as follows. It has depth t. Each level $\tau = 0, \ldots, t$ represents the state of the quantum tape after the τth gate is applied and before the $(\tau+1)$st gate is applied. Each node v has four labels.

- $\tau(v) \in \{0, \ldots, t\}$, representing the level of v
- $\mu(v) \in \{0,1\}^{\leq m}$, representing the sequence of measurement outcomes that leads to v
- $\sigma(v) \in \{0,1\}^s$, representing the computational basis state at v
- $\alpha(v) \in \{1, i, -1, -i\}$, representing the numerator of the amplitude of v

Our construction of the tree will guarantee that the path amplitude associated with v is a fourth root of unity divided by $2^{\tau(v)p}$, which justifies the restriction on $\alpha(v)$ in the fourth bullet. Labels of nodes across a given level need not be unique; if $\tau(v) = \tau(u)$ and $\mu(v) = \mu(u)$ and $\sigma(v) = \sigma(u)$, then v and u represent interference.

We now define the tree inductively as follows. The root node v, representing the initial state, has $\tau(v) = 0$, $\mu(v) = \lambda$, $\sigma(v) = 0^s$, and $\alpha(v) = 1$. Now consider an arbitrary node v. If $\tau(v) = t$ then v is a leaf. Otherwise, v has children that depend on the type and operands of the $(\tau(v)+1)$st gate applied given that $\mu(v)$ is observed (which will always be well-defined). There are three cases, corresponding to the three types of gates M can use (library gates, measurement gates, and query gates).

- Suppose the gate is a unitary operator U from M's library, applied to qubits $j_1, \ldots, j_q \in \{1, \ldots, s\}$. Let $\big(a_{0^q} + b_{0^q} i, \ \ldots, \ a_{1^q} + b_{1^q} i\big)^T$ denote the $\sigma(v)_{j_1,\ldots,j_q}$ column of U in the computational basis. Then v has $2 \cdot 2^q$ groups of children, corresponding to each of the real numbers that make up this column. The children corresponding to a_{0^q} are as follows (the other groups are similar). There are $\widehat{r}$ identical children, where $\widehat{r}$ is the number of accepting computation paths generated by the hypothesized (t', s')-generator for $|a_{0^q}|$ on precision parameter p. Each of these children u satisfies $\tau(u) = \tau(v) + 1$, $\mu(u) = \mu(v)$, $\sigma(u)$ equals $\sigma(v)$ with bits $j_1, \ldots, j_q$ set to 0, and $\alpha(u) = \alpha(v) \cdot \mathrm{sgn}(a_{0^q})$. For the children corresponding to b_{0^q}, we would set $\alpha(u) = \alpha(v) \cdot i \cdot \mathrm{sgn}(b_{0^q})$.

- Suppose the gate is a single-qubit projective measurement of the jth qubit in the computational basis. Then v has 2^p identical children u, each with $\tau(u) = \tau(v) + 1$, $\mu(u) = \mu(v)\sigma(v)_j$, $\sigma(u) = \sigma(v)$, and $\alpha(u) = \alpha(v)$.
- Suppose the gate is a quantum query gate, and suppose $\sigma(v)$ and the operands of the query gate are such that x_i is to be XORed into the jth qubit. Then v has 2^p identical children u, each with $\tau(u) = \tau(v) + 1$, $\mu(u) = \mu(v)$, $\sigma(u)$ equals $\sigma(v)$ with $\sigma(v)_j$ replaced by $\sigma(v)_j \oplus x_i$, and $\alpha(u) = \alpha(v)$.

The reason the latter two cases use 2^p children is so that every internal node has an implicit denominator of 2^p in the transition amplitude, which allows us to factor out this common denominator across each level.

In order to describe how the computation tree reflects the evolution of the quantum tape, we introduce the following notation.

- $V_{\tau,\mu,\sigma,\alpha} = \{v \;:\; \tau(v) = \tau,\ \mu(v) = \mu,\ \sigma(v) = \sigma,\ \alpha(v) = \alpha\}$
- $V_{\tau,\mu,\sigma} = \bigcup_\alpha V_{\tau,\mu,\sigma,\alpha}$
- $V_{\tau,\mu} = \bigcup_\sigma V_{\tau,\mu,\sigma}$
- $V_\tau = \bigcup_\mu V_{\tau,\mu}$

Suppose we run M but do not renormalize state vectors after measurements. Then after τ gates have been applied, we have a vector for each sequence of measurement outcomes μ that could have occurred during the first τ steps. The nodes in $V_{\tau,\mu}$ together with their amplitudes $\alpha(v)/2^{\tau p}$ approximately give the vector for μ, since these are exactly the nodes whose computation paths are consistent with the measurement outcomes $\mu_1 \cdots \mu_{|\mu|}$. In other words, the vector for μ is approximately $\sum_{v \in V_{\tau,\mu}} \big(\alpha(v)/2^{\tau p}\big)\big|\sigma(v)\big\rangle$ and thus the squared 2-norm of the latter vector is approximately the probability of observing μ. This suggests that at the end of the computation, when $\tau = t$, the sum of these squared 2-norms over all μ causing M to accept should approximately equal the probability M accepts. Lemma 1 below confirms this. Care is needed in the formal proof, however, for two reasons. First, the approximations used to generate the children of a node when a library gate is applied are not generally consistent with any unitary evolution. Second, there can be exponentially many sequences μ that cause M to accept, and just summing simple error estimates over all these sequences does not work since under our target efficiency parameters, we cannot guarantee exponentially good approximations in the probabilities of observing every individual μ. We handle this by instead arguing that in a certain sense, these probabilities are approximated well "on average", using the fact that the approximation errors are relative to the probability weights of the paths. This is good enough for our purpose.

Lemma 1.

$$\left|\Pr(M\ \textit{accepts}) - \frac{1}{2^{2tp}} \sum_{\substack{\mu\in\{0,1\}^m\\ \textit{causing } M\\ \textit{to accept}}} \sum_{\sigma\in\{0,1\}^s} \sum_{\alpha\in\{1,i,-1,-i\}} \Big(\big|V_{t,\mu,\sigma,\alpha}\big|^2 - \big|V_{t,\mu,\sigma,\alpha}\big|\cdot\big|V_{t,\mu,\sigma,-\alpha}\big|\Big)\right| \;<\; 1/2-\epsilon$$

We prove Lemma 1 in Section 3.3.2 below. With Lemma 1 in hand, we now show how to construct a randomized algorithm N running in time $O\big(t\cdot t'(p)\big)$ and space $O\big(s+\log t+s'(p)\big)$ such that

- if $\Pr(M \text{ accepts}) \geq 1-\epsilon$ then $\Pr(N \text{ accepts}) > 1/2$, and
- if $\Pr(M \text{ accepts}) \leq \epsilon$ then $\Pr(N \text{ accepts}) < 1/2$.

This suffices to prove Theorem 5.

We first construct a nondeterministic algorithm N' taking as input a tuple (x,μ,σ,α) where $\mu\in\{0,1\}^m$, $\sigma\in\{0,1\}^s$, and $\alpha\in\{1,i,-1,-i\}$, whose number of accepting computation paths satisfies $\#N' = |V_{t,\mu,\sigma,\alpha}|$. This is straightforward: have N' nondeterministically guess a root-to-leaf path in the computation tree. The only information about the current node v it needs to keep track of is $\sigma(v)$ and $\alpha(v)$, taking space $O(s)$. It keeps a pointer into μ, taking space $O(\log t)$. It determines the correct sequence of gates by simulating the classical part of M, taking $O(t)$ time and $O(s)$ space. When processing a library gate, N' nondeterministically guesses one of the $2\cdot 2^q$ groups of children, then runs the appropriate (t',s')-generator on input p to nondeterministically pick one of the children in that group. When processing a measurement gate, N' checks that applying the measurement to the current $\sigma(v)$ yields the next bit of μ. It rejects if not and otherwise continues by using that bit of μ as the measurement outcome and generating 2^p nondeterministic branches. Processing a quantum query gate is trivial. When it reaches a leaf v, N' checks that $\sigma(v)=\sigma$ and $\alpha(v)=\alpha$ (it already knows that $\mu(v)=\mu$ having made it this far) and accepts if so and rejects otherwise. As constructed, N' has the desired behavior, and it runs in time $O\big(t\cdot t'(p)\big)$ and space $O\big(s+\log t+s'(p)\big)$ (with respect to $n=|x|$).

We next construct nondeterministic algorithms N^+ and N^- such that on input x,

$$\#N^+ \;=\; \sum_{\substack{\mu\in\{0,1\}^m\\ \text{causing } M\\ \text{to accept}}} \sum_{\sigma\in\{0,1\}^s} \sum_{\alpha\in\{1,i,-1,-i\}} \big|V_{t,\mu,\sigma,\alpha}\big|^2$$

and

$$\#N^- \;=\; \sum_{\substack{\mu\in\{0,1\}^m\\ \text{causing } M\\ \text{to accept}}} \sum_{\sigma\in\{0,1\}^s} \sum_{\alpha\in\{1,i,-1,-i\}} \big|V_{t,\mu,\sigma,\alpha}\big|\cdot\big|V_{t,\mu,\sigma,-\alpha}\big|.$$

Since they are similar, we just describe N^+.

By the closure of #P functions under multiplication and under uniform exponential sums, we can generate the desired number of accepting computation paths by nondeterministically guessing $\mu\in\{0,1\}^m$, $\sigma\in\{0,1\}^s$, and $\alpha\in\{1,i,-1,-i\}$, then running two independent copies of N' on input (x,μ,σ,α) and accepting iff both accept and μ causes M to accept. (Since every accepting execution of N' follows an execution of M with measurement outcomes μ, we know at the end whether μ causes M to accept.) However, this standard method runs in space $O\big(t + s'(p)\big)$ due to the need to store μ of length m, which could be as large as t. (Storing σ and α is not problematic.) We bring the space usage of N^+ down to $O\big(s + \log t + s'(p)\big)$ as follows. We run the two copies of N' *in parallel*, keeping them in sync so that they access each bit of μ at the same time. (Note that since N^+ can reject after seeing a single disagreement with μ, the two copies being run will apply the same sequence of gates and thus access each bit of μ at the same time.) It follows that N^+ only needs to reference each bit of μ once. This implies that rather than guessing and storing μ at the beginning, N^+ can guess each bit of μ only when needed by the two copies and overwrite the previous bit of μ. As constructed, N^+ runs in time $O\big(t \cdot t'(p)\big)$ and space $O\big(s + \log t + s'(p)\big)$ and generates the desired number of accepting computation paths.

By Lemma 1,

$$\left| \Pr(M \text{ accepts}) - \frac{1}{2^{2tp}}\big(\#N^+ - \#N^-\big) \right| \; < \; 1/2 - \epsilon.$$

Thus,

- if $\Pr(M \text{ accepts}) \geq 1 - \epsilon$ then $\#N^+ - \#N^- > 2^{2tp-1}$, and
- if $\Pr(M \text{ accepts}) \leq \epsilon$ then $\#N^+ - \#N^- < 2^{2tp-1}$.

We can now use a standard technique to obtain the final randomized simulation N. Since t and $t'(p)$ are constructible, we can assume without loss of generality that N^+ and N^- are constructed so as to have exactly 2^g computation paths for some constructible function $g \leq O\big(t \cdot t'(p)\big)$. This allows us to compare numbers of accepting paths to numbers of rejecting paths. By nondeterministically picking N^+ or N^- to run, and flipping the answer if N^- was chosen, we get $\big(\#N^+ - \#N^-\big) + 2^g$ accepting computation paths. We can generate an additional 2^{g+1} dummy computation paths, exactly $2^g + 2^{2tp-1}$ of which reject, to shift the critical number of accepting paths to exactly half the total number of paths. This can be done without dominating the time or space efficiency, by the constructibility condition.

As constructed, N runs in time $O\big(t \cdot t'(p)\big)$ and space $O\big(s + \log t + s'(p)\big)$ and accepts x with probability greater than 1/2 if $x \in L$ and with probability less than 1/2 if $x \notin L$. This finishes the proof of Theorem 5.

3.3.2 Postponing Measurements

In this section we prove Lemma 1. We start by describing a framework for analyzing quantum algorithms with intermediate measurements by implicitly postponing the measurements and tracking the unitary evolution of the resulting purification. We stress that we are doing so for reasons of analysis only; our actual simulations do not involve postponing measurements.

Recall that we are assuming for simplicity of notation that on our fixed input x, M uses exactly s qubits and always applies exactly t quantum gates, exactly m of which are measurements, regardless of the observed sequence of measurement outcomes. We conceptually postpone the measurements in the computation by

(1) introducing m ancilla qubits initialized to all 0's,

(2) replacing the ith measurement on each classical computation path by an operation that entangles the ith ancilla qubit with the qubit being measured (by applying a CNOT to the ancilla qubit with the measured qubit as the control), and

(3) measuring the m ancilla qubits at the end.

In the τth step of the simulation, we apply a unitary operator U_τ on a system of $s+m$ qubits, where U_τ acts independently on each of the subspaces corresponding to distinct sequences of measurement outcomes that can be observed before time step τ. More precisely, consider the set of $\mu \in \{0,1\}^{\leq m}$ such that given that μ is observed, the τth gate is applied after μ is observed but not after the $(|\mu|+1)$st measurement gate is applied. Let $\mathcal{U}_\tau$ be the set of these μ such that the τth gate is unitary, and let $\mathcal{M}_\tau$ be the set of these μ such that the τth gate is a measurement. For $\nu \in \{0,1\}^m$, let P_ν denote the projection on the state space of the ancilla qubits to the one-dimensional subspace spanned by $|\nu\rangle$.

For $\mu \in \mathcal{U}_\tau$, let $G_{\tau,\mu}$ denote the unitary operator on the state space of s qubits induced by the τth gate applied given that μ is observed. Then U_τ acts as $G_{\tau,\mu} \otimes I$ on the range of $I \otimes P_{\mu 0^{m-|\mu|}}$. For each $\mu \in \mathcal{M}_\tau$, U_τ applies an entangling operator $E_{\tau,\mu}$ that acts only on the range of $I \otimes \big(P_{\mu 0^{m-|\mu|}} + P_{\mu 1 0^{m-1-|\mu|}}\big)$. We arbitrarily set the behavior of U_τ on the remaining subspaces to the identity operator. Thus,

$$U_\tau \;=\; \left(\sum_{\mu \in \mathcal{U}_\tau} G_{\tau,\mu} \otimes P_{\mu 0^{m-|\mu|}}\right) + \left(\sum_{\mu \in \mathcal{M}_\tau} E_{\tau,\mu}\right) + R,$$

where R is a term that expresses the behavior on the remaining subspaces.

It is well-known, and can be verified from first principles, that the probability of observing any sequence of measurement outcomes $\mu \in \{0,1\}^m$ when M is run equals the probability of observing μ after the evolution $U = U_t U_{t-1} \cdots U_2 U_1$ with all of the ancilla qubits initialized to 0. That is, $\Pr(\mu \text{ observed}) = \big\|(I \otimes P_\mu) U |0^{s+m}\rangle\big\|^2$. It follows that $\Pr(M \text{ accepts}) = \big\|PU|0^{s+m}\rangle\big\|^2$, where P denotes the sum of $I \otimes P_\mu$ over all μ causing M to accept.

Now for each $\mu \in \mathcal{U}_\tau$, if $G_{\tau,\mu}$ comes from a library gate then let $\widehat{G}_{\tau,\mu}$ be an operator analogous to $G_{\tau,\mu}$ but where each real and imaginary part of the matrix in the computational basis is replaced by a number with the same sign whose absolute value is $\widehat{r}/2^p$, where $\widehat{r}$ is the value from the appropriate (t', s')-generator on precision parameter p. If $G_{\tau,\mu}$ comes from a quantum query gate then let $\widehat{G}_{\tau,\mu} = G_{\tau,\mu}$. Naturally, we define

$$\widehat{U}_\tau \;=\; \left(\sum_{\mu \in \mathcal{U}_\tau} \widehat{G}_{\tau,\mu} \otimes P_{\mu 0^{m-|\mu|}}\right) + \left(\sum_{\mu \in \mathcal{M}_\tau} E_{\tau,\mu}\right) + R$$

and $\widehat{U} = \widehat{U}_t \cdots \widehat{U}_1$.

Lemma 1 now follows from the following two claims.[3]

Claim 1. $\Big|\big\|P\widehat{U}|0^{s+m}\rangle\big\|^2 - \big\|PU|0^{s+m}\rangle\big\|^2\Big| \;<\; 1/2 - \epsilon$

[3] A result very similar to Claim 1 is needed in Section 4.1 for deriving Theorem 6 from Theorem 7. Specifically, if each $\widehat{G}_{\tau,\mu}$ is unitary and approximates $G_{\tau,\mu}$ within ϵ' then in the statement of Claim 1 we can replace $< 1/2 - \epsilon$ by $\leq 2t\epsilon'$, and we have that $\big\|P\widehat{U}|0^{s+m}\rangle\big\|^2$ equals the probability of acceptance of the modified quantum computation. Further, if the approximations are only up to global phase factors, then we can change the definition of $\widehat{U}$ to counteract the global phase, and then $\big\|P\widehat{U}|0^{s+m}\rangle\big\|^2$ still equals the probability of acceptance.

Claim 2.

$$\left\|P\widehat{U}|0^{s+m}\rangle\right\|^2 \;=\; \frac{1}{2^{2tp}} \sum_{\substack{\mu\in\{0,1\}^m\\ \textit{causing } M\\ \textit{to accept}}} \; \sum_{\sigma\in\{0,1\}^s} \; \sum_{\alpha\in\{1,i,-1,-i\}} \Big(\left|V_{t,\mu,\sigma,\alpha}\right|^2 - \left|V_{t,\mu,\sigma,\alpha}\right|\cdot\left|V_{t,\mu,\sigma,-\alpha}\right|\Big)$$

Proof of Claim 1. Suppose $V_{\tau,\mu}$ is the q-qubit unitary operator corresponding to $G_{\tau,\mu}$ (assuming the latter represents the application of a library gate), and let $\widehat{V}_{\tau,\mu}$ be similar but where the real and imaginary parts are replaced by their approximations. Since each of these approximations is within $1/2^p$ of the correct value, we have $\left\|\widehat{V}_{\tau,\mu}-V_{\tau,\mu}\right\| \leq 2^{q+1}/2^p \leq (1/2-\epsilon)/5t$. Tensoring with the identity does not change the operator norm of an operator, so we also have $\left\|\widehat{G}_{\tau,\mu} - G_{\tau,\mu}\right\| \leq (1/2-\epsilon)/5t$. We claim that this implies that $\left\|\widehat{U}_\tau - U_\tau\right\| \leq (1/2-\epsilon)/5t$ for all τ; this holds since for every unit vector $|\psi\rangle$ in the state space of $s+m$ qubits, we have

$$\begin{aligned}
\left\|\left(\widehat{U}_\tau - U_\tau\right)|\psi\rangle\right\|^2 &= \left\|\sum_{\mu\in\mathcal{U}_\tau}\Big(\big(\widehat{G}_{\tau,\mu}\otimes P_{\mu 0^{m-|\mu|}}\big) - \big(G_{\tau,\mu}\otimes P_{\mu 0^{m-|\mu|}}\big)\Big)|\psi\rangle\right\|^2 \\
&= \sum_{\mu\in\mathcal{U}_\tau}\left\|\Big(\big(\widehat{G}_{\tau,\mu} - G_{\tau,\mu}\big)\otimes I\Big)\big(I\otimes P_{\mu 0^{m-|\mu|}}\big)|\psi\rangle\right\|^2 \\
&\leq \sum_{\mu\in\mathcal{U}_\tau}\Big(\frac{1/2-\epsilon}{5t}\Big)^2\left\|\big(I\otimes P_{\mu 0^{m-|\mu|}}\big)|\psi\rangle\right\|^2 \\
&\leq \Big(\frac{1/2-\epsilon}{5t}\Big)^2.
\end{aligned}$$

Now we claim that for all $\tau = 0,\ldots,t$,

$$\left\|\widehat{U}_\tau\cdots\widehat{U}_1 - U_\tau\cdots U_1\right\| \;\leq\; \left(1+\frac{1/2-\epsilon}{5t}\right)^\tau - 1.$$

We prove this by induction on τ. The base case $\tau = 0$ is trivial since $\widehat{U}_0\cdots\widehat{U}_1 = U_0\cdots U_1 = I$. For the induction step, we assume the claim holds for $\tau-1$ and prove it for τ. By the triangle inequality, it suffices to show the following two inequalities.

$$\left\|\widehat{U}_\tau\big(\widehat{U}_{\tau-1}\cdots\widehat{U}_1\big) - U_\tau\big(\widehat{U}_{\tau-1}\cdots\widehat{U}_1\big)\right\| \;\leq\; \frac{1/2-\epsilon}{5t}\cdot\left(1+\frac{1/2-\epsilon}{5t}\right)^{\tau-1} \tag{1}$$

$$\left\|U_\tau\big(\widehat{U}_{\tau-1}\cdots\widehat{U}_1\big) - U_\tau\big(U_{\tau-1}\cdots U_1\big)\right\| \;\leq\; \left(1+\frac{1/2-\epsilon}{5t}\right)^{\tau-1} - 1 \tag{2}$$

Inequality (1) follows from the facts that

$$\left\|\widehat{U}_\tau - U_\tau\right\| \;\leq\; \frac{1/2-\epsilon}{5t}$$

and

$$\left\|\widehat{U}_{\tau-1}\cdots\widehat{U}_1\right\| \;\leq\; \left(1+\frac{1/2-\epsilon}{5t}\right)^{\tau-1}$$

(the latter following from the induction hypothesis and the fact that $\|U_{\tau-1}\cdots U_1\| = 1$). Inequality (2) follows from the induction hypothesis and the fact that $\|U_\tau\| = 1$. This finishes the induction. It follows that

$$\big\|\widehat{U} - U\big\| \;\le\; \left(1 + \frac{1/2-\epsilon}{5t}\right)^t - 1 \;\le\; e^{(1/2-\epsilon)/5} - 1 \;\le\; \left(1 + 2\cdot\frac{1/2-\epsilon}{5}\right) - 1 \;=\; 0.4\big(1/2-\epsilon\big).$$

Finally, defining $|\Delta\rangle = \big(\widehat{U} - U\big)|0^{s+m}\rangle$, we have

$$\begin{aligned}\Big|\big\|P\widehat{U}|0^{s+m}\rangle\big\|^2 - \big\|PU|0^{s+m}\rangle\big\|^2\Big| \;&=\; \Big|\langle 0^{s+m}|\widehat{U}^\dagger P|\Delta\rangle + \langle\Delta|PU|0^{s+m}\rangle\Big| \\ &\le\; \big\|\widehat{U}^\dagger\big\|\cdot\big\||\Delta\rangle\big\| + \big\||\Delta\rangle\big\| \\ &\le \Big(1 + 0.4\big(1/2-\epsilon\big)\Big)\cdot 0.4\big(1/2-\epsilon\big) + 0.4\big(1/2-\epsilon\big) \\ &<\; 1/2-\epsilon,\end{aligned}$$

where the third line follows by the facts that $\big\|\widehat{U} - U\big\| \le 0.4(1/2-\epsilon)$ and $\|U\| = 1$. This completes the proof of Claim 1. □

Proof of Claim 2. For each node v, define

$$\big|\psi(v)\big\rangle \;=\; \frac{\alpha(v)}{2^{\tau(v)p}}\big|\sigma(v)\mu(v)0^{m-|\mu(v)|}\big\rangle.$$

Note that $\big|\psi(v)\big\rangle$ is the basis state of v multiplied by its amplitude, with the ancilla qubits set to indicate the sequence of measurement outcomes that leads to v. It follows from the definition of the tree that for each internal node v,

$$\widehat{U}_{\tau(v)+1}\big|\psi(v)\big\rangle \;=\; \sum_{\text{children } u \text{ of } v}\big|\psi(u)\big\rangle.$$

Thus since $\big|\psi(v)\big\rangle = |0^{s+m}\rangle$ when v is the root, by induction on τ we have

$$\widehat{U}_\tau\cdots\widehat{U}_1|0^{s+m}\rangle \;=\; \sum_{v\in V_\tau}\big|\psi(v)\big\rangle$$

for all $\tau = 0,\ldots,t$. It follows that

$$\begin{aligned}\big\|P\widehat{U}|0^{s+m}\rangle\big\|^2 \;&=\; \left\|P\sum_{v\in V_t}\big|\psi(v)\big\rangle\right\|^2 \\ &=\; \left\|\sum_{\substack{\mu\in\{0,1\}^m \\ \text{causing } M \\ \text{to accept}}}\;\sum_{v\in V_{t,\mu}}\big|\psi(v)\big\rangle\right\|^2 \\ &=\; \sum_{\substack{\mu\in\{0,1\}^m \\ \text{causing } M \\ \text{to accept}}}\;\sum_{\sigma\in\{0,1\}^s}\left\|\sum_{v\in V_{t,\mu,\sigma}}\big|\psi(v)\big\rangle\right\|^2\end{aligned}$$

$$= \sum_{\substack{\mu\in\{0,1\}^m \\ \text{causing } M \\ \text{to accept}}} \sum_{\sigma\in\{0,1\}^s} \left| \sum_{v\in V_{t,\mu,\sigma}} \frac{\alpha(v)}{2^{tp}} \right|^2$$

$$= \frac{1}{2^{2tp}} \sum_{\substack{\mu\in\{0,1\}^m \\ \text{causing } M \\ \text{to accept}}} \sum_{\sigma\in\{0,1\}^s} \sum_{\alpha\in\{1,i,-1,-i\}} \left(\left|V_{t,\mu,\sigma,\alpha}\right|^2 - \left|V_{t,\mu,\sigma,\alpha}\right| \cdot \left|V_{t,\mu,\sigma,-\alpha}\right| \right).$$

This completes the proof of Claim 2 and the proof of Lemma 1. □

3.4 Remarks

We make two remarks concerning our main randomized simulation, Theorem 5.

Handling nonobliviousness to the quantum computation path. We believe that nothing prevents our proof of Theorem 5 from carrying over to any reasonable model of quantum computation that is nonoblivious to the quantum computation path. In this case, the sequence of gates leading to a node v in the computation tree does not depend only on $\mu(v)$, but this does not present a significant problem for our proof. However, the proof becomes more technical due to the (limited) interactions between the local operations applied on different quantum computation paths. These complications arise for the same reason as the unintuitive conditions on the transition function in the models from [8] and [38]. We feel that working out the details of such a result would not be well-motivated since the currently envisioned realizations of quantum computers do not support such nonoblivious behavior.

Alternate proof using the density operator formalism. The proof of Theorem 5 can be rephrased in the language of density operators. We now briefly sketch how to do this. After each time step, the mixed state of the quantum tape can be expressed as an operator on s qubits, called the density operator, which captures all the observable information. The density operator of the initial state is $|0^s\rangle\langle 0^s|$. Suppose the first operation applied is a unitary gate, and let U denote the corresponding unitary operator on the entire state space. Then the density operator after step 1 is $U|0^s\rangle\langle 0^s|U^\dagger$. Suppose the next operation applied is a measurement, and let M_0 and M_1 be the corresponding projection operators. Then the density operator after step 2 is $\sum_{\mu\in\{0,1\}} M_\mu U|0^s\rangle\langle 0^s|U^\dagger M_\mu^\dagger$. Now the next operation can depend on the measurement outcome μ; suppose it is unitary U_0 if $\mu = 0$ or unitary U_1 if $\mu = 1$. Since $M_0 U|0^s\rangle\langle 0^s|U^\dagger M_0^\dagger$ represents the unnormalized state given $\mu = 0$, the subsequent unnormalized state is represented by $U_0 M_0 U|0^s\rangle\langle 0^s|U^\dagger M_0^\dagger U_0^\dagger$, and similarly for $\mu = 1$. Thus the density operator after step 3 is $\sum_{\mu\in\{0,1\}} U_\mu M_\mu U|0^s\rangle\langle 0^s|U^\dagger M_\mu^\dagger U_\mu^\dagger$. Continuing in this way, we find that the mixed state at the end is

$$\sum_{\mu\in\{0,1\}^m} L_{t,\mu} \cdots L_{1,\mu}|0^s\rangle\langle 0^s|L_{1,\mu}^\dagger \cdots L_{t,\mu}^\dagger$$

where $L_{\tau,\mu}$ are some linear operators. The probability of acceptance is the trace of

$$\sum_{\substack{\mu\in\{0,1\}^m \\ \text{causing } M \\ \text{to accept}}} L_{t,\mu} \cdots L_{1,\mu}|0^s\rangle\langle 0^s|L_{1,\mu}^\dagger \cdots L_{t,\mu}^\dagger.$$

If we approximate each real and imaginary part of each $L_{\tau,\mu}$ as a rational number with denominator 2^p (using the (t', s')-generators) then the trace of the resulting sum is approximately the probability of acceptance. (This can be proved by translating to the postponed measurement framework and using the argument from Claim 1. We do not know of a clean way to directly phrase this argument in terms of density operators.) Factoring out $1/2^{2tp}$ yields a sum of products of Gaussian integer matrices, and we just need to express the trace of this sum using a difference of #P functions. This involves guessing $\sigma \in \{0,1\}^s$ (summing over the diagonal entries), guessing a μ that leads to acceptance, and guessing a path through the matrix product to generate the (σ, σ) entry of the product (and guessing whether to take the real or imaginary part of each entry). One of the two #P functions sums the positive terms in the expression, while the other sums the negative terms. The key for preserving the space bound is to guess the path starting in the middle and *simultaneously* guessing two paths outward. That way, each bit of μ only needs to be accessed once and can be guessed at that time and later overwritten. The unbounded-error randomized simulation then follows as before by combining the two #P functions and generating dummy computation paths to shift the critical fraction of paths to exactly 1/2.

4 Quantum Simulation

In this section we prove Theorem 2. In Section 4.1 we state our simulation result in full generality and discuss its relationship to previous work. In Section 4.2 we describe the intuition and new ideas behind the main component of the proof (Theorem 7 below), and then in Section 4.3 we give the full proof of Theorem 7.

4.1 Overview

We start with our precise definition of a universal set.[4]

Definition 3. *A finite set S of unitary quantum gates is* universal *if there exists a q_0 such that for all $q \geq q_0$ the following holds. For every q-qubit unitary operator U and every $\epsilon > 0$ there exist q-qubit unitary operators $U_1, \ldots, U_k$, each of which is a gate from S applied to some of the q qubits, and there exists a global phase factor $e^{i\theta}$, such that $\left\|U - e^{i\theta} U_k \cdots U_1\right\| \leq \epsilon$.*

There is a long line of research on constructing universal sets [12, 15, 6, 27, 13, 7, 25, 9, 34].[5] Examples of universal sets include the Toffoli gate together with the Hadamard gate [34], and the CNOT gate together with any single-qubit unitary gate whose square does not preserve the computational basis [34].

Our quantum simulation result basically states that every quantum algorithm can be simulated with only a small overhead in time and space by another quantum algorithm whose library is an arbitrary universal set S, provided S is closed under adjoint. Recall that in our terminology, the library gates of a quantum algorithm all act on q qubits, where q is the number of work tape heads. Since S may contain gates acting on different numbers of qubits, when we say the library is S we allow tensoring with the identity so that all gates act on the same number of qubits (we also allow

[4] Some notions of universality allow ancilla qubits; however, as far as we know, all such results have been subsumed by results that do not need ancilla qubits.

[5] For these results, the global phase factor $e^{i\theta}$ does not need to depend on ϵ, but this does not matter for our purposes.

rearranging the qubits, so a gate from S can be applied to any subset of the quantum work tape head locations). We now state the general form of our simulation result.

Theorem 6. *For every universal set S with parameter q_0 such that S is closed under adjoint and each number in $\mathcal{A}(S)$ has a (t', s')-approximator, the following holds. Suppose language L is solvable by a quantum algorithm M running in time t and space s with error $\epsilon < 1/2$ having $q \geq q_0$ quantum work tape heads, and such that each number in $\mathcal{A}(M)$ has a (t', s')-approximator. Then L is also solvable by a quantum algorithm with library S running in time $O\big(t \cdot \mathrm{polylog}(1/\epsilon') + t'(p)\big)$ and space $O\big(s + \log(1/\epsilon') + s'(p)\big)$ with error $\epsilon + 2t\epsilon'$, where ϵ' is any function constructible by a deterministic algorithm with the latter time and space bounds, and p is a certain function in $\Theta_q\big(\log(1/\epsilon')\big)$.*

The function p in Theorem 6 represents the precision parameter on which the simulation invokes the approximators for the numbers in $\mathcal{A}(M) \cup \mathcal{A}(S)$. The parameters t, s, ϵ, ϵ', and p are all functions of the input length n, and the big O's are with respect to $n \to \infty$. The bulk of the proof of Theorem 6 is the following result. Let $U(d)$ denote the set of unitary operators on $\mathbb{C}^d$.

Theorem 7 (Space-Efficient Version of the Solovay-Kitaev Theorem). *For each constant integer $d \geq 2$ the following holds. Suppose $S \subseteq U(d)$ is a finite set closed under adjoint such that for every $U \in U(d)$ and every $\epsilon > 0$ there exists a sequence $U_1, \ldots, U_k \in S$ and a global phase factor $e^{i\theta}$ such that $\big\|U - e^{i\theta} U_k \cdots U_1\big\| \leq \epsilon$. Then for every $U \in U(d)$ and every $\epsilon > 0$ there exists a sequence $U_1, \ldots, U_k \in S$ with $k \leq \mathrm{polylog}(1/\epsilon)$ and a global phase factor $e^{i\theta}$ such that $\big\|U - e^{i\theta} U_k \cdots U_1\big\| \leq \epsilon$. Moreover, such a sequence can be computed by a deterministic algorithm running in time $\mathrm{polylog}(1/\epsilon)$ and space $O(\log(1/\epsilon))$, given as input ϵ and matrices that are at distance at most $f(\epsilon)$ from U and the gates in S, where f is a certain polynomial depending only on d.*

Note that the algorithm in Theorem 7 runs in space $O(\log(1/\epsilon))$ while outputting a list of $\mathrm{polylog}(1/\epsilon)$ gates. This means that it outputs the labels of the gates on the fly, in the order they are to be applied, and the output list does not count toward the space bound. Also, the algorithm needs some hardcoded information about S, and the constant factors in the efficiency parameters depend on S. Finally, the values of k and $e^{i\theta}$ in the conclusion of the theorem are not generally the same as in the hypothesis for the same U and ϵ, and the global phase factor $e^{i\theta}$ in the conclusion may depend on ϵ even if the global phase factors in the hypothesis do not.

In Section 4.2 we explain the intuition and new ideas behind Theorem 7, and then in Section 4.3 we give the formal proof. We do not attempt to optimize the degree of the polylog in the running time; this can likely be done with a complicated analysis and rearrangement of the ingredients in the proof. However, we do mention a few simple optimizations in Section 4.3.4.

Proof of Theorem 6. Assume without loss of generality that every operator in S acts on q qubits. First, compute ϵ' and $p = \big\lceil \log_2 \frac{2^{q+1}}{f(\epsilon')} \big\rceil$ where f is the polynomial from Theorem 7 for $d = 2^q$. Then run the (t', s')-approximators on precision parameter p to obtain matrices that approximate the gates in S and in M's library within $f(\epsilon')$. Then start simulating M, and every time it applies a library gate U, run the algorithm from Theorem 7 with ϵ' as the ϵ-parameter to find a sequence of gates from S whose product ϵ'-approximates U up to a global phase factor, and apply those gates instead. Note that the algorithm from Theorem 7 needs to be rerun at every step since we do not have enough space to store the approximating sequences for M's library gates. The time and space complexities of the new algorithm are immediate. We just need to verify that the probability any input is accepted changes by at most $2t\epsilon'$; we omit the argument as a nearly identical one appears in Section 3.3.2. (The differences are that in that proof, nonunitary approximations are allowed and global phase factors do not appear in the approximations.) □

Proof of Theorem 2. Let $\epsilon = 1/3$ and let $\epsilon' \approx 1/20t$ be a constructible function. Apply Theorem 6, using the fact that each number in $\mathcal{A}(M) \cup \mathcal{A}(S)$ has a $\big(\text{poly}(p), O(p)\big)$-approximator (as noted in the proof of Theorem 1). Then use amplification to bring the error down to 1/3. □

Theorem 7 is a strengthening of the well-known Solovay-Kitaev Theorem [25]. The latter theorem states that there exists an appropriate sequence of gates as in Theorem 7.[6] Moreover, the proof gives a deterministic algorithm for computing such a sequence in time and space $\text{polylog}(1/\epsilon)$, in a highly idealized model of computation in which exact numerical calculations (including matrix diagonalization, which is impossible using just $+, -, \times, \div$ and kth roots) can be performed at unit cost each (see [11], Appendix 3 of [31], or Section 8.3 of [26]). However, we cannot do exact calculations, since the entries of our matrices may require infinitely many bits to specify and we are charged for the space to store numbers and the time to compute with them. The complexity of the algorithm in a standard finite-precision model has not been studied in the literature, other than some remarks in [26, page 76]. A careful analysis shows that approximating the matrix entries of U and the gates in S with precision parameter $p = \text{polylog}(1/\epsilon)$ yields sufficiently good approximations to all matrices involved, leading to an algorithm running in time and space $\text{polylog}(1/\epsilon)$. The difficulty in Theorem 7 is in getting the space complexity down to $O(\log(1/\epsilon))$, which also necessitates getting the precision parameter down to $O(\log(1/\epsilon))$, while maintaining a $\text{polylog}(1/\epsilon)$ running time.

Other results on quantum compiling are known. While in the standard proof of the Solovay-Kitaev Theorem [11, 31] the length of the approximating sequence of gates is $O(\log^{3.97}(1/\epsilon))$, Section 8.3 of [26] presents a slightly more technical proof that gets the sequence length down to $O(\log^{3+\delta}(1/\epsilon))$ for any constant $\delta > 0$. Section 13.7 of [26] describes a different approach that gets the sequence length down to $O\big(\log^2(1/\epsilon)\log\log(1/\epsilon)\big)$ but does not work for every universal set. Harrow, Recht, and Chuang [22] present yet a different approach that gets the sequence length down to $O(\log(1/\epsilon))$ (which is optimal up to constant factors) but does not work for every universal set and is not even constructive.

4.2 Intuition

We now give the intuition behind the proof of Theorem 7, focusing on the new ideas. In Section 4.2.1 we give a quick overview of the standard Solovay-Kitaev algorithm [11]. In the subsequent sections we motivate and develop our improvement by discussing three primary issues.

4.2.1 Overview of the Standard Algorithm

The standard version of the algorithm takes as input a unitary operator U and an integer $\ell \geq 0$ and returns a sequence of gates from S whose product $\widetilde{U}$ ϵ_ℓ-approximates[7] U, where $\epsilon_0 > 0$ is a certain small constant and $\epsilon_\ell \leq O(\epsilon_{\ell-1}^{1.5}) \ll \epsilon_{\ell-1}$ for $\ell > 0$. We suppress the dependence of $\widetilde{U}$ on

[6] There is actually a minor but slightly nontrivial argument needed to obtain the Solovay-Kitaev Theorem for arbitrary universal sets in $U(d)$, which we could not find mentioned in the literature but seems to have been implicitly assumed in the literature. This point is discussed in Section 4.3.

[7] We ignore the issue of global phase factors throughout Section 4.2; this issue is treated carefully in Section 4.3.

ℓ in order to declutter the notation later on. The algorithm is recursive in ℓ. For the base case $\ell = 0$, by our assumption on S we can use brute force to find an ϵ_0-approximation using a constant number of gates from S. The induction consists of a bootstrapping argument: given the ability to make recursive calls that find $\epsilon_{\ell-1}$-approximations using gates from S, we would like to find ϵ_ℓ-approximations using gates from S. The key that makes this possible is the following remarkable fact, whose proof is inspired by Lie theory.

Fact 1 (Key Fact, Informal Version). *If U is $\epsilon_{\ell-1}$-close to the identity I, then it is possible to find unitary operators V and W such that for any unitary operators $\widetilde{V}$ and $\widetilde{W}$ that $\epsilon_{\ell-1}$-approximate V and W respectively, the group commutator $\widetilde{V}\widetilde{W}\widetilde{V}^\dagger\widetilde{W}^\dagger$ ϵ_ℓ-approximates U.*

This can be turned into a recursive algorithm as follows. First we must "translate" U to the neighborhood of I in order to apply the key fact. To do this, we make a recursive call on U to obtain a sequence of gates from S whose product $\epsilon_{\ell-1}$-approximates U. We define $\Upsilon = U$ and let $\widetilde{\Upsilon}$ denote this product. The notation U and $\widetilde{U}$ is always with respect to our arbitrary level ℓ, while Υ and $\widetilde{\Upsilon}$ are with respect to level $\ell - 1$. The purpose of this nonstandard notation is to simplify the notation later on. Now $U\widetilde{\Upsilon}^\dagger$ is $\epsilon_{\ell-1}$-close to I, so we can apply the key fact to it. What good is this? Note that since $U = U\widetilde{\Upsilon}^\dagger\widetilde{\Upsilon}$, we have that for any unitary operator that ϵ_ℓ-approximates $U\widetilde{\Upsilon}^\dagger$, multiplying it on the right by $\widetilde{\Upsilon}$ yields a unitary operator that ϵ_ℓ-approximates U. Thus, if we can find a sequence of gates whose product ϵ_ℓ-approximates $U\widetilde{\Upsilon}^\dagger$, then we can append it to the sequence we have for $\widetilde{\Upsilon}$ to get a sequence whose product ϵ_ℓ-approximates U.[8] The key fact helps us achieve the former. Specifically, we compute V and W from $U\widetilde{\Upsilon}^\dagger$, and then we recursively find a sequence of gates from S whose product $\widetilde{V}$ is $\epsilon_{\ell-1}$-close to V, and we recursively find a similar sequence for W. The key fact tells us that $\widetilde{V}\widetilde{W}\widetilde{V}^\dagger\widetilde{W}^\dagger$ is ϵ_ℓ-close to $U\widetilde{\Upsilon}^\dagger$. Thus we can just string together four sequences of gates from S whose products equal $\widetilde{V}$, $\widetilde{W}$, $\widetilde{V}^\dagger$, and $\widetilde{W}^\dagger$ to get a sequence whose product ϵ_ℓ-approximates $U\widetilde{\Upsilon}^\dagger$. We already have sequences for $\widetilde{V}$ and $\widetilde{W}$, by the recursive calls. To obtain a sequence whose product equals $\widetilde{V}^\dagger$, just take the sequence for $\widetilde{V}$, reverse the order of the gates, and invert each gate. (This is where we need the assumption that S is closed under adjoint.) A sequence for $\widetilde{W}^\dagger$ is obtained similarly. Declaring $\widetilde{U} = \widetilde{V}\widetilde{W}\widetilde{V}^\dagger\widetilde{W}^\dagger\widetilde{\Upsilon}$, we have that $\widetilde{U}$ is ϵ_ℓ-close to U, and we have found a sequence whose product equals $\widetilde{U}$.

If we seek an ϵ-approximation to U, we can just pick L to be the smallest value such that $\epsilon_L \leq \epsilon$ and run the algorithm on U and $\ell = L$. In particular, $L = \Theta(\log\log(1/\epsilon))$ levels of recursion suffice. Since the base case always produces a sequence whose length is at most a constant, say m, and at each level the maximum length of the approximating sequence gets multiplied by 5, a sequence produced at the ℓth level has length at most $5^\ell m$. Using $L = \Theta(\log\log(1/\epsilon))$ levels, the sequence has length at most polylog$(1/\epsilon)$. We model the execution of the algorithm as a recursion tree where the leaves are at level 0, the root is at level L, and each internal node (representing a call for some U) has three children representing the calls for Υ, V, and W.

The algorithm sketched above runs in time polylog$(1/\epsilon)$, in a highly idealized model of computation in which exact numerical calculations can be performed at unit cost each. However, we need to work in the standard model of computation, in which we are charged for the space to store numbers and the time to compute with them. By some technical analysis and tweaking of numerical methods, it is possible to make the above algorithm work in the standard model, running in time polylog$(1/\epsilon)$. Our improvement (Theorem 7) is an algorithm that runs in space $O(\log(1/\epsilon))$ while

[8] Note that the sequence for $\widetilde{\Upsilon}$ comes first in the sequence order, since $\widetilde{\Upsilon}$ is the right multiplicand of $(U\widetilde{\Upsilon}^\dagger)\widetilde{\Upsilon}$.

still running in time polylog$(1/\epsilon)$. We now discuss the obstacles to obtaining such a space-efficient algorithm and how we overcome them.

4.2.2 The Numerical Precision Problem

Since the operators we deal with may require infinitely many bits to specify, we need to work with finite-precision approximations to them. That is, if the "intended" operator is U, we instead take as input a (not necessarily unitary) matrix $\widehat{U}$ guaranteed to be close to U and for which we have an exact binary representation.[9] At level ℓ, when we are seeking an ϵ_ℓ-approximation to U, we assume that $\widehat{U}$ is within some δ_ℓ of U. Thus, given $\widehat{U}$, our goal is to output a sequence of gates from S whose product $\widetilde{U}$ is ϵ_ℓ-close to U. Since we only know $\widehat{U}$ and not U, we must ensure $\widetilde{U}$ is ϵ_ℓ-close to every unitary operator U that is within δ_ℓ of $\widehat{U}$. Note that this requires $\delta_\ell \le \epsilon_\ell$ since otherwise there might not exist an operator that simultaneously ϵ_ℓ-approximates every U that is within δ_ℓ of $\widehat{U}$. However, δ_ℓ may need to be much smaller than ϵ_ℓ. Let us consider how small δ_ℓ needs to be. For the base case $\ell = 0$, constant δ_0-approximations suffice. Suppose that $\delta_\ell \ge \Omega(\delta_{\ell-1}^{\alpha})$ suffices for the reduction from level ℓ to level $\ell - 1$, where α is some constant. Since $\epsilon_\ell \le O(\epsilon_{\ell-1}^{1.5})$, the above observation shows that we must have $\alpha \ge 1.5$. Since the root of the recursion tree is at level L, the output is a sequence of gates whose product is only guaranteed to approximate U within $\epsilon_L = \big(\Theta(\epsilon_0)\big)^{1.5^L}$, while we need an approximation to the intended input U within $\delta_L = \big(\Theta(\delta_0)\big)^{\alpha^L}$. Hence, $\log(1/\epsilon) \le \log(1/\epsilon_L) \le O(1.5^L)$, while just writing down a sufficiently good approximation to the intended input takes $\Omega(\log(1/\delta_L)) \ge \Omega(\alpha^L) \ge \Omega\big(\log^{\log_{1.5}\alpha}(1/\epsilon_L)\big) \ge \Omega\big(\log^{\log_{1.5}\alpha}(1/\epsilon)\big)$ bits. As we mention below, it turns out that we cannot avoid writing down and storing such approximations. Thus, to have any hope of getting a logarithmic space algorithm, we must achieve $\alpha = 1.5$. We now explain how to do this and why it is not trivial.

Consider an arbitrary node at level $\ell > 0$ in the recursion tree, and pretend for simplicity that the intended input U is already $\epsilon_{\ell-1}$-close to I, so we can apply the key fact directly to U without having to deal with the translation step.

The proof of the key fact in the infinite-precision model prescribes a "desired" input/output relationship from the intended input U of this node to the intended inputs V, W of its two children. In our finite-precision model we need to replace this relationship by a different input/output relationship, such that when the input matrix has a finite binary representation, the two output matrices have finite binary representations computable by a time-space efficient algorithm. The goal is that when the input $\widehat{U}$ is within δ_ℓ of the intended input U, the outputs $\widehat{V}, \widehat{W}$ are within $\delta_{\ell-1}$ of the intended outputs V, W. In each step of the computation the absolute error may increase. This is partly inherent in the conditioning (e.g., square-rooting a number can square-root the error in the worst case), and partly because of the time-space efficiency requirement (e.g., roundoff errors, or the use of iterative rather than direct methods). Unfortunately, these errors seem to prevent us from accomplishing the above goal[10] with $\alpha = 1.5$.

To circumvent the problem, we revise our goal for the new input/output relationship: when the input $\widehat{U}$ is within δ_ℓ of the intended input U, we now only require that the outputs $\widehat{V}, \widehat{W}$ are within $\delta_{\ell-1}$ of *some* V, W which are the "desired" outputs corresponding to *some* unitary operator

[9] It turns out that using *floating* point numbers does not asymptotically improve the efficiency since it is always the case that a constant fraction of the bits past the radix point are potentially nonzero. Thus we always work with *fixed* point numbers and with absolute errors.

[10] See Section 4.3.2 for more details. In fact, the square roots inherent in the proof of the key fact seem to impose $\alpha \ge 2$, though it is conceivable α could be reduced somewhat by case analysis.

U' that is within $O(\epsilon_\ell)$ of U.[11] This is good enough, because then U' is $O(\epsilon_{\ell-1})$-close to I, and the key fact shows that if $\widetilde{V}, \widetilde{W}$ are unitary operators that $\epsilon_{\ell-1}$-approximate V, W, then $\widetilde{V}\widetilde{W}\widetilde{V}^\dagger\widetilde{W}^\dagger$ is $O(\epsilon_\ell)$-close to U' and hence $O(\epsilon_\ell)$-close to U. This gives us what we wanted, using a small adjustment in parameters to absorb the constant factor.

We now sketch how we accomplish the revised goal with $\alpha = 1.5$ in the case where $d = 2$. For that we need to take a closer look at the proof of the key fact. It uses a correspondence between unitary operators and skew-hermitian operators via the logarithm/exponential maps.[12] Given U, it first converts to the skew-hermitian domain, then finds the desired operators in this domain, then converts back to the unitary domain to get V, W. Let F be the skew-hermitian matrix $\log U$. Given $\widehat{U}$ within δ_ℓ of U, it turns out we can obtain a matrix $\widehat{F}$ that is within $O(\delta_\ell)$ of F in time polylog$(1/\epsilon)$ and space $O(\log(1/\epsilon))$ (see Section 4.3.2). As we mentioned in the above discussion of the original goal, this is not close enough for computing the desired V, W corresponding to U within $\delta_{\ell-1}$ when $\alpha = 1.5$. The key for achieving the revised goal is the following fact.

> Every skew-hermitian operator corresponds exactly to some unitary operator, and nearby skew-hermitian operators correspond to nearby unitary operators.

For $\alpha = 1.5$ we can ensure that $\widehat{F}$ is within $O(\epsilon_\ell)$ of F. Thus, given $\widehat{F}$, if we could find some (finite-precision) skew-hermitian matrix F' within $O(\epsilon_\ell)$ of $\widehat{F}$ and hence within $O(\epsilon_\ell)$ of F, then the corresponding unitary operator $U' = \exp F'$ would be within $O(\epsilon_\ell)$ of U, and we could proceed to compute the desired V, W corresponding to U' within $\delta_{\ell-1}$ since we would have an *exact* representation of F'. To find F', we can take $\widehat{F}$ and perturb it in a natural way to make it skew-hermitian; then using the fact that $\widehat{F}$ is $O(\epsilon_\ell)$-close to *some* skew-hermitian operator (namely F), it can be shown that F' is within $O(\epsilon_\ell)$ of $\widehat{F}$. This is the basic idea for accomplishing the revised goal in the case where $d = 2$. A technical problem arises when $d > 2$; the workaround uses an idea due to Nagy [30] and is presented in Section 4.3.5.

In the end, we set $\delta_\ell = \epsilon_\ell / c$ for some large constant c. Since all the matrices at level ℓ only need to approximate their intended operators within $O(\delta_\ell)$, we can always assume they only take up $O(\log(1/\delta_\ell))$ space. At the top level, this comes out to $O(\log(1/\epsilon))$ space, which is necessary but not sufficient for achieving the desired space bound. Since the constants in the big O's do not depend on ℓ, the space for storing a matrix goes down by a constant factor at each level as we go down the recursion tree. Thus, if we can get by with storing only a constant number of matrices at each node along the current path to the root, then the total space usage will be a geometric sum dominated by $O(\log(1/\delta_L)) \le O(\log(1/\epsilon))$, giving us the desired space bound. We follow this general approach, but there are a number of obstacles to getting it to work.

4.2.3 Reducing the Space in the Overall Architecture

Let us quickly rehash the notation for the algorithm as described so far. For our arbitrary node at level ℓ, we have $U, \widehat{U}, \widetilde{U}$ which denote the intended input, the input, and the product of the sequence of gates output by our algorithm, which ϵ_ℓ-approximates U. For the first child, we have $\Upsilon, \widehat{\Upsilon}, \widetilde{\Upsilon}$ where $\Upsilon = U$, $\widehat{\Upsilon}$ is a truncation of $\widehat{U}$, and $\widetilde{\Upsilon}$ $\epsilon_{\ell-1}$-approximates Υ. For the remaining two children, we have $V, \widehat{V}, \widetilde{V}$ and $W, \widehat{W}, \widetilde{W}$. However, in order to compute $\widehat{V}$ and $\widehat{W}$, we need to

[11] This is related to the notion of *backward stability* from numerical analysis.

[12] Recall that a linear operator F on $\mathbb{C}^d$ is skew-hermitian iff $F^\dagger = -F$, or equivalently, F is unitarily diagonalizable with imaginary eigenvalues.

somehow obtain a matrix $\widehat{U\widetilde{\Upsilon}^\dagger}$ that is sufficiently close to $U\widetilde{\Upsilon}^\dagger$. We discuss this issue in Section 4.2.4 below; for now, let us assume that such a matrix is magically provided to us.

In this section we address the following issue. The algorithm as described in Section 4.2.1 recursively finds the sequences corresponding to $\widetilde{V}$ and $\widetilde{W}$, and it stores these two sequences since they both need to be used twice in the returned sequence (once as is, and once in reverse order with each gate inverted). However, we do not have enough space to store the sequences.

The standard approach for avoiding the space overhead of storing intermediate results in a computation is to recompute the intermediate results whenever they are needed. However, in our case this would increase the running time to at least $2^{\Omega(L^2)}$ (since the sequence for a node at level ℓ has length $2^{\Omega(\ell)}$ and thus the degree of the node would increase to at least $2^{\Omega(\ell)}$), which would defeat our goal of maintaining a polylog$(1/\epsilon)$ running time.[13]

We must ask each recursive call to output its sequence on the fly, in the order the gates are to be applied, rather than returning the sequence for us to manipulate. To generate the sequence corresponding to $\widetilde{U} = \widetilde{V}\widetilde{W}\widetilde{V}^\dagger\widetilde{W}^\dagger\widetilde{\Upsilon}$, we can make a call on $\widehat{\Upsilon}$ to generate the prefix corresponding to $\widetilde{\Upsilon}$, and calls on $\widehat{W}$ then $\widehat{V}$ to generate the suffix corresponding to $\widetilde{V}\widetilde{W}$. We just need to worry about generating the middle part corresponding to $\widetilde{V}^\dagger\widetilde{W}^\dagger$.[14] We augment the procedure with an additional input indicating whether the sequence corresponding to $\widetilde{U}$ should be output "forward" or "inverse" (the latter meaning that the order is reversed and each gate is inverted). Then to obtain the sequence corresponding to $\widetilde{U}$, we can make a forward call on $\widehat{\Upsilon}$, then inverse calls on $\widehat{W}$ then $\widehat{V}$, then forward calls on $\widehat{W}$ then $\widehat{V}$. Unless, of course, the current call is in inverse mode, in which case we should make inverse calls on $\widehat{V}$ then $\widehat{W}$, then forward calls on $\widehat{V}$ then $\widehat{W}$, then an inverse call on $\widehat{\Upsilon}$.

Now, the notation $\widetilde{U}$ refers to the product of the gates that would be output if the call is in forward mode.

4.2.4 Obtaining the Matrix $\widehat{U\widetilde{\Upsilon}^\dagger}$

If we had a matrix $\widehat{\widetilde{\Upsilon}}$ within δ_ℓ of $\widetilde{\Upsilon}$, then we could multiply $\widehat{U}$ with $\widehat{\widetilde{\Upsilon}}^\dagger$ to obtain a matrix within $3\delta_\ell$ of $U\widetilde{\Upsilon}^\dagger$, which is good enough for the computation of $\widehat{V}, \widehat{W}$. But how do we obtain $\widehat{\widetilde{\Upsilon}}$? Note that $\widetilde{\Upsilon}$ can be represented exactly as a sequence of gates from S. However, what we need is (good approximations to) the entries of the matrix $\widetilde{\Upsilon}$, which we must obtain by multiplying (good approximations to) the matrices corresponding to the gates in the sequence that comprises $\widetilde{\Upsilon}$.

A relatively simple way of obtaining $\widehat{\widetilde{\Upsilon}}$ is as follows (see Section 4.3.4 for some more streamlined ways). Before making the five regular recursive calls, we make a forward call on $\widehat{\Upsilon}$ but use a flag to indicate that no actual output should be produced; the algorithm should just "go through the motions" for the purpose of finding $\widehat{\widetilde{\Upsilon}}$. Now at a leaf, there may be several calls on the path to the root that indicated that no output should be produced. If there are *any* such calls, then the leaf

[13] One might naively think that this technique could still be interesting since it might lead to a $o(\log(1/\epsilon))$ space algorithm (if we do not care about the running time) by avoiding even writing down the intermediate matrices. However, this does not work: it turns out that because of the need to recompute $\widehat{U\widetilde{\Upsilon}^\dagger}$, just the recordkeeping for all the backtracking would already take $\omega(\log(1/\epsilon))$ space.

[14] Note that we cannot just make calls on $\widehat{W}^\dagger$ then $\widehat{V}^\dagger$, since a call on $\widehat{W}^\dagger$ would output some sequence whose product approximates $W^\dagger$ but is not guaranteed to be the inverse of $\widetilde{W}$, as required for the mathematics behind the algorithm to go through.

should not produce any actual output.

If we hypothetically made a call on $\widehat{\Upsilon}$ in forward mode with the output flag on, there would be $5^{\ell-1}$ leaves in Υ's subtree that produced actual output (i.e., were called with the output flag on). Let $\Upsilon_1, \ldots, \Upsilon_{5^{\ell-1}}$ and $\widehat{\Upsilon}_1, \ldots, \widehat{\Upsilon}_{5^{\ell-1}}$ denote the intended and actual inputs to these leaves. Then $\widetilde{\Upsilon} = \widetilde{\Upsilon}_{5^{\ell-1}}^{mode_{5^{\ell-1}}} \cdots \widetilde{\Upsilon}_1^{mode_1}$, where $\widetilde{\Upsilon}_i$ denotes the product of the sequence of gates that would be output at Υ_i's leaf if it were hypothetically called in forward mode with the output flag on (so $\widetilde{\Upsilon}_i$ ϵ_0-approximates Υ_i), and $mode_i$ represents either † (if the leaf is in inverse mode) or nothing (if the leaf is in forward mode).

The basic idea is to obtain $\widehat{\widetilde{\Upsilon}}$ by multiplying together $\widehat{\widetilde{\Upsilon}}_{5^{\ell-1}}^{mode_{5^{\ell-1}}} \cdots \widehat{\widetilde{\Upsilon}}_1^{mode_1}$, where $\widehat{\widetilde{\Upsilon}}_i$ approximates $\widetilde{\Upsilon}_i$ within roughly $\delta_\ell / 5^{\ell-1}$ and takes space $O(\log(5^{\ell-1}/\delta_\ell)) \leq O(\log(1/\delta_\ell))$. However, doing this multiplication exactly would take space $O(5^{\ell-1} \cdot \log(1/\delta_\ell))$, which is too large since ℓ is superconstant in general. To keep the space down, we truncate the current matrix to $O(\log(1/\delta_\ell))$ bits after each matrix is multiplied on. See Section 4.3.3 for the calculation proving that the approximation error does not grow too large. When we are at Υ_i's node, we need to immediately multiply $\widehat{\widetilde{\Upsilon}}_i^{mode_i}$ onto the current matrix (then truncate), and therefore the current matrix needs to have been passed down through the recursion to Υ_i's node.[15] The updated matrix must be returned up so that it can then be passed down to Υ_{i+1}'s node.

For any given leaf with input $\widehat{U}$, there may be many nodes along the path to the root that are "interested" in $\widetilde{U}$, at different levels of precision (namely, those nodes that are in the midst of their dummy call, and such that if that dummy call were hypothetically made with the output flag on, then the current leaf would also have the output flag on). We must pass around a list of matrices, one for each such node. The space taken up by this list is a geometric sum dominated by $O(\log(1/\epsilon))$, and we only need to maintain one copy of the list throughout the algorithm (since when the list is passed to a recursive call, the caller does not need to retain a copy). Thus the overall space bound is $O(\log(1/\epsilon))$.

4.3 Proof of Theorem 7

The essence of Theorem 7 is extracted in the following theorem. Let $SU(d)$ denote the set of unitary operators on $\mathbb{C}^d$ with determinant 1.

Theorem 8. *For each constant integer $d \geq 2$ there exists an $\epsilon^* > 0$ such that the following holds. Suppose m is a positive integer and $S \subseteq SU(d)$ is a finite set closed under adjoint such that for every $U \in SU(d)$ there exists a sequence $U_1, \ldots, U_k \in S$ with $k \leq m$ such that $\|U - U_k \cdots U_1\| \leq \epsilon^*$. Then for every $U \in SU(d)$ and every $\epsilon > 0$ there exists a sequence $U_1, \ldots, U_k \in S$ with $k \leq \mathrm{polylog}(1/\epsilon)$ such that $\|U - U_k \cdots U_1\| \leq \epsilon$. Moreover, such a sequence can be computed by a deterministic algorithm running in time $\mathrm{polylog}(1/\epsilon)$ and space $O(\log(1/\epsilon))$, given as input ϵ and matrices that are at distance at most $f(\epsilon)$ from U and the gates in S, where f is a certain polynomial depending only on d.*

The difference between Theorem 7 and Theorem 8 is that in Theorem 8, everything takes place in $SU(d)$ and global phase shifts are not allowed in the approximation guarantees; also, S is only required to be able to approximate an arbitrary U within some fixed ϵ^*, but the length of the approximating sequence must have an upper bound that is independent of U. Also, the

[15] Alternatively, we could keep it in U's "stack frame" and directly modify it there.

f-polynomials in the two theorems are not the same. In both theorems, the finite-precision input matrices are required to approximate the intended unitary operators without global phase factors.

As we show in Section 4.3.1, Theorem 7 follows from Theorem 8. The main part of the reduction deals with the problem that in Theorem 7, S can approximate every U up to global phase factors, whereas in Theorem 8, global phase factors are not allowed. Resolving this issue is easy but not trivial. Although this part of the reduction appears to be necessary for obtaining the Solovay-Kitaev Theorem for arbitrary universal sets, we could not find it in the literature. The remaining part of the reduction is just a technical argument showing that given a finite-precision approximation to an operator in $U(d)$, we can efficiently obtain finite-precision approximations to each global phase shift of the operator that lies in $SU(d)$. This involves using Taylor series and taking some care to ensure the logarithmic space bound.

In Section 4.3.2 and Section 4.3.3 we give the proof of Theorem 8 for the case $d = 2$, to avoid some technical issues that crop up in the general case. Although we do not attempt to optimize the degree of the polylog in the running time, we mention a few ways to reduce this degree in Section 4.3.4. In Section 4.3.5 we explain how to modify the proof to work for arbitrary d.

We break up the proof of Theorem 8 for $d = 2$ into two parts. The first part, given in Section 4.3.2, is a lemma that forms the kernel of the algorithm. This lemma basically says that given any $U \in SU(2)$ such that $\|U - I\| \le \epsilon$, we can produce $V, W \in SU(2)$ such that for every $\widetilde{V}, \widetilde{W} \in SU(2)$ with $\|V - \widetilde{V}\|$, $\|W - \widetilde{W}\| \le \epsilon$, we have $\|U - \widetilde{V}\widetilde{W}\widetilde{V}^\dagger\widetilde{W}^\dagger\| \le O(\epsilon^{1.5})$. Moreover, we can do it with essentially the minimum possible precision — less precision than a naive analysis of the standard Solovay-Kitaev argument would yield. The second part, given in Section 4.3.3, constructs the full algorithm using this key tool as the building block.

Before diving into the formal proof, we discuss two simple but relevant points regarding numerical calculations. First, note that whenever a real number is guaranteed to be δ-close to some intended real number, we can assume it has at most $\lceil \log_2(1/\delta) \rceil + 1$ bits past the radix point, since with a slight change in parameters, we can guarantee that it is $\delta/2$-close to the intended number, and then truncate it while increasing the distance by at most $\delta/2$. A similar statement holds for matrices. Thus, throughout the whole proof, we tacitly assume that every matrix that is δ-close to some intended matrix only takes up space $O(\log(1/\delta))$. Second, we need to do arithmetic calculations on real numbers represented exactly in binary. We use the fact that addition, subtraction, and multiplication of such numbers can be performed by deterministic algorithms running in polynomial time and linear space, and that division and square roots can be computed up to p bits past the radix point by deterministic algorithms running in time polynomial in the input length and p, and space linear in the input length and p.

4.3.1 Reduction from Theorem 7 to Theorem 8

Proof of Theorem 7. Fix a constant integer $d \ge 2$ and let ϵ^* be as in Theorem 8. Let f' denote the f-polynomial from Theorem 8. Henceforth, all constants may depend on d.

Let S be as in Theorem 7, and obtain S' for use in Theorem 8 as follows: for each gate $U \in S$, include in S' all global phase shifts of U that lie in $SU(d)$. (Note that there are exactly d such shifts, namely those corresponding to the dth roots of the complex conjugate of the determinant of U, so S' is finite.) Since S is closed under adjoint and the global phase shifts of U that lie in $SU(d)$ are the adjoints of the global phase shifts of $U^\dagger$ that lie in $SU(d)$, it follows that S' is closed under adjoint.

There exists a finite set $T \subseteq U(d)$ such that every operator in $SU(d)$ is at distance at most

$\epsilon^*/2c$ from some operator in T, where c is a constant to be specified later. By our assumption on S, each operator in T is at distance at most $\epsilon^*/2c$ from a global phase shift of the product of some sequence of gates in S. Let m be the maximum length of this sequence over all operators in T. Then by the triangle inequality, for every $U \in SU(d)$ there is a sequence $U_1, \dots, U_k \in S'$ with $k \leq m$ such that $\|U - e^{i\theta} U_k \cdots U_1\| \leq \epsilon^*/c$ for some global phase factor $e^{i\theta}$.

But there is a problem with the latter approximation: to use Theorem 8, we need $\theta = 0$. We can handle this as follows. It can be verified that $\|U - e^{i\theta} U_k \cdots U_1\| \leq \epsilon^*/c$ implies that the determinant $(e^{i\theta})^d$ of $e^{i\theta} U_k \cdots U_1$ satisfies $|(e^{i\theta})^d - 1| \leq O(\epsilon^*/c)$ and thus $\theta \in 2\pi j/d \pm O(\epsilon^*/c)$ mod 2π, for some $j \in \{0, \dots, d-1\}$. Therefore, $\|e^{i\theta} U_k \cdots U_1 - e^{2\pi i j/d} U_k \cdots U_1\| \leq O(\epsilon^*/c)$, and by the triangle inequality we have $\|U - e^{2\pi i j/d} U_k \cdots U_1\| \leq O(\epsilon^*/c)$. Note that the right side of the latter inequality is at most ϵ^* provided c is large enough. If $j = 0$ then this is just what we want. If $j \neq 0$, then we can turn $e^{2\pi i j/d} U_k \cdots U_1$ into a product of gates from S' by multiplying say U_1 by $e^{-2\pi i j/d}$, which keeps it in S' since in the definition of S' we included *all* appropriate phase shifts of each gate in S.

Now that we have m and S' as required for Theorem 8, consider any $U \in U(d)$ and let $e^{i\phi} U$ be any global phase shift that lies in $SU(d)$. Now provided we can obtain $f'(\epsilon)$-approximations to $e^{i\phi} U$ and the gates in S' in time polylog$(1/\epsilon)$ and space $O(\log(1/\epsilon))$ from $f(\epsilon)$-approximations to U and the gates in S, we can run the algorithm from Theorem 8 to find a sequence $U_1, \dots, U_k \in S'$ with $k \leq \text{polylog}(1/\epsilon)$ such that $\|e^{i\phi} U - U_k \cdots U_1\| \leq \epsilon$ in time polylog$(1/\epsilon)$ and space $O(\log(1/\epsilon))$. Since each U_i is a global phase shift of a gate in S, we can output the labels of the gates in S corresponding to $U_1, \dots, U_k$, and then some global phase shift of the resulting product will be at distance at most ϵ from U.

Thus all we need to do is show that, given a matrix $\widehat{U}$ such that $\|\widehat{U} - U\| \leq f(\epsilon)$ for some (unknown) $U \in U(d)$, we can efficiently compute matrices $\widehat{U}^{(1)}, \dots, \widehat{U}^{(d)}$ such that for each global phase shift $e^{i\theta} U$ that lies in $SU(d)$ there is a j such that $\|\widehat{U}^{(j)} - e^{i\theta} U\| \leq f'(\epsilon)$, where f is a certain polynomial (depending on f').

Note that each of the real numbers comprising $\widehat{U}$ can be assumed to have only $O(\log(1/\epsilon))$ bits. The first step is to exactly compute $\det(\widehat{U})$. This number is within poly(ϵ) of $\det(U)$, for an arbitrarily high degree polynomial, provided $f(\epsilon)$ is small enough. The next step is to approximate the polar angle of $\det(U)$ using either the arcsin function or the arccos function (depending on whether the real or imaginary part of $\det(\widehat{U})$ is smaller in absolute value, to ensure fast convergence of the Taylor series). Computing the Taylor series to $O(\log(1/\epsilon))$ terms ensures that the contribution of the Taylor series truncation to the total absolute error is poly(ϵ). There is another poly(ϵ) contribution to the total error coming from the error in the input approximation. However, we cannot exactly evaluate the first $O(\log(1/\epsilon))$ terms on the approximate input for two reasons: raising an $O(\log(1/\epsilon))$-bit number to the power $O(\log(1/\epsilon))$ would take space $O(\log^2(1/\epsilon))$, and the coefficients of the Taylor series involve division by numbers that are not powers of 2. The former issue can be solved by truncating to $O(\log(1/\epsilon))$ bits after multiplying on each copy of the number; this contributes poly(ϵ) to the total error. The latter issue can be solved by just doing division to $O(\log(1/\epsilon))$ bits; this also contributes poly(ϵ) to the total error.

Now that we have a poly(ϵ)-approximation to the angle of $\det(U)$, we can approximately divide by $-d$ to get a poly(ϵ)-approximation to the angle of one of the correct phase shifts. The other angles can be approximately obtained by adding multiples of $2\pi/d$. To get poly(ϵ)-approximations to the actual phase shifts, approximately convert these angles to the corresponding complex numbers on the unit circle and multiply $\widehat{U}$ by the resulting numbers. This finishes the proof. □

4.3.2 The Kernel ($d = 2$)

Our goal in this section is to prove the following result.

Lemma 2. *There exist constants b, c, and $\epsilon_0 > 0$ such that the following holds. There is a deterministic algorithm that, given parameter $0 < \epsilon \leq \epsilon_0$, runs in time* $\mathrm{polylog}(1/\epsilon)$ *and space* $O(\log(1/\epsilon))$ *and achieves the following. The input is a matrix $\widehat{U}$ promised to have the following property: there exists a $U \in SU(2)$ such that $\big\|\widehat{U} - U\big\| \leq 3b\epsilon^{1.5}/c$ and $\|U - I\| \leq \epsilon$. The output is two matrices $\widehat{V}, \widehat{W}$ having the following property: there exist $V, W \in SU(2)$ such that $\big\|\widehat{V} - V\big\|$, $\big\|\widehat{W} - W\big\| \leq \epsilon/c$ and for every $\widetilde{V}, \widetilde{W} \in SU(2)$ with $\big\|V - \widetilde{V}\big\|$, $\big\|W - \widetilde{W}\big\| \leq \epsilon$, we have $\big\|U - \widetilde{V}\widetilde{W}\widetilde{V}^\dagger\widetilde{W}^\dagger\big\| \leq b\epsilon^{1.5}$.*

We first give some setup for the proof. Recall that $SU(d)$ denotes the set of unitary operators on $\mathbb{C}^d$ with determinant 1 under multiplication, and let $su(d)$ denote the set of skew-hermitian operators on $\mathbb{C}^d$ with trace 0 under addition. The identity in $SU(d)$ is I, and the identity in $su(d)$ is 0. There is a bijection between $U \in SU(d)$ with $\|U - I\| < 2$ and $F \in su(d)$ with $\|F - 0\| < \pi$, given by the logarithm map $\ln : SU(d) \to su(d)$ and the exponential map $\exp : su(d) \to SU(d)$. Throughout this section, we always assume that operators in $SU(d)$ are at distance < 2 from I and that operators in $su(d)$ are at distance $< \pi$ from 0, so that we may move freely between these two domains. In the $SU(d)$ domain we make use of the group commutator, defined as $[V, W]_{gp} = VWV^\dagger W^\dagger$; note that $[V, W]_{gp} \in SU(d)$ if $V, W \in SU(d)$. In the $su(d)$ domain we make use of the commutator, defined as $[G, H] = GH - HG$; note that $[G, H] \in su(d)$ if $G, H \in su(d)$. We also need the following facts relating distances in the two domains.

Fact 2. *If $U_1, U_2 \in SU(d)$ with corresponding $F_1, F_2 \in su(d)$ are such that $\|F_1 - F_2\| \leq \delta$, then $\|U_1 - U_2\| \leq O(\delta)$.*

Fact 3. *If $U \in SU(d)$ with corresponding $F \in su(d)$ is such that $\|U - I\| \leq \delta$, then $\|F - 0\| \leq O(\delta)$.*

Fact 2 follows from Corollary 6.2.32 in [24], which states that for all $d \times d$ complex matrices A and E, $\big\|\exp(A + E) - \exp(A)\big\| \leq \|E\|e^{\|E\|+\|A\|}$. Fact 3 is a partial converse to Fact 2 in the case $U_2 = I$. Actually, the more precise relation $\|U - I\| = 2\sin\left(\frac{1}{2}\|F - 0\|\right)$ holds, but Fact 3 as stated is all we need.

Proof of Lemma 2. We leave b and c free for now and decide how to set them later. We just let $\epsilon_0 > 0$ be small enough to make all the big O's and little o's in the argument work. Throughout this section, constants hidden in big O's may depend on b, c, and d (though it turns out the form of the dependence on b and c matters). Significant portions of the proof work for arbitrary $d \geq 2$ (and are used for the generalization to $d > 2$ described in Section 4.3.5), so we present those portions for general d. Let ϵ, $\widehat{U}$, and U be as in the statement of Lemma 2.

The standard proof of the kernel lemma in the general Solovay-Kitaev Theorem uses the following three structural results.

Fact 4. *If $U \in SU(d)$ with corresponding $F \in su(d)$ is such that $\|U - I\| \leq \delta$, then there exist $V, W \in SU(d)$ with corresponding $G, H \in su(d)$ such that $F = [G, H]$ and $\|V - I\|$, $\|W - I\| \leq O(\delta^{0.5})$.*

Fact 5. *If $U, V, W \in SU(d)$ with corresponding $F, G, H \in su(d)$ are such that $F = [G, H]$ and $\|V - I\|$, $\|W - I\| \leq \delta$, then $\big\|U - [V, W]_{gp}\big\| \leq O(\delta^3)$.*

Fact 6. *If $V, W, \widetilde{V}, \widetilde{W} \in SU(d)$ are such that $\|V - I\|,\ \|W - I\| \leq \delta$ and $\|V - \widetilde{V}\|,\ \|W - \widetilde{W}\| \leq \gamma$, then $\left\|[V, W]_{gp} - \left[\widetilde{V}, \widetilde{W}\right]_{gp}\right\| \leq O(\delta\gamma + \gamma^2)$.*

Proofs of Fact 4 can be found in [11, 26]. Fact 5 follows from basic results in Lie theory,[16] or just using the infinite series for matrix exponentiation. A proof of Fact 6 can be found in [11]. The existence of V, W as in Lemma 2 follows from these three facts. Specifically, let $V, W \in SU(d)$ be as guaranteed by Fact 4 (with $\delta = \epsilon$). Then for every $\widetilde{V}, \widetilde{W} \in SU(d)$ with $\|V - \widetilde{V}\|,\ \|W - \widetilde{W}\| \leq \epsilon$, we have

$$\left\|U - \left[\widetilde{V}, \widetilde{W}\right]_{gp}\right\| \ \leq\ \|U - [V, W]_{gp}\| + \left\|[V, W]_{gp} - \left[\widetilde{V}, \widetilde{W}\right]_{gp}\right\| \ \leq\ O(\epsilon^{1.5}) + O(\epsilon^{1.5}) \ \leq\ O(\epsilon^{1.5})$$

by Fact 5 (with $\delta = \Theta(\epsilon^{0.5})$) and Fact 6 (with $\delta = \Theta(\epsilon^{0.5})$ and $\gamma = \epsilon$).

Fact 4 is the only step that needs to be turned into an algorithm. That is, given an approximation to some $U \in SU(d)$ such that $\|U - I\| \leq \epsilon$, we would like to compute approximations to some $V, W \in SU(d)$ such that $F = [G, H]$ and $\|V - I\|,\ \|W - I\| \leq O(\epsilon^{0.5})$, where $F, G, H \in su(d)$ correspond to U, V, W. However, it turns out that this is a bit too ambitious of a goal: we only have an approximation to U within roughly $\epsilon^{1.5}$, which is not good enough to be able to find approximations to some correct V, W within roughly ϵ, due to loss in precision in the numerical calculations.[17] Instead, we shoot for the following revised goal: to find good enough approximations to some V, W such that $\|V - I\|,\ \|W - I\| \leq O(\epsilon^{0.5})$ and $[G, H] = F'$ where $\|F - F'\| \leq O(\epsilon^{1.5})$. This is sufficient because then by Fact 2, $\|U - U'\| \leq O(\epsilon^{1.5})$ (where $U' \in SU(d)$ corresponds to F') and so for every $\widetilde{V}, \widetilde{W} \in SU(d)$ with $\|V - \widetilde{V}\|,\ \|W - \widetilde{W}\| \leq \epsilon$, we have

$$\left\|U - \left[\widetilde{V}, \widetilde{W}\right]_{gp}\right\| \ \leq\ \|U - U'\| + \left\|U' - \left[\widetilde{V}, \widetilde{W}\right]_{gp}\right\| \ \leq\ O(\epsilon^{1.5}) + O(\epsilon^{1.5}) \ \leq\ O(\epsilon^{1.5})$$

by Fact 5 and Fact 6.

To achieve this revised goal, we follow the natural approach, which is to attempt to compute F' from U, then G, H from F', then V, W from G, H. For the second step, of course, we need to know a very accurate approximation to F' (this is the whole reason for introducing F': we do not have a good enough approximation to F). In fact, in the first step we shall compute a suitable F' exactly. Formally, here are the three steps.

(1) Given $\widehat{U}$ such that $\|\widehat{U} - U\| \leq 3b\epsilon^{1.5}/c$ for some $U \in SU(d)$ with $\|U - I\| \leq \epsilon$, compute $F' \in su(d)$ exactly such that $\|F - F'\| \leq O(\epsilon^{1.5})$, where $F \in su(d)$ corresponds to U.

(2) Given $F' \in su(2)$ such that $\|F' - 0\| \leq O(\epsilon)$, compute $\widehat{G}, \widehat{H}$ such that $\|\widehat{G} - G\|,\ \|\widehat{H} - H\| \leq o(\epsilon)$ for some $G, H \in su(2)$ with $F' = [G, H]$ and $\|G - 0\|,\ \|H - 0\| \leq O(\epsilon^{0.5})$.

(3) Given $\widehat{G}, \widehat{H}$ such that $\|\widehat{G} - G\|,\ \|\widehat{H} - H\| \leq o(\epsilon)$ for some $G, H \in su(d)$ with $\|G - 0\|,\ \|H - 0\| \leq O(\epsilon^{0.5})$, compute $\widehat{V}, \widehat{W}$ such that $\|\widehat{V} - V\|,\ \|\widehat{W} - W\| \leq \epsilon/c$, where $V, W \in SU(d)$ correspond to G, H.

[16] The commutator is the bracket for the Lie algebra $su(d)$ of the Lie group $SU(d)$.

[17] It turns out we need to ensure that the product of $\|G - 0\|$ and $\|H - 0\|$ is roughly $\|F - 0\|$ while keeping these two quantities $\leq O(\epsilon^{0.5})$, so the algorithm must do something tantamount to computing a square root, which could cause the absolute error to get almost square rooted in the worst case.

We have restricted to $d = 2$ in step (2) because this is the only part that does not work for $d > 2$ within the desired space bound. We can stitch these three steps together to accomplish the revised goal (from the previous paragraph) as follows. To connect step (1) to step (2), note that $\|F' - 0\| \leq O(\epsilon)$ since by Fact 3, $\|F - 0\| \leq O(\epsilon)$. Also note that $\|V - I\|$, $\|W - I\| \leq O(\epsilon^{0.5})$ by Fact 2. Thus, as in the previous paragraph, we conclude that for every $\widetilde{V}, \widetilde{W} \in SU(d)$ with $\|V - \widetilde{V}\|$, $\|W - \widetilde{W}\| \leq \epsilon$, we have

$$\left\| U - \left[\widetilde{V}, \widetilde{W}\right]_{gp} \right\| \ \leq \ O(\epsilon^{1.5}).$$

It turns out that the constant in this big O is $a_1 b/c + a_2$ for some constants a_1, a_2 (which may depend on d). We can make this quantity at most b by setting $b = a_2 + 1$ and $c = a_1 b$. This gives us what we want, provided each of the three steps runs in time polylog$(1/\epsilon)$ and space $O(\log(1/\epsilon))$.

We first describe step (3), since it is the simplest. We just compute $\widehat{V} = I + \widehat{G} + \widehat{G}^2/2$ and $\widehat{W} = I + \widehat{H} + \widehat{H}^2/2$. Using the Taylor series for the exponential it can be verified that

$$\left\|\widehat{V} - V\right\| \ \leq \ \left\|\widehat{V} - (I + G + G^2/2)\right\| + \left\|(I + G + G^2/2) - V\right\| \ \leq \ o(\epsilon) + O(\epsilon^{1.5}) \ \leq \ \epsilon/c,$$

and similarly for W. This takes care of step (3).

We now describe step (1). First compute $\widehat{F} = \widehat{U} - I$. Using the Taylor series for the exponential it can be verified that

$$\left\|\widehat{F} - F\right\| \ \leq \ \left\|\widehat{F} - (U - I)\right\| + \left\|(U - I) - F\right\| \ \leq \ O(\epsilon^{1.5}) + O(\epsilon^2) \ \leq \ O(\epsilon^{1.5}).$$

If $\widehat{F} \in su(d)$ then we can take $F' = \widehat{F}$ and be done. Otherwise, we obtain $F' \in su(d)$ from $\widehat{F}$ as follows. To make it skew-hermitian, we set the real parts of the diagonal entries to 0, and forget the entries below the diagonal and replace them with the negative complex conjugates of the corresponding entries above the diagonal. Then, to make it have trace 0, we replace an arbitrary diagonal entry with the negative sum of the other diagonal entries. Using the facts that $F \in su(d)$ and $\left\|\widehat{F} - F\right\| \leq O(\epsilon^{1.5})$, it can be verified that $\left\|\widehat{F} - F'\right\| \leq O(\epsilon^{1.5})$ and thus $\|F - F'\| \leq O(\epsilon^{1.5})$. This takes care of step (1).

Finally, we describe step (2). This is the only part where we must restrict our attention to $d = 2$. When $d > 2$, we do not know how to do this within the target space efficiency, due to significant roundoff errors resulting from divisions in the standard matrix diagonalization algorithms (but there is a workaround, which we discuss in Section 4.3.5). When $d = 2$, a single step of the Jacobi eigenvalue algorithm [20] can be used to diagonalize F', and then an algorithm meeting our efficiency constraints can be gleaned from the proof in [11] of Fact 4 (taking some care to avoid significant roundoff errors). Rather than give more details about this approach, we describe a more direct and elegant approach using so-called Pauli vectors (based on the proof of the Solovay-Kitaev Theorem for $d = 2$ in [31]).

Let $f' = (f'_X, f'_Y, f'_Z) \in \mathbb{R}^3$ be such that $\left(0, -\frac{i}{2}f'_X, -\frac{i}{2}f'_Y, -\frac{i}{2}f'_Z\right)$ are the coordinates of F' in the basis of Pauli matrices I, X, Y, Z (note that since $F' \in su(2)$, the I-coordinate must be 0 and the others must be imaginary). Then f' is called the *Pauli vector* of F' (or of U'). For all $G, H \in su(2)$, the Pauli vector of $[G, H]$ is the cross product of the Pauli vectors of G and H (see Appendix 3 of [31]).

The Euclidean distance between Pauli vectors is within constant factors of the distance between the associated operators in $su(2)$. Thus $\|f'\| \leq O(\epsilon)$, and we seek Pauli vectors $g, h \in \mathbb{R}^3$ (associated with some $G, H \in su(2)$) such that $g \times h = f'$ and $\|g\|,\ \|h\| \leq O(\epsilon^{0.5})$. Further, given F' we can compute f' by a simple change of basis, and given vectors $\widehat{g}, \widehat{h} \in \mathbb{R}^3$ such that $\|\widehat{g} - g\|,\ \|\widehat{h} - h\| \leq o(\epsilon)$, we can do the reverse change of basis to obtain matrices $\widehat{G}, \widehat{H}$ such that $\|\widehat{G} - G\|,\ \|\widehat{H} - H\| \leq o(\epsilon)$.[18] Thus, the bottom line is the following: given $f' \in \mathbb{R}^3$ we wish to compute $\widehat{g}, \widehat{h} \in \mathbb{R}^3$ such that $\|\widehat{g} - g\|,\ \|\widehat{h} - h\| \leq o(\epsilon)$ for some $g, h \in \mathbb{R}^3$ such that f', g, h are mutually perpendicular and $\|g\|,\ \|h\| = \|f'\|^{0.5}$.

If $f' = (0, 0, 0)$ then this is trivial; otherwise, assume for example that either $f'_Y \neq 0$ or $f'_Z \neq 0$. Then let $g' = f' \times (1, 0, 0)$ and $h' = f' \times g'$, and note that we can compute g' and h' exactly. Define $g = \|f'\|^{0.5} \cdot\ g'\ /\ \|g'\|$ and $h = \|f'\|^{0.5} \cdot\ h'\ /\ \|h'\|$. We show how to obtain an approximation $\widehat{g}_X$ to g_X; the cases of other coordinates, as well as h, are symmetric. Note that $g_X = \left(\operatorname{sgn} g'_X\right) \cdot \left(\|f'\|\ \cdot\ (g'_X)^2\ /\ \|g'\|^2\right)^{0.5}$. We can approximate this as follows. Compute $(g'_X)^2$ and $\|g'\|^2$ exactly and take the quotient to within ϵ^3. Compute $\|f'\|^2$ exactly and take the square root to within ϵ^3. Then take the product of these two approximations; the result approximates $\|f'\|\ \cdot\ (g'_X)^2\ /\ \|g'\|^2$ within $O(\epsilon^3)$. Then take the square root within $\epsilon^{1.5}$, and finally multiply by $\operatorname{sgn} g'_X$. The result is within $\sqrt{O(\epsilon^3)} + \epsilon^{1.5} \leq o(\epsilon)$ of g_X.

Analyzing the running time and space usage of the algorithm is straightforward. Each real number the algorithm uses can be written with $O(\log(1/\epsilon))$ bits past the radix point, and so arithmetic operations on these numbers can all be computed in time $\operatorname{polylog}(1/\epsilon)$ and space $O(\log(1/\epsilon))$. This concludes the proof of Lemma 2. □

4.3.3 The Full Algorithm ($d = 2$)

Proof of Theorem 8 ($d = 2$). Let $f(\epsilon)$ be a certain polynomial, which we can take to be a sufficiently high power of ϵ. Let b, c, and $\epsilon_0 > 0$ be the constants guaranteed by Lemma 2. We can assume $b\epsilon_0^{1.5} < \epsilon_0 < 1$ and $c \geq 4$ since in Lemma 2 we can take ϵ_0 arbitrarily small and c arbitrarily large. For integers $\ell > 0$ define $\epsilon_\ell = b\epsilon_{\ell-1}^{1.5}$. Let $\epsilon^* = \epsilon_0/c$. Let S and m be as hypothesized in Theorem 8. Let `compute-VW`$(\ell, \widehat{U})$ be the algorithm from Lemma 2 that takes $\widehat{U}$ and outputs $\widehat{V}$ and $\widehat{W}$, using $\epsilon = \epsilon_{\ell-1}$. Then we claim that Algorithm 1 witnesses Theorem 8.

Algorithm 1: Algorithm for Theorem 8

```
Input: parameter $\epsilon > 0$, matrices at distance at most $f(\epsilon)$ from $U$ and the gates in $S$
Output: sequence $U_1, \ldots, U_k \in S$ such that $\|U - U_k \cdots U_1\| \leq \epsilon$

let $L \geq 0$ be the smallest integer such that $\epsilon_L \leq \epsilon$
let matrix $\widehat{U}$ be such that $\|\widehat{U} - U\| \leq \epsilon_L/c$
compute-sequence($L$, $\widehat{U}$, forward, on)
```

In Algorithm 1, the procedure `compute-sequence` (which is given in a separate figure) takes an integer $\ell \geq 0$ and a finite-precision matrix $\widehat{U}$ which is close to some "intended" operator $U \in SU(2)$ and finds a sequence of gates from S whose product $\widetilde{U}$ satisfies $\|U - \widetilde{U}\| \leq \epsilon_\ell$. The procedure also takes as input *mode*, indicating whether the sequence should be inverted (see Section 4.2.3), and *flag*, indicating whether the procedure should output the sequence (see Section 4.2.4). It also takes finite-precision matrices $\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}$ where $\widehat{M}_j$ is associated with the node at level j on the current path to the root in the recursion tree. If the latter node is in the process of computing an

[18] It happens to be the case that $\widehat{G}, \widehat{H} \in su(2)$ and $\widehat{g}, \widehat{h}$ are their Pauli vectors, but this is immaterial for us.

```
Procedure compute-sequence($\ell$, $\widehat{U}$, $mode$, $flag$, $\widehat{M}_L$, ..., $\widehat{M}_{\ell+1}$)
  Input: integer $0 \le \ell \le L$, matrix $\widehat{U}$, $mode \in \{forward, inverse\}$, $flag \in \{on, off\}$, list of
         matrices $\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}$
  Output: sequence of gates from $S$ satisfying the properties in Claim 3
  Returns: updated list of matrices $\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}$ satisfying the properties in Claim 3
  if $\ell = 0$ then
      find $U_1, \ldots, U_k \in S$ with $k \le m$ such that $\|\widehat{U} - \widetilde{U}\| \le \epsilon^* + 2\epsilon_0/c$ where $\widetilde{U} = U_k \cdots U_1$
      if $flag = on$ then
          if $mode = forward$ then output the labels of $U_1, \ldots, U_k$
          else if $mode = inverse$ then output the labels of $U_k^\dagger, \ldots, U_1^\dagger$
      end
      for $j \leftarrow 1$ to $L$ do
          if $\widehat{M}_j \neq ?$ then
              compute $\widehat{\widetilde{U}}$ such that $\|\widehat{\widetilde{U}} - \widetilde{U}\| \le \epsilon_j/(4c \cdot 5^{j-1})$
              if $mode = forward$ then $\widehat{M}_j \leftarrow \widehat{\widetilde{U}}\widehat{M}_j$ truncated to $O(\log(1/\epsilon_j))$ bits
              else if $mode = inverse$ then $\widehat{M}_j \leftarrow \widehat{\widetilde{U}}^\dagger \widehat{M}_j$ truncated to $O(\log(1/\epsilon_j))$ bits
          end
      end
  else if $\ell > 0$ then
      $\widehat{\Upsilon} \leftarrow \widehat{U}$ truncated to $O(\log(1/\epsilon_{\ell-1}))$ bits
      $(?, \ldots, ?, \widehat{\widetilde{\Upsilon}}) \leftarrow$ compute-sequence($\ell - 1$, $\widehat{\Upsilon}$, $forward$, $off$, $?, \ldots, ?, I$)
      $(\widehat{V}, \widehat{W}) \leftarrow$ compute-VW($\ell$, $\widehat{U}\widehat{\widetilde{\Upsilon}}^\dagger$)
      if $mode = forward$ then
          $(\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}, ?) \leftarrow$ compute-sequence($\ell - 1$, $\widehat{\Upsilon}$, $forward$, $flag$, $\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}, ?$)
          $(\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}, ?) \leftarrow$ compute-sequence($\ell - 1$, $\widehat{W}$, $inverse$, $flag$, $\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}, ?$)
          $(\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}, ?) \leftarrow$ compute-sequence($\ell - 1$, $\widehat{V}$, $inverse$, $flag$, $\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}, ?$)
          $(\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}, ?) \leftarrow$ compute-sequence($\ell - 1$, $\widehat{W}$, $forward$, $flag$, $\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}, ?$)
          $(\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}, ?) \leftarrow$ compute-sequence($\ell - 1$, $\widehat{V}$, $forward$, $flag$, $\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}, ?$)
      else if $mode = inverse$ then
          $(\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}, ?) \leftarrow$ compute-sequence($\ell - 1$, $\widehat{V}$, $inverse$, $flag$, $\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}, ?$)
          $(\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}, ?) \leftarrow$ compute-sequence($\ell - 1$, $\widehat{W}$, $inverse$, $flag$, $\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}, ?$)
          $(\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}, ?) \leftarrow$ compute-sequence($\ell - 1$, $\widehat{V}$, $forward$, $flag$, $\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}, ?$)
          $(\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}, ?) \leftarrow$ compute-sequence($\ell - 1$, $\widehat{W}$, $forward$, $flag$, $\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}, ?$)
          $(\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}, ?) \leftarrow$ compute-sequence($\ell - 1$, $\widehat{\Upsilon}$, $inverse$, $flag$, $\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}, ?$)
      end
  end
  return $\widehat{M}_L, \ldots, \widehat{M}_{\ell+1}$
```

approximation to its own $\widetilde{\Upsilon}$, it does so by multiplying together a sequence of matrices and truncating after each is multiplied on (see Section 4.2.4), and $\widehat{M_j}$ represents the current intermediate value in this computation. The input $\widehat{M_j}$ can be the symbol ?, which indicates that the node at level j is not "interested" in $\widetilde{U}$. The correctness of Algorithm 1 follows immediately from the following inductive claim. We analyze the time and space complexity after the proof of this claim.

Claim 3. *For every integer $0 \le \ell \le L$ and every matrix $\widehat{U}$ with $\big\|\widehat{U} - U\big\| \le \epsilon_\ell/c$ for some $U \in SU(2)$, there exists a $\widetilde{U} \in SU(2)$ with $\big\|U - \widetilde{U}\big\| \le \epsilon_\ell$ such that if we execute*

$$\texttt{compute-sequence(}\ell,\ \widehat{U},\ \mathit{mode},\ \mathit{flag},\ \widehat{M_L},\ \ldots,\ \widehat{M_{\ell+1}}\texttt{)}$$

then the following three properties hold.

(i) If flag = off then no output is produced.

(ii) If flag = on then the output is a sequence of gates from S whose product equals $\widetilde{U}$ (if mode = forward) or $\widetilde{U}^\dagger$ (if mode = inverse).

(iii) For each $j \in \{\ell+1, \ldots, L\}$, if $\widehat{M_j} \neq$? and $\big\|\widehat{M_j} - M_j\big\| \le \delta$ for some $M_j \in SU(2)$ and some $\delta \ge 0$, then the returned value of $\widehat{M_j}$ satisfies $\widehat{M_j} \neq$? and

$$\big\|\widehat{M_j} - \widetilde{U} M_j\big\| \ \le \ (1+\delta)\big(1 + \epsilon_j/(2c \cdot 5^{j-1})\big)^{5^\ell} - 1 \qquad \text{(if } \mathit{mode} = \mathit{forward}\text{)}$$

or

$$\big\|\widehat{M_j} - \widetilde{U}^\dagger M_j\big\| \ \le \ (1+\delta)\big(1 + \epsilon_j/(2c \cdot 5^{j-1})\big)^{5^\ell} - 1 \qquad \text{(if } \mathit{mode} = \mathit{inverse}\text{).}$$

Proof of Claim 3. We begin with the base case $\ell = 0$. We know that there exists a sequence $U_1, \ldots, U_k \in S$ with $k \le m$ such that $\big\|\widehat{U} - U_k \cdots U_1\big\| \le \epsilon^* + \epsilon_0/c$ since there exists a sequence $U_1, \ldots, U_k \in S$ with $k \le m$ such that $\big\|U - U_k \cdots U_1\big\| \le \epsilon^*$ by our assumption on S. Thus by trying all possible sequences (using the fact that S is finite) and considering a (hardcoded) finite approximation to each sequence within a sufficiently small constant, the algorithm can find a sequence $U_1, \ldots, U_k \in S$ with $k \le m$ such that $\big\|\widehat{U} - U_k \cdots U_1\big\| \le \epsilon^* + 2\epsilon_0/c$, and thus line 2 succeeds. By the triangle inequality, this sequence satisfies $\big\|U - U_k \cdots U_1\big\| \le \epsilon^* + 3\epsilon_0/c = 4\epsilon_0/c \le \epsilon_0$ since $c \ge 4$. Thus properties (i) and (ii) hold by lines 3-6.

To verify property (iii), fix $j \in \{1, \ldots, L\}$ and assume $\widehat{M_j} \neq$? and $\big\|\widehat{M_j} - M_j\big\| \le \delta$ for some $M_j \in SU(2)$ and some $\delta \ge 0$. Line 9 can be accomplished using the known matrices at distance at most $f(\epsilon)$ from the gates in S. Suppose *mode* = *forward* (the case *mode* = *inverse* is similar). Let $\widehat{M_j'}$ denote the returned value of $\widehat{M_j}$ (computed in line 10). To show that

$$\big\|\widehat{M_j'} - \widetilde{U} M_j\big\| \ \le \ (1+\delta)\big(1 + \epsilon_j/(2c \cdot 5^{j-1})\big)^{5^0} - 1,$$

by the triangle inequality it suffices to show the following three inequalities.

$$\big\|\widehat{M_j'} - \widehat{\widetilde{U}}\widehat{M_j}\big\| \ \le \ (1+\delta) \cdot \epsilon_j/(4c \cdot 5^{j-1}) \tag{3}$$

$$\big\|\widehat{\widetilde{U}}\widehat{M_j} - \widetilde{U}\widehat{M_j}\big\| \ \le \ (1+\delta) \cdot \epsilon_j/(4c \cdot 5^{j-1}) \tag{4}$$

$$\big\|\widetilde{U}\widehat{M_j} - \widetilde{U} M_j\big\| \ \le \ \delta \tag{5}$$

To prove Inequality (3), note that truncating to $O(\log(1/\epsilon_j))$ bits with a suitably large constant in the big O ensures that the left side is actually at most $\epsilon_j/(4c \cdot 5^{j-1})$. Inequality (4) follows from the facts that $\|\widehat{\widetilde{U}} - \widetilde{U}\| \leq \epsilon_j/(4c \cdot 5^{j-1})$ and $\|\widehat{M_j}\| \leq (1+\delta)$ (the latter following from the facts that $\|M_j\| = 1$ and $\|\widehat{M_j} - M_j\| \leq \delta$). Inequality (5) follows from the facts that $\|\widetilde{U}\| = 1$ and $\|\widehat{M_j} - M_j\| \leq \delta$.

We now carry out the induction step. Assuming the claim holds for $\ell - 1$, we prove it for ℓ. Property (i) follows trivially from the induction hypothesis, since if *flag* = *off* then all recursive calls are made with *flag* = *off*.

By choosing a suitably large constant in the big O on line 15, we have $\|\widehat{\Upsilon} - \Upsilon\| \leq \epsilon_{\ell-1}/c$ where $\Upsilon = U$. Let $\widetilde{\Upsilon} \in SU(2)$ be the operator (depending only on $\widehat{\Upsilon}$ and $\ell - 1$) guaranteed by the induction hypothesis. Then $\|\Upsilon - \widetilde{\Upsilon}\| \leq \epsilon_{\ell-1}$, and $\widehat{\widetilde{\Upsilon}}$ found on line 16 satisfies

$$\|\widehat{\widetilde{\Upsilon}} - \widetilde{\Upsilon}\| \;\leq\; \big(1 + \epsilon_\ell/(2c \cdot 5^{\ell-1})\big)^{5^{\ell-1}} - 1 \;\leq\; e^{\epsilon_\ell/2c} - 1 \;\leq\; (1 + \epsilon_\ell/c) - 1 \;=\; \epsilon_\ell/c$$

(using $\widehat{M_\ell} = M_\ell = I$ and $\delta = 0$). Also, no output is produced by the call on line 16.

It follows that $\|U\widetilde{\Upsilon}^\dagger - I\| \leq \epsilon_{\ell-1}$ and

$$\|\widehat{U}\widehat{\widetilde{\Upsilon}}^\dagger - U\widetilde{\Upsilon}^\dagger\| \;\leq\; \|\widehat{U}\widehat{\widetilde{\Upsilon}}^\dagger - \widehat{U}\widetilde{\Upsilon}^\dagger\| + \|\widehat{U}\widetilde{\Upsilon}^\dagger - U\widetilde{\Upsilon}^\dagger\| \;\leq\; \|\widehat{U}\| \cdot \|\widehat{\widetilde{\Upsilon}}^\dagger - \widetilde{\Upsilon}^\dagger\| + \|\widehat{U} - U\| \;\leq\; 3\epsilon_\ell/c$$

since $\|\widehat{U}\| \leq \|U\| + \|\widehat{U} - U\| \leq 1 + \epsilon_\ell/c \leq 2$ and $\|\widehat{\widetilde{\Upsilon}}^\dagger - \widetilde{\Upsilon}^\dagger\| \leq \epsilon_\ell/c$ and $\|\widehat{U} - U\| \leq \epsilon_\ell/c$. By Lemma 2 (using $\widehat{U}\widehat{\widetilde{\Upsilon}}^\dagger$ in place of $\widehat{U}$, $U\widetilde{\Upsilon}^\dagger$ in place of U, and $\epsilon_{\ell-1}$ in place of ϵ), there exist $V, W \in SU(2)$ such that $\|\widehat{V} - V\|$, $\|\widehat{W} - W\| \leq \epsilon_{\ell-1}/c$ (where $\widehat{V}, \widehat{W}$ are as computed on line 17) and for every $\widetilde{V}, \widetilde{W} \in SU(2)$ with $\|V - \widetilde{V}\|$, $\|W - \widetilde{W}\| \leq \epsilon_{\ell-1}$, we have

$$\|U\widetilde{\Upsilon}^\dagger - \widetilde{V}\widetilde{W}\widetilde{V}^\dagger\widetilde{W}^\dagger\| \;\leq\; \epsilon_\ell.$$

In particular, this last inequality holds for the operators $\widetilde{V}, \widetilde{W}$ whose existence is guaranteed by the induction hypothesis applied to $\widehat{V}$ and $\widehat{W}$. Defining $\widetilde{U} = \widetilde{V}\widetilde{W}\widetilde{V}^\dagger\widetilde{W}^\dagger\widetilde{\Upsilon}$, we have $\|U - \widetilde{U}\| \leq \epsilon_\ell$. Property (ii) is now immediate from lines 19-23 (if *mode* = *forward*) or 25-29 (if *mode* = *inverse*), using the induction hypothesis applied to $\widehat{\Upsilon}$, $\widehat{V}$, and $\widehat{W}$.

To verify property (iii), fix $j \in \{\ell+1, \ldots, L\}$ and assume $\widehat{M_j} \neq ?$ and $\|\widehat{M_j} - M_j\| \leq \delta$ for some $M_j \in SU(2)$ and some $\delta \geq 0$. Assume *mode* = *forward* (the case *mode* = *inverse* is similar). After line 19, by the induction hypothesis applied to $\widehat{\Upsilon}$, we have

$$\|\widehat{M_j} - \widetilde{\Upsilon}M_j\| \;\leq\; (1+\delta)\big(1 + \epsilon_j/(2c \cdot 5^{j-1})\big)^{5^{\ell-1}} - 1.$$

After line 20, by the induction hypothesis applied to $\widehat{W}$ (with $\widetilde{\Upsilon}M_j$ in place of M_j and $(1+\delta)\big(1 + \epsilon_j/(2c \cdot 5^{j-1})\big)^{5^{\ell-1}} - 1$ in place of δ), we have

$$\|\widehat{M_j} - \widetilde{W}^\dagger\widetilde{\Upsilon}M_j\| \;\leq\; (1+\delta)\big(1 + \epsilon_j/(2c \cdot 5^{j-1})\big)^{2 \cdot 5^{\ell-1}} - 1.$$

Continuing with lines 21, 22, and 23, we find that the returned value of $\widehat{M_j}$ satisfies

$$\|\widehat{M_j} - \widetilde{V}\widetilde{W}\widetilde{V}^\dagger\widetilde{W}^\dagger\widetilde{\Upsilon}M_j\| \;\leq\; (1+\delta)\big(1 + \epsilon_j/(2c \cdot 5^{j-1})\big)^{5^{\ell}} - 1.$$

Since $\widetilde{V}\widetilde{W}\widetilde{V}^\dagger\widetilde{W}^\dagger\widetilde{\Upsilon} = \widetilde{U}$, we are done. This concludes the proof of Claim 3. □

We now analyze the time and space complexity of Algorithm 1. We start with the space complexity.

Each of the matrices the algorithm needs to deal with (the ones with a$\widehat{\ }$) is associated with some level. (For example, $\widehat{M_j}$ is at level j, $\widehat{\widehat{U}}$ on line 9 is at level j, and for lines 15-17, $\widehat{U}$ and $\widehat{\widehat{\Upsilon}}$ are at level ℓ while $\widehat{\Upsilon}$, $\widehat{V}$, and $\widehat{W}$ are at level $\ell-1$.) Each matrix at level ℓ is poly(ϵ_ℓ)-close to some intended unitary operator and thus only takes space $O(\log(1/\epsilon_\ell))$. By Lemma 2, the call on line 17 only takes space $O(\log(1/\epsilon_{\ell-1})) \leq O(\log(1/\epsilon))$. Now we just need to verify that at every point during the execution of the algorithm, each level ℓ has at most a constant number of matrices associated with it; then the total space usage is $\sum_{\ell=0}^{L} O(\log(1/\epsilon_\ell)) \leq O(\log(1/\epsilon))$.

Each node on the path to the root only needs to keep track of its input $\widehat{U}$, as well as $\widehat{\widehat{\Upsilon}}$, $\widehat{\Upsilon}$, $\widehat{V}$, and $\widehat{W}$, so this is fine. Regarding the list of matrices $\widehat{M_L}, \ldots, \widehat{M_1}$, we only need to store one matrix $\widehat{M_j}$ at a time, for each j. This is because for the call on line 16, we can store $\widehat{M_L}, \ldots, \widehat{M_{\ell+1}}$ and these matrices are never touched during the execution of that call. For the calls on lines 19-29, we pass the list $\widehat{M_L}, \ldots, \widehat{M_{\ell+1}}$ to each call and do not need to store the old values while the call executes.

It is straightforward to verify that the total running time is at most polylog$(1/\epsilon)$: the processing time for each node in the recursion tree is polylog$(1/\epsilon)$, and the tree has depth $O(\log\log(1/\epsilon))$ and each node has six children, so the size of the recursion tree is polylog$(1/\epsilon)$. This finishes the proof of Theorem 8 for the case $d = 2$. □

4.3.4 Optimizations

Although we did not focus on optimizing the degree of the polylog in the running time, we now briefly mention a few simple optimizations. First, when procedure `compute-sequence` is called with *mode* = *forward*, the recursive call on line 16 can be folded into the call on line 19 as follows.

$$\left(\widehat{M_L}, \ldots, \widehat{M_{\ell+1}}, \widehat{\widehat{\Upsilon}}\right) \leftarrow \texttt{compute-sequence(}\ell-1,\ \widehat{\Upsilon},\ \textit{forward},\ \textit{flag},\ \widehat{M_L},\ \ldots,\ \widehat{M_{\ell+1}}, I\texttt{)}$$

(Then the call to `compute-VW` must come after this line.) This saves one recursive call when *mode* = *forward*, but six calls are still used when *mode* = *inverse*.

As a further optimization, we show how to modify the algorithm so that five calls suffice in both cases (at a constant factor cost in space). To achieve this, first interchange V and $V^\dagger$ (as well as W and $W^\dagger$) throughout the algorithm and the proof. Then $\widetilde{U} = \widetilde{V}^\dagger \widetilde{W}^\dagger \widetilde{V} \widetilde{W} \widetilde{\Upsilon}$, and the forward case first makes forward calls on $\widehat{\Upsilon}$ then $\widehat{W}$ then $\widehat{V}$, then inverse calls on $\widehat{W}$ then $\widehat{V}$, while the inverse case first makes forward calls on $\widehat{V}$ then $\widehat{W}$, then inverse calls on $\widehat{V}$ then $\widehat{W}$ then $\widehat{\Upsilon}$. Note that now all forward calls precede all inverse calls. If *mode* = *forward*, then as we pointed out above, $\widehat{\widehat{\Upsilon}}$ can be obtained through the first recursive call without the need for a special dummy call. The point of the reshuffling is the following: we claim that when *mode* = *inverse*, the desired matrix $\widehat{\widehat{\Upsilon}}$ must have been computed at some time in the past, so we can remember it rather than making a dummy call, and further, the total space for all matrices that need to be "in storage" at any point during the computation is only $O(\log(1/\epsilon))$. To see this, first note that for an intended input U to a node at level ℓ, the algorithm finds operators, each expressed as a sequence of gates from S, that approximate U within $\epsilon_0, \ldots, \epsilon_{\ell-1}$ at levels $0, \ldots, \ell-1$, as well as finite-precision matrices that approximate these operators within the corresponding ϵ_{j+1}/c. Suppose the algorithm is modified so that whenever an inverse call is made, the list of these finite-precision matrices is provided as additional input. Then the desired $\widehat{\widetilde{\Upsilon}}$ can be read off as the last matrix in the list, but we now must check that the invariant can be maintained. When an inverse call makes its final call, which is also an inverse call, it can just pass on the first $\ell - 1$ matrices in the list. All other inverse calls that can occur are of $\widehat{V}$-type or $\widehat{W}$-type. For these cases, the node that made the call previously made a forward call on the same $\widehat{V}$ or $\widehat{W}$ (by the reshuffling). This node can store the list for V and the list for W, which were computed during these forward calls, and then pass on the lists to the corresponding inverse calls. By a geometric sum, the space to store these two lists is $O(\log(1/\epsilon_\ell))$ where ℓ is the level of the node. Since we only need to store a pair of lists at each node along the current path to the root, by another geometric sum we find that the total space overhead of these lists is $O(\log(1/\epsilon))$.

Kitaev [26] has shown that the length of the output sequence in the standard Solovay-Kitaev algorithm can be reduced to $O(\log^{3+\delta}(1/\epsilon))$ for every positive constant δ. He accomplishes this by rearranging the two main ingredients (the translation step from U to $U\widetilde{\Upsilon}^\dagger$ to get into the neighborhood of the identity, and the kernel lemma, which only works in the neighborhood of the identity) in a much more careful way, using the fact that the kernel lemma produces operators that are somewhat close to the identity. This leads to a complicated recursion tree.

4.3.5 Generalization to Arbitrary Dimensions

When $d > 2$ we do not know how to prove the kernel lemma (Lemma 2) with the $O(\log(1/\epsilon))$ space bound, due to the apparent need to diagonalize a skew-hermitian matrix: the known numerical methods for doing this seem to need to store very accurate approximations (requiring large space) in order to combat roundoff errors from the divisions. However, when $d > 2$ we can prove a result similar to Lemma 2 but in which the algorithm produces ϵ/c-approximations to *four* operators $V_1, W_1, V_2, W_2 \in SU(d)$ such that for every $\widetilde{V}_1, \widetilde{W}_1, \widetilde{V}_2, \widetilde{W}_2 \in SU(2)$ with $\|V_1 - \widetilde{V}_1\|$, $\|W_1 - \widetilde{W}_1\|$, $\|V_2 - \widetilde{V}_2\|$, $\|W_2 - \widetilde{W}_2\| \leq \epsilon$, we have $\|U - \widetilde{V}_1\widetilde{W}_1\widetilde{V}_1^\dagger\widetilde{W}_1^\dagger\widetilde{V}_2\widetilde{W}_2\widetilde{V}_2^\dagger\widetilde{W}_2^\dagger\| \leq b\epsilon^{1.5}$. (In this result b, c, and ϵ_0 may depend on d.) The idea is due to Nagy [30], and it allows us to bypass the diagonalization and replace it with simpler calculations which can be performed in small space with sufficient accuracy (at a cost in the degree of the running time).[19] We now describe this modification to Lemma 2; the overall architecture discussed in Section 4.3.3 is then trivial to adapt.

Recall that the problem is step (2): given a matrix $F' \in su(d)$ such that $\|F' - 0\| \leq O(\epsilon)$, we would like to compute good approximations to some $G, H \in su(d)$ such that $F' = [G, H]$ and $\|G - 0\|$, $\|H - 0\| \leq O(\epsilon^{0.5})$. There are proofs in [11, 26] that such G, H exist. The proof in [11] involves converting to an orthonormal basis in which all diagonal entries in F' are 0 (an off-diagonal matrix), doing some simple manipulations, and then converting back to the computational basis. Converting to an off-diagonal matrix can be done by first diagonalizing and then conjugating by a Fourier matrix.

Conjugation by a Fourier matrix does not present any numerical problems, and neither do the simple manipulations. Thus the diagonalization is the only obstacle. Nagy's idea is to write $F' = F'_1 + F'_2$ where $F'_1 \in su(d)$ is the diagonal part in the computational basis and $F'_2 \in su(d)$ is the off-diagonal part in the computational basis. Then $\|F'_1 - 0\|$, $\|F'_2 - 0\| \leq O(\epsilon)$, and we can decompose $F'_1 = [G_1, H_1]$ and $F'_2 = [G_2, H_2]$ where $\|G_1 - 0\|$, $\|H_1 - 0\|$, $\|G_2 - 0\|$, $\|H_2 - 0\| \leq O(\epsilon^{0.5})$ and obtain $o(\epsilon)$-approximations to G_1, H_1, G_2, H_2 in time polylog$(1/\epsilon)$ and space $O(\log(1/\epsilon))$ using

[19] Nagy's motivation was that diagonalization is generally not available in implementations of systems for symbolic computation.

simple numerical calculations. What good is this? Suppose $U', U_1', U_2' \in SU(d)$ correspond to F', F_1', F_2'. Then using the fact that $\|F_1' - 0\|,\ \|F_2' - 0\| \leq O(\epsilon)$, it can be shown (Problem 8.16 in [26]) that $\|U' - U_1'U_2'\| \leq O(\epsilon^2)$. It follows from Fact 5 and Fact 6 that for every $\widetilde{V}_1, \widetilde{W}_1 \in SU(d)$ with $\|V_1 - \widetilde{V}_1\|,\ \|W_1 - \widetilde{W}_1\| \leq \epsilon$ (where as usual $V_1, W_1 \in SU(d)$ correspond to G_1, H_1), we have

$$\left\|U_1' - \left[\widetilde{V}_1, \widetilde{W}_1\right]_{gp}\right\| \ \leq\ \|U_1' - [V_1, W_1]_{gp}\| + \left\|[V_1, W_1]_{gp} - \left[\widetilde{V}_1, \widetilde{W}_1\right]_{gp}\right\| \ \leq\ O(\epsilon^{1.5}) + O(\epsilon^{1.5}) \ \leq\ O(\epsilon^{1.5}),$$

and similarly for U_2'. Thus, we have

$$\begin{aligned}\left\|U - \left[\widetilde{V}_1, \widetilde{W}_1\right]_{gp}\left[\widetilde{V}_2, \widetilde{W}_2\right]_{gp}\right\| &\leq\ \|U - U'\| + \|U' - U_1'U_2'\| + \left\|U_1'U_2' - \left[\widetilde{V}_1, \widetilde{W}_1\right]_{gp}\left[\widetilde{V}_2, \widetilde{W}_2\right]_{gp}\right\| \\ &\leq\ O(\epsilon^{1.5}) + O(\epsilon^2) + O(\epsilon^{1.5}) \\ &\leq\ O(\epsilon^{1.5}),\end{aligned}$$

which is what we wanted to show.

5 Time-Space Lower Bound

In this section we develop one application of our time-space efficient simulation of quantum computations by unbounded-error randomized computations, namely time-space lower bounds for quantum algorithms solving problems closely related to satisfiability. We provide background on this research area in Section 5.1. In Section 5.2 we derive Theorem 3 and mention some extensions. We present some directions for further research in Section 5.3.

5.1 Background

Satisfiability, the problem of deciding whether a given Boolean formula has at least one satisfying assignment, has tremendous practical and theoretical importance. It emerged as a central problem in complexity theory with the advent of NP-completeness in the 1970's. Proving lower bounds on the complexity of satisfiability remains a major open problem. Complexity theorists conjecture that satisfiability requires exponential time and linear space to solve in the worst case. Despite decades of effort, the best single-resource lower bounds for satisfiability on general-purpose models of computation are still the trivial ones — linear for time and logarithmic for space. However, since the late 1990's we have seen a number of results that rule out certain nontrivial combinations of time and space complexity.

One line of research [17, 18, 40, 14, 41], initiated by Fortnow, focuses on proving stronger and stronger time lower bounds for deterministic algorithms that solve satisfiability in small space. For subpolynomial (i.e., $n^{o(1)}$) space bounds, the current record [41] states that no such algorithm can run in time $O(n^c)$ for any $c < 2\cos(\pi/7) \approx 1.8019$.

A second research direction aims to strengthen the lower bounds by considering more powerful models of computation than the standard deterministic one. Diehl and Van Melkebeek [14] initiated the study of lower bounds for problems related to satisfiability on randomized models with bounded error. They showed that for every integer $\ell \geq 2$, Σ_ℓSAT cannot be solved in time $O(n^c)$ by subpolynomial-space randomized algorithms with bounded two-sided error for any $c < \ell$, where Σ_ℓSAT denotes the problem of deciding the validity of a given fully quantified Boolean formula with ℓ alternating blocks of quantifiers beginning with an existential quantifier. Σ_ℓSAT represents the analogue of satisfiability for the ℓth level of the polynomial-time hierarchy; Σ_1SAT corresponds to satisfiability. Proving nontrivial time-space lower bounds for satisfiability on randomized algorithms with bounded two-sided error remains open.

Allender et al. [2] considered the even more powerful (but physically unrealistic) model of randomized algorithms with unbounded error. They settled for problems that are even harder than Σ_ℓSAT for any fixed ℓ, namely MajSAT and MajMajSAT, the equivalents of satisfiability and Σ_2SAT in the counting hierarchy. MajSAT is the problem of deciding whether a given Boolean formula is satisfied for at least half of the assignments to its variables. MajMajSAT is the problem of deciding whether a given Boolean formula φ on disjoint variable sets y and z has the property that for at least half of the assignments to y, φ is satisfied for at least half of the assignments to z. Toda [36] proved that the polynomial-time hierarchy reduces to the class PP, which represents polynomial-time randomized computations with unbounded two-sided error and forms the first level of the counting hierarchy. Apart from dealing with harder problems, the quantitative strength of the time bounds in the Allender et al. lower bounds is also somewhat weaker. They showed that no randomized algorithm can solve MajMajSAT in time $O(n^{1+o(1)})$ and space $O(n^{1-\delta})$ for any positive constant δ.

We refer to [37] for a detailed survey of the past work on time-space lower bounds for satisfiability and related problems, including a presentation of the Allender et al. lower bound that is somewhat different from the original one.

5.2 Results

This paper studies the most powerful model that is considered physically realistic, namely quantum algorithms with bounded error. We obtain the first nontrivial time-space lower bound for quantum algorithms solving problems related to satisfiability. In the bounded two-sided error randomized setting, the reason we can get lower bounds for Σ_ℓSAT for $\ell \geq 2$ but not for $\ell = 1$ relates to the fact that we know efficient simulations of such randomized computations in the second level of the polynomial-time hierarchy but not in the first level. In the quantum setting we know of no efficient simulations in *any* level of the polynomial-time hierarchy. As an application of our simulation by unbounded-error randomized algorithms, we bring the lower bounds of Allender et al. to bear on quantum algorithms. We show that either a time lower bound holds for quantum algorithms solving MajMajSAT or a time-space lower bound holds for MajSAT (Theorem 3). In particular, we get a single time-space lower bound for MajMajSAT (Corollary 1). We use the following general lower bound on unbounded-error randomized algorithms with random access, which is implicit in [2].

Theorem 9 (Allender et al. [2]). *For every real d and every positive real δ there exists a real $c > 1$ such that either*

- MajMajSAT *does not have an unbounded-error randomized algorithm running in time $O(n^c)$, or*
- MajSAT *does not have an unbounded-error randomized algorithm running in time $O(n^d)$ and space $O(n^{1-\delta})$.*

Proof of Theorem 3. This follows immediately from Theorem 1 and Theorem 9 if we absorb the time and space overheads of Theorem 1 into the relationship among the parameters c, d, and δ. □

Proof of Corollary 1. This follows immediately from Theorem 3 because MajSAT trivially reduces to MajMajSAT, and a quantum algorithm running in time $O(n^{1+o(1)})$ and space $O(n^{1-\delta})$ trivially runs in time $O(n^c)$ for every $c > 1$. □

By exploiting the full power of Theorem 4 and Theorem 5, we can weaken the conditions on the error and the complexity of the transition amplitudes in Theorem 3. For example, combining Theorem 4 with Theorem 9 yields a stronger version of Theorem 3 that holds for quantum algorithms M with error $\epsilon \leq 1/2 - 1/n^{O(1)}$ and such that each number in $\mathcal{A}(M)$ has a $\left(2^{o(p)}, 2^{o(p)}\right)$-approximator.

5.3 Future Directions

Several questions remain open regarding time-space lower bounds on quantum models of computation. An obvious goal is to obtain a quantitative improvement to our lower bound. It would be nice to get a particular constant $c > 1$ such that MajMajSAT cannot be solved by quantum algorithms running in $O(n^c)$ time and subpolynomial space. The lower bound of Allender et al. does yield this; however, the constant c is very close to 1, and determining it would require a complicated analysis involving constant-depth threshold circuitry for iterated multiplication [23]. Perhaps there is a way to remove the need for this circuitry in the quantum setting.

A major goal is to prove quantum time-space lower bounds for problems that are simpler than MajMajSAT. Ideally we would like lower bounds for satisfiability itself, although lower bounds for its cousins in PH or $\oplus$P would also be very interesting. The difficulty in obtaining such lower bounds arises from the fact that we know of no simulations of quantum computations in these classes. The known time-space lower bounds for satisfiability and related problems follow the indirect diagonalization paradigm, which involves assuming the lower bound does not hold and then deriving a contradiction with a direct diagonalization result. For example, applying this paradigm to quantum algorithms solving Σ_ℓSAT would entail assuming that Σ_ℓSAT has an efficient quantum algorithm. Since Σ_ℓSAT is complete for the class Σ_ℓP under very efficient reductions, this hypothesis gives a general simulation of the latter class on quantum algorithms. To reach a contradiction with a direct diagonalization result, we seem to need a way to convert these quantum computations back into polynomial-time hierarchy computations.

Obtaining a single time-space lower bound for MajSAT instead of MajMajSAT may be within reach. Recall that Theorem 9 only needs two types of inclusions to derive a contradiction. Under the hypothesis that MajSAT has a bounded-error quantum algorithm running in time $O(n^{1+o(1)})$ and space $O(n^{1-\delta})$, Theorem 1 yields the second inclusion but not the first. One can use the hypothesis to replace the second majority quantifier of a MajMajSAT formula with a quantum computation. However, we do not know how to use the hypothesis again to remove the first majority quantifier, because the hypothesis only applies to majority-quantified *deterministic* computations. Fortnow and Rogers [19] prove that $\mathrm{PP}^{\mathrm{BQP}} = \mathrm{PP}$, and their proof shows how to absorb the "quantumness" into the majority quantifier so that we *can* apply the hypothesis again. However, their proof critically uses time-expensive amplification and is not efficient enough to yield a lower bound for MajSAT via Theorem 9. It might be possible to exploit the space bound to obtain a more efficient inclusion. It might also be possible to exploit more special properties of the construction in [2] to circumvent the need for the amplification component.

Acknowledgments

We thank Stefan Arnold, Sue Coppersmith, Scott Diehl, John Watrous, and anonymous reviewers for helpful comments and pointers.

References

[1] L. Adleman, J. DeMarrais, and M.-D. Huang. Quantum computability. *SIAM Journal on Computing*, 26(5):1524–1540, 1997.

[2] E. Allender, M. Koucký, D. Ronneburger, S. Roy, and V. Vinay. Time-space tradeoffs in the counting hierarchy. In *Proceedings of the 16th IEEE Conference on Computational Complexity*, pages 295–302, 2001.

[3] E. Allender and M. Ogihara. Relationships among PL, #L, and the determinant. *RAIRO – Theoretical Informatics and Applications*, 30:1–21, 1996.

[4] E. Allender and K. Wagner. Counting hierarchies: Polynomial time and constant depth circuits. In G. Rozenberg and A. Salomaa, editors, *Current Trends in Theoretical Computer Science*, pages 469–483. World Scientific, 1993.

[5] S. Arnold. Personal communication, November 2010.

[6] A. Barenco. A universal two-bit gate for quantum computation. *Proceedings of the Royal Society A*, 449:679–683, 1995.

[7] A. Barenco, C. Bennett, R. Cleve, D. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52:3457–3467, 1995.

[8] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997.

[9] P. O. Boykin, T. Mor, M. Pulver, V. Roychowdhury, and F. Vatan. A new universal and fault-tolerant quantum basis. *Information Processing Letters*, 75(3):101–107, 2000.

[10] J. Chiaverini, J. Britton, D. Leibfried, E. Knill, M. Barrett, R. Blakestad, W. Itano, J. Jost, C. Langer, R. Ozeri, T. Schaetz, and D. Wineland. Implementation of the semiclassical quantum Fourier transform in a scalable system. *Science*, 308(5724):997–1000, 2005.

[11] C. Dawson and M. Nielsen. The Solovay-Kitaev Algorithm. *Quantum Information and Computation*, 6(1):81–95, 2006.

[12] D. Deutsch. Quantum computational networks. *Proceedings of the Royal Society A*, 425:73–90, 1989.

[13] D. Deutsch, A. Barenco, and A. Ekert. Universality in quantum computation. *Proceedings of the Royal Society A*, 449:669–677, 1995.

[14] S. Diehl and D. van Melkebeek. Time-space lower bounds for the polynomial-time hierarchy on randomized machines. *SIAM Journal on Computing*, 36(3):563–594, 2006.

[15] D. DiVincenzo. Two-bit gates are universal for quantum computation. *Physical Review A*, 51:1015–1022, 1995.

[16] D. DiVincenzo. The physical implementation of quantum computation. *Fortschritte der Physik*, 48:771–784, 2000.

[17] L. Fortnow. Time-space tradeoffs for satisfiability. *Journal of Computer and System Sciences*, 60(2):337–353, 2000.

[18] L. Fortnow, R. Lipton, D. van Melkebeek, and A. Viglas. Time-space lower bounds for satisfiability. *Journal of the ACM*, 52(6):835–865, 2005.

[19] L. Fortnow and J. Rogers. Complexity limitations on quantum computation. *Journal of Computer and System Sciences*, 59(2):240–252, 1999.

[20] G. Golub and C. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, third edition, 1996.

[21] L. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th ACM Symposium on Theory of Computing*, pages 212–219, 1996.

[22] A. Harrow, B. Recht, and I. Chuang. Efficient discrete approximations of quantum gates. *Journal of Mathematical Physics*, 43(9):4445–4451, 2002.

[23] W. Hesse. Division is in uniform TC^0. In *Proceedings of the 28th International Colloquium On Automata, Languages, and Programming*, pages 104–114, 2001.

[24] R. Horn and C. Johnson. *Topics in Matrix Analysis.* Cambridge University Press, 1994.

[25] A. Kitaev. Quantum computations: Algorithms and error correction. *Russian Mathematical Surveys*, 52(6):1191–1249, 1997.

[26] A. Kitaev, A. Shen, and M. Vyalyi. *Classical and Quantum Computation.* American Mathematical Society, 2002.

[27] S. Lloyd. Almost any quantum logic gate is universal. *Physical Review Letters*, 75:346–349, 1995.

[28] M. Mariantoni, H. Wang, T. Yamamoto, M. Neeley, R. Bialczak, Y. Chen, M. Lenander, E. Lucero, A. O'Connell, D. Sank, M. Weides, J. Wenner, Y. Yin, J. Zhao, A. Korotkov, A. Cleland, and J. Martinis. Implementing the quantum von Neumann architecture with superconducting circuits. *Science Express*, 2011.

[29] A. Mobasher, S. Fathololoumi, and S. Rahimi. Quantum dot quantum computation. Technical Report 2007-05, University of Waterloo Electrical and Computer Engineering Department, 2007.

[30] A. Nagy. On an implementation of the Solovay-Kitaev Algorithm. *CoRR*, abs/quant-ph/0606077v1, 2006.

[31] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2000.

[32] S. Perdrix and P. Jorrand. Classically controlled quantum computation. *Mathematical Structures in Computer Science*, 16(4):601–620, 2006.

[33] A. Politi, J. Matthews, and J. O'Brien. Shor's quantum factoring algorithm on a photonic chip. *Science*, 325(5945):1221, 2009.

[34] Y. Shi. Both Toffoli and controlled-NOT need little help to do universal quantum computation. *Quantum Information and Computation*, 3(1):84–92, 2003.

[35] P. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[36] S. Toda. PP is as hard as the polynomial time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

[37] D. van Melkebeek. A survey of lower bounds for satisfiability and related problems. *Foundations and Trends in Theoretical Computer Science*, 2:197–303, 2007.

[38] J. Watrous. Space-bounded quantum complexity. *Journal of Computer and System Sciences*, 59(2):281–326, 1999.

[39] J. Watrous. On the complexity of simulating space-bounded quantum computations. *Computational Complexity*, 12(1-2):48–84, 2003.

[40] R. Williams. Inductive time-space lower bounds for SAT and related problems. *Computational Complexity*, 15(4):433–470, 2006.

[41] R. Williams. Time-space tradeoffs for counting NP solutions modulo integers. *Computational Complexity*, 17(2):179–219, 2008.

[42] H. Wiseman and G. Milburn. *Quantum Measurement and Control.* Cambridge University Press, 2009.

[43] A. Yakaryılmaz and A. C. C. Say. Unbounded-error quantum computation with small space bounds. *Information and Computation*, 209(6):873–892, 2011.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de