Quantum Information & Computation
ISSN 1533-7146 Volume 25 (2025) pp. 36–56

Cerebration Science Publishing

Article

Unconditional Proofs of Quantumness Between Small-Space Machines

A. C. Cem Say * and M. Utkan Gezer

Department of Computer Engineering, Boğaziçi University, Bebek, İstanbul 34342, Türkiye; utkan.gezer@bogazici.edu.tr
* Correspondence author: say@bogazici.edu.tr

Received date: 24 November 2024; Accepted date: 21 January 2025; Published online: 15 February 2025

Abstract: A proof of quantumness is a protocol through which a classical machine can test whether a purportedly quantum device, with comparable time and memory resources, is performing a computation that is impossible for classical computers. Existing approaches to provide proofs of quantumness depend on unproven assumptions about some task being impossible for machines of a particular model under certain resource restrictions. We study a setup where both devices have space bounds $o(\log \log n)$. Under such memory budgets, it has been unconditionally proven that probabilistic Turing machines are unable to solve certain computational problems. We formulate a new class of problems and show that these problems are polynomial-time solvable for quantum machines, impossible for classical machines, and have the property that their solutions can be "proved" by a small-space quantum machine to a classical machine with the same space bound. These problems form the basis of our newly defined protocol, where the polynomial-time verifier's verdict about the tested machine's quantumness is not conditional on an unproven weakness assumption.

Keywords: proofs of quantumness; interactive proof systems; probabilistic Turing machines; quantum finite automata

1. Introduction

It is hoped that sufficiently advanced quantum computers will be able to solve problems that are impossible for classical computers employing comparable amounts of resources. This brings up interesting questions about how such a claim of "quantum advantage" can be verified by a classical machine: How can a device check the correctness of a claimed solution for a problem that is too difficult for it to solve on its own? Furthermore, even if the "weak" device can verify that the problem has been solved correctly, is this sufficient to conclude that the "strong" device is indeed a quantum computer, rather than a classical one implementing a fast algorithm unknown to the verifying party?

Brakerski et al. [1] define a *proof of quantumness* as "a method for provably demonstrating (to a classical verifier) that a quantum device can perform computational tasks that a classical device with comparable resources cannot." All approaches to providing such protocols listed in [1] depend on unproven assumptions about a particular task being difficult for computational machines of a particular model. For instance, in one such approach, a "strong" computer can factorize a given composite number, the verifier can quickly check that the factors are correct, and then "conclude" that it is faced with a computer that has run an efficient quantum algorithm like [2], based on the belief that polynomial-time factorization is impossible for classical machines. As another example, Mahadev's protocol [3] for a classical computer to interactively verify the result of an efficient quantum computation is based on the assumption that the Learning with Errors problem [4] is intractable for polynomial-time *quantum* machines, and, of course, can provide a framework for proofs of quantumness only if the conjecture $\mathsf{BPP} \subsetneq \mathsf{BQP}$ is true.

More recently, Zhan [5] and Girish et al. [6] have focused on proofs between logarithmic-space bounded machines. They have shown that, for any language in the class BQL, such a quantum machine can prove its work (on deciding about the membership of the common input string in the language) to a classical logspace verifier by transmitting a "proof string" of polynomial length. Considered as a framework for proofs of quantumness, this approach is also based on the (as yet unproven) assumption that BQL is a proper superset of BPL, the corresponding class for classical logspace machines.

DOI: 10.2478/qic-2025-0002

In this paper, we present the first framework for proofs of quantumness where the conclusion of the verifier is not conditional on an unproven presupposition about a weakness of a family of machines, by studying the case where both interacting computers have drastically small ($o(\log\log n)$) memory budgets. It is known [7,8] that certain separation problems are impossible for probabilistic Turing machines (PTM) using $o(\log\log n)$ space, even if they are allowed to employ uncomputable numbers as their transition probabilities. On the other hand, two-way finite automata with quantum and classical states (2QCFA's), which are, essentially, constant-space Turing machines which have been augmented with a fixed-size quantum register, are known [9–12] to be able to perform some language recognition tasks that are beyond the capabilities of classical small-space machines. We combine these facts with techniques stemming from the study of interactive proof systems with finite-state verifiers [7,13–16] and quantum finite automata [17,18] to construct setups where a polynomial-time interaction between the two space-bounded automata ends with the verifier accepting with very high probability only if it is faced with a genuinely quantum machine obeying the communication protocol. Note that the memory sizes of presently realizable quantum computers also justify the consideration of extremely small space bounds in this context.

The paper is structured as follows: Section 2 presents the building blocks to be used to construct our framework. The small-space Turing machine models that represent the classical and quantum computers which take part in the tests of quantumness are defined. We allow our classical machines to use the largest realistic set of transition probabilities, namely, efficiently computable reals between 0 and 1, and naturally impose the same computability restriction on the transition amplitudes of the quantum machines.[1] A list of tasks that are known or presumed to be difficult for small-space PTMs, but are solvable by 2QCFA algorithms, is given in Section 2.1. Section 2.2 describes how we pair a classical verifier with a (quantum or classical) "prover" in our protocols and presents a method by which a constant-space prover can convince a similarly bounded verifier about the membership of their common input string in any language recognized by two-way deterministic two-head finite automata possessing a property that we define. To our knowledge, this is the first step in the characterization of the power of interactive proof systems with such severely constrained provers.

We give a formal definition of what we mean by "checking a proof of quantumness" in Section 3. Intuitively, one describes a verifier V which accepts with high probability if it is faced with a genuinely quantum prover P^Q that can convince V that it is able to solve any given instance of a computational problem **P**, and every classical prover P^C that attempts to convince V is guaranteed to be less successful than P^Q on an infinite set of common input strings W_{P^C}. Constructing such a verifier therefore requires identifying a problem **P** that has the following properties:

1. **P** should be solvable for a polynomial-time QTM.
2. **P** should be provably impossible to solve for a PTM within the same time and space bounds.
3. The solution of each instance of **P** should be "provable" by a QTM to a PTM, both operating within the same time and space bounds, through a polynomial-time interaction where only classical messages are exchanged.

As noted above, we are able to identify problems that satisfy all these properties where the common space bound is very small. In Section 4, we present a specific family of such problems and describe a template for proof-of-quantumness protocols for each member of that family.

In Section 5, we give another protocol based on a problem that seems to be beyond the capability of the provers described in Section 4. That verifier, which employs a version of Freivalds' celebrated (exponential-time) probabilistic finite-state algorithm [20] for recognizing a nonregular language, can be considered to be checking a proof of quantumness on the condition that the underlying problem is classically difficult. Section 6 lists several remaining open questions. Appendix A presents a generalization of a theorem by Dwork and Stockmeyer [21] that may provide a starting point for future proof-of-quantumness protocols.

2. Building Blocks

We start by establishing some key strengths and weaknesses of the machine models that underlie the "proof of quantumness" setup to be studied. Section 2.1 presents the "stand-alone" versions of the classical (probabilistic) and quantum machines that will be relevant in our discussion, and lists the quantum advantage results under the

[1] The 2QCFA algorithms we will use are known [9] to rely on the highly precise manipulation of a small number of qubits. This requirement is mitigated by our imposition of polynomial limits on their runtimes, noting that a polynomial-time quantum computation can be carried out within constant accuracy if the transition amplitudes are specified to logarithmically many bits of precision [19]. Our definitions allow the descriptions of classical machines that take part in our protocols to specify transition probabilities with the same precision as the quantum ones.

considered common space bounds. In Section 2.2, we describe how those stand-alone machines can be put together to form interactive proof systems and present a technique that will be used by the provers to be presented in Section 4.

In the following, we assume that the reader is familiar with the fundamental notions of quantum computation and the theory of computational complexity [22]. Let $\mathcal{C}_{[0,1]}$ denote the set of efficiently computable numbers (i.e., those which are associated with deterministic Turing machines that can compute them to b-bit precision within time polynomial in b) in $[0,1]$.

2.1. Basic Models and Quantum Advantage

The two basic computational models considered in this paper are probabilistic and quantum Turing machines. In both cases, the machine has a read-only input tape. For any input string w, this tape will contain the string $\triangleright w \triangleleft$, where $\triangleright$ and $\triangleleft$ are special endmarker symbols. Details about the work tapes differ among models, as will be seen.

Definition 1. *A* probabilistic Turing machine (PTM)[2] *is a 7-tuple* $\left(S_c, \Sigma, K, \delta, c_0, c_{\text{acc}}, c_{\text{rej}}\right)$*, where*

- S_c *is the finite set of states,*
- Σ *is the finite input alphabet, not containing* $\triangleright$ *and* $\triangleleft$*,*
- K *is the finite work alphabet, including the blank symbol* $\sqcup$*,*
- $\delta : \left(S_c \setminus \{ c_{\text{acc}}, c_{\text{rej}} \}\right) \times (\Sigma \cup \{ \triangleright, \triangleleft \}) \times K \times S_c \times K \times \{ -1, 0, 1 \} \times \{ -1, 0, 1 \} \to \mathcal{C}_{[0,1]}$*, such that, for each fixed* c*,* σ*, and* κ*,* $\sum_{c', \kappa', d_i, d_w} \delta(c, \sigma, \kappa, c', \kappa', d_i, d_w) = 1$*, is the transition function,*
- $c_0 \in S_c$ *is the start state, and*
- $c_{\text{acc}}, c_{\text{rej}} \in S_c$*, such that* $c_{\text{acc}} \neq c_{\text{rej}}$*, are the accept and reject states, respectively.*

A PTM has a single read-write work tape, which is initially blank, with the work head positioned on the leftmost cell. The computation of a PTM M on input $w \in \Sigma^*$ begins with the input head positioned on the first symbol of the string $\triangleright w \triangleleft$, and the machine in state c_0. In every computational step, if M is in some state $c \notin \{ c_{\text{acc}}, c_{\text{rej}} \}$ with the input head scanning some symbol σ at the ith tape position and the work head scanning some symbol κ at the jth tape position, it switches to state c', sets the scanned work symbol to κ', and locates the input and work heads at the $(i + d_i)$th and $(j + d_w)$th positions respectively with probability $\delta(c, \sigma, \kappa, c', \kappa', d_i, d_w)$. We assume that δ is defined such that the input head never moves beyond the tape area delimited by the endmarkers. If M is in state c_{acc} (c_{rej}), it halts and accepts (rejects).

A PTM is said to *run within space* $s(n)$ if, on any input of length n, at most $s(n)$ work tape cells are scanned throughout the computation. We are going to focus on machines that run within space bounds that are $o(\log \log n)$. A computation performed by a machine that runs within $O(1)$ space can also be performed by a machine which does not use its work tape at all and encodes all the workspace in its state set. The definition of this constant-space model, known as the *two-way probabilistic finite automaton (2PFA)*, is easily obtained from Definition 1 by removing the components related to the work tape. A *two-way deterministic finite automaton (2DFA)* is simply a 2PFA whose transition function is restricted to the range $\{ 0, 1 \}$.

Our protocols in Sections 4 and 5 will make use of the well-known capability of 2PFA's to implement "alarm clocks" that can be set to any desired polynomial time bound. A detailed proof of the following fact can be found, for example, in [16]:

Fact 1. *For any pair of positive integers* $k_{\text{p}}, t > 0$ *and desired "error" bound* $\varepsilon_{\text{p}} > 0$*, there exists a 2PFA with expected runtime in* $O\left(n^{t+1}\right)$*, such that the probability that this machine halts in fewer than* $k_{\text{p}} n^t$ *steps is* ε_{p}*.*

For our *quantum Turing machine (QTM)* model, we adopt the definition of Watrous [23]. This model is best visualized as a deterministic Turing machine (with a classical work tape) which has been augmented by adding a finite quantum register and a quantum work tape, each cell of which holds a qubit. The randomness in the behavior of a QTM is a result of the operations performed on the quantum part of the machine at intermediate steps. A QTM is said to run within space $s(n)$ if, on any input of length n, at most $s(n)$ cells are scanned on both the classical and quantum work tapes during the computation.

[2] This definition has been obtained by restricting the transition probabilities of the model in [21] to efficiently computable numbers.

It turns out that all the QTMs that we need in our results in this paper are constant-space machines. As described above, machines obeying this restriction can be modeled in a simpler way by removing the work tapes from the definition. Doing this to the QTMs of [23] yields the well-studied quantum finite automaton model due to Ambainis and Watrous [9], which we describe in detail below.[3] The following definition, which can be viewed as a 2DFA augmented with a quantum register of constant size, is somewhat more involved than Definition 1, because of the way it breaks down each computational step into two stages for evolving the quantum state and the remaining classical part of the machine separately:

Definition 2. *A* two-way finite automaton with quantum and classical states (2QCFA) *is a 9-tuple* $\left(S_q, S_c, \Sigma, \delta_q, \delta_c, q_0, c_0, c_{\text{acc}}, c_{\text{rej}}\right)$*, where*

- S_q *is the finite set of basis states of the quantum part of the machine,*
- S_c *is the finite set of classical states,*
- Σ *is the finite input alphabet, not containing* $\rhd$ *and* $\lhd$*,*
- δ_q *is the quantum transition function, which, for any* $c \in (S_c \setminus \{ c_{\text{acc}}, c_{\text{rej}} \})$ *and* $\sigma \in (\Sigma \cup \{ \rhd, \lhd \})$*, maps the pair* (c, σ) *to an action that will be performed on the quantum part of the machine, as will be described below,*
- δ_c *is the classical transition function, which determines the input head movement direction and resulting classical state associated with the current computational step, as will be described below,*
- $q_0 \in S_q$ *is the initial quantum state,*
- $c_0 \in S_c$ *is the classical start state, and*
- $c_{\text{acc}}, c_{\text{rej}} \in S_c$*, such that* $c_{\text{acc}} \neq c_{\text{rej}}$*, are the accept and reject states, respectively.*

Just like in the classical case, the input string to a 2QCFA is delimited by endmarkers (beyond which the machine does not attempt to move its head) on the read-only tape. At the start of the computation, the input head is located on the left endmarker, the state of the quantum register is described by the vector $|q_0\rangle$, and the classical state is c_0. Every computational step begins with the action $\delta_q(c, \sigma)$ determined by the current classical state c and the currently scanned tape symbol σ being performed on the quantum register. Each such action is either a unitary transformation or a projective measurement of the quantum state, resulting in some outcome τ with an associated probability. The second and final stage of the computational step is a classical transition that determines the pair (c', d) of the resulting classical state and head movement direction of this step: If $\delta_q(c, \sigma)$ was a unitary transformation, this pair is simply $\delta_c(c, \sigma)$, depending again only on c and σ. If, on the other hand, $\delta_q(c, \sigma)$ was a measurement with outcome τ, the pair (c', d) is $\delta_c(c, \sigma, \tau)$, which also depends on that outcome. Computation halts with acceptance (rejection) when the 2QCFA reaches the classical state c_{acc} (c_{rej}).

Since we want our classical and quantum machine models to be "comparable" in all respects except the fact that one is classical and the other is quantum, we restrict all entries of all matrices describing the transitions of 2QCFAs to be complex numbers whose real and imaginary parts are efficiently computable.[4]

Let $\mathtt{A}, \mathtt{B} \subseteq \Sigma^*$ with $\mathtt{A} \cap \mathtt{B} = \emptyset$. We say that a (classical or quantum) machine M *separates* $\mathtt{A}$ *and* $\mathtt{B}$ *(with error bound* ε*)* if there exists a number $\varepsilon < \frac{1}{2}$ such that

- for all $w \in \mathtt{A}$, M accepts input w with probability at least $1 - \varepsilon$, and
- for all $w \in \mathtt{B}$, M rejects input w with probability at least $1 - \varepsilon$.

Equivalently, one says that M *solves the promise problem* $(\mathtt{A}, \mathtt{B})$. We say that M *recognizes* $\mathtt{A}$ if M separates $\mathtt{A}$ and the complement of $\mathtt{A}$.

For any given 2PFA M^C recognizing some language $\mathtt{L}$, it is trivial to construct a 2QCFA that recognizes $\mathtt{L}$ by mimicking M^C step by step. 2QCFAs are, in fact, strictly more powerful computational machines than 2PFAs. We review the relevant facts below.

Although some nonregular languages like $\mathtt{EQ}_{\mathtt{ab}} = \{ a^n b^n \mid n \geq 0 \}$ are known [20] to have exponential-time 2PFAs that recognize them, no $o(\log \log n)$-space PTM can recognize a nonregular language in polynomial expected

[3] Remscrim [24] provides a detailed explanation of how quantum finite automata are generalized to QTMs with larger space bounds.

[4] The unrealistic case where uncomputable numbers are allowed in the descriptions of these machines has been studied in [10,21]. Remscrim [11] has investigated the effects of imposing more restrictions on the set of allowed matrix entries.

time, as proven by Dwork and Stockmeyer [21]. On the other hand, Ambainis and Watrous [9] describe a polynomial-time 2QCFA that recognizes $\texttt{EQ}_{\texttt{ab}}$. Naturally, polynomial-time 2QCFAs also exist for infinitely many other languages whose recognition is reducible to determining whether two different symbols appear equally many times in the input string, e.g., $\texttt{EQ} = \{ w \mid w \in \{ a, b \}^* \text{ and } w \text{ contains equally many } a\text{'s and } b\text{'s} \}$ [12] and

$$\texttt{POWER-EQ} = \left\{ aba^7ba^{7\cdot 8}ba^{7\cdot 8^2} \cdots ba^{7\cdot 8^n} \;\middle|\; n \geq 0 \right\},$$

studied in [10]. More recently, Remscrim [11] has shown that any language corresponding to the word problem of a finitely generated virtually abelian group can be recognized in cubic time by some bounded-error 2QCFA. EQ is one of the "simplest" members of this set of languages. Let $\mathsf{BQTISP}_*(t(n), s(n))$ denote the class of languages recognizable by QTMs that run in at most $t(n)$ expected time and within at most $s(n)$ space on all inputs of length at most n, so that such a QTM recognizing the corresponding language exists for any desired positive error bound. All languages mentioned in this paragraph are known to be in $\mathsf{BQTISP}_*(n^k, 1)$ for some constant k depending on the specific language.

In their seminal paper, Ambainis and Watrous [9] also gave an exponential-time 2QCFA algorithm for PAL, the language of palindromes on a binary alphabet. This provided another early example of quantum superiority over classical machines, since it is known [7,8] that no sublogarithmic-space PTM can recognize PAL in any amount of time.

A *rational 1PFA* [25] is a 2PFA whose transition function is restricted to contain only rational numbers in its range, and not to assign positive probabilities to transitions that do not move the head to the right. A language L is said to belong to the class $\mathsf{S}_{\mathsf{Q}}^{=}$ [25,26] if there exists a rational 1PFA that accepts all and only the members of L with probability $\frac{1}{2}$. $\mathsf{S}_{\mathsf{Q}}^{\neq}$ is the class of languages whose complements are in $\mathsf{S}_{\mathsf{Q}}^{=}$.

Using a technique developed in [17], Yakaryılmaz and Say have proven [18] that all languages in $\mathsf{S}_{\mathsf{Q}}^{=} \cup \mathsf{S}_{\mathsf{Q}}^{\neq}$ can be recognized by 2QCFAs [12]. Combining this result with Remscrim's work on word problems [11], Table 1 lists some other interesting languages known to have 2QCFAs that recognize them in $2^{O(n)}$ expected time for inputs of length n.[5] For each of those languages, and for any desired small positive value of ε, there exists a 2^{kn}-time 2QCFA (with a correspondingly large k) recognizing the language with error bound ε.

Table 1. Some languages recognized in $2^{O(n)}$ time by 2QCFAs.

Language	Description
PAL	$\{ w \mid w = w^R \}$
TWIN	$\{ w\#w \mid w \in \{ a, b \}^* \}$
MULT	$\{ x\#y\#z \mid x, y, z$ are natural numbers in binary notation and $x \cdot y = z \}$
SQUARE	$\left\{ a^i b^{i^2} \;\middle|\; i > 0 \right\}$
POWER	$\left\{ a^i b^{2^i} \;\middle|\; i > 0 \right\}$
any "polynomial language"	A *polynomial language* [27] is defined as $\left\{ a_1^{n_1} \cdots a_k^{n_k} b_1^{p_1(n_1,\ldots,n_k)} \cdots b_r^{p_r(n_1,\ldots,n_k)} \;\middle|\; p_i(n_1, \ldots, n_k) \geq 0 \right\}$, where $a_1, \ldots, a_k, b_1, \ldots, b_r$ are distinct symbols, and each p_i is a polynomial with integer coefficients.
$\texttt{W}_{\texttt{G}}$	the word problem for G, where G is any finitely generated virtually free group

2.2. Interactive Proof Systems with Small-Space Provers

An *interactive proof system* consists of two entities called the *verifier* and the *prover* that share a common finite input alphabet Σ and a finite communication alphabet Γ. The verifier definition in our setup will essentially be a version of Definition 1 that has been augmented to enable the machine to communicate with the prover.

Definition 3. *A* verifier *is a 9-tuple* $\left(S_{\text{silent}}, S_{\text{com}}, \Sigma, K, \Gamma, \delta, c_0, c_{\text{acc}}, c_{\text{rej}}\right)$*, where*

- S_{silent} *is the finite set of noncommunicating states,*
- S_{com} *is the finite set of communicating states, such that* $S_{\text{silent}} \cap S_{\text{com}} = \emptyset$*,*

[5] w^R denotes the reverse of string w.

- Σ *is the finite input alphabet, not containing* $\triangleright$ *and* $\triangleleft$*,*
- K *is the finite work alphabet, including the blank symbol* $\sqcup$*,*
- Γ *is the finite communication alphabet,*
- δ *is the transition function, described below,*
- $c_0 \in S_{\text{silent}}$ *is the start state, and*
- $c_{\text{acc}}, c_{\text{rej}} \in S_{\text{silent}}$*, such that* $c_{\text{acc}} \neq c_{\text{rej}}$*, are the accept and reject states, respectively.*

The main difference between this verifier model and the "stand-alone" PTM model of Definition 1 is the communication cell, through which the messages to and from the prover will be conveyed. When a verifier V starts execution, the communication cell contains the blank symbol $\sqcup$. The transition function δ is defined in two parts. In any computational step starting with V at some state $c \in S_{\text{silent}}$ with its input head scanning some symbol σ at the ith tape position, its work head scanning some symbol κ at the jth tape position, and the communication cell containing the symbol γ, $\delta(c, \sigma, \kappa, \gamma, c', \kappa', d_i, d_w)$ is the probability with which V will switch to state c', write κ' in the scanned work tape cell, and then locate the input and work heads at the $(i + d_i)$th and $(j + d_w)$th tape positions, respectively.

Each communication state c is associated with a symbol $\gamma_s \in \Gamma$. Whenever the machine enters a state $c \in S_{\text{com}}$, the symbol γ_s is written in the communication cell. As will be described below, the prover is supposed to respond to this communication with some symbol γ' in its next computational step. If this response is not received by the time V starts its next step, it rejects immediately. Otherwise, V transitions from state c to state c', replaces the scanned work tape symbol with κ', and moves its heads in the directions associated with d_i and d_w, respectively, with probability $\delta(c, \sigma, \kappa, \gamma', c', \kappa', d_i, d_w)$, having received the communication symbol γ' in response to its message while scanning the input symbol σ and the work tape symbol κ.

Note that both the runtime and the acceptance probability of a verifier reading a particular input string and communicating with a particular prover may depend on the messages of the prover provided as responses during this communication.

In the study of interactive proof systems where computationally limited verifiers are faced with provers of unbounded power, one has no need to commit to a specific machine definition for the role of the prover. In the context of this paper, the prover is known to be a $o(\log \log n)$-space machine, and it is the verifier's task to determine whether the prover is indeed quantum (i.e., a QTM augmented with a communication cell) as it claims to be, or just a classical probabilistic Turing machine as the verifier itself. In the following definitions of these two types of provers, we obey the convention [7] that the prover sends a symbol through the communication cell only at the steps in which it detects that a new communication symbol has just been written by the verifier. As mentioned above, the QTM algorithms we will describe require only $O(1)$ space, so our definition of a quantum prover is just a properly augmented version of Definition 2:

Definition 4. *A* (constant-space) quantum prover *is a 9-tuple* $\left(S_q, S_c, \Sigma, \Gamma, \delta_q, \delta_{\text{silent}}, \delta_{\text{com}}, q_0, c_0\right)$*, where*

- S_q *is the finite set of basis states of the quantum part of the machine,*
- S_c *is the finite set of classical states,*
- Σ *is the finite input alphabet, not containing* $\triangleright$ *and* $\triangleleft$*,*
- Γ *is the finite communication alphabet,*
- δ_q *is the quantum transition function, which, for any* $c \in (S_c \setminus \{ c_{\text{acc}}, c_{\text{rej}} \})$ *and* $\sigma \in (\Sigma \cup \{ \triangleright, \triangleleft \})$*, maps the pair* (c, σ) *to an action (a unitary transition or a measurement) that will be performed on the quantum part of the machine, exactly as in Definition 2,*
- δ_{silent} *is the "silent" classical transition function, which determines the input head movement direction and resulting classical state associated with the current computational step when no new incoming symbol has been detected in the communication cell, as the function* δ_c *in Definition 2,*
- δ_{com} *is the classical transition function to be employed when a new message from the verifier has been detected in the communication cell, as described below,*
- $q_0 \in S_q$ *is the initial quantum state, and*
- $c_0 \in S_c$ *is the classical start state.*

Any such quantum prover P^Q has access to a communication cell that it shares with the verifier. Just like the stand-alone 2QCFAs we saw in Section 2.1, P^Q performs each computational step in two stages. In addition to

the quantum and classical transition functions δ_q and δ_{silent}, which serve the same purpose as their counterparts in Definition 2, P^Q also has a second classical transition function δ_{com}, which is employed instead of δ_{silent} only at the steps where a new communication symbol written by the verifier is detected in the communication cell. Each such step starting from classical state c and with symbol σ being scanned in the input tape shared by the verifier first performs the quantum transition indicated by $\delta_q(c, \sigma)$ as described in Section 2.1, but continues with a classical transition (that also depends on the symbol γ in the communication cell) from δ_{com}, rather than δ_{silent}. In more detail, if $\delta_q(c, \sigma)$ was a unitary transition, $\delta_{\text{com}}(c, \sigma, \gamma)$ is a triple (c', d, γ'), specifying the resulting classical state c' and input head direction d of P^Q, as well as the symbol γ' that P^Q will write in the communication cell, overwriting its previous content. If $\delta_q(c, \sigma)$ dictated a measurement with outcome τ, $\delta_{\text{com}}(c, \sigma, \tau, \gamma)$ is similarly a triple indicating the new state, head direction, and response of P^Q.

We define a classical prover analogously. Since we will aim to prove that the verifiers to be presented in Sections 4 and 5 reject the "quantumness" claims of *any* classical prover whose space bound is (a possibly increasing function of n) within $o(\log \log n)$, we base this model on the general PTM with the work tape (Definition 1).

Definition 5. *A* classical prover *is a 7-tuple* $(S_c, \Sigma, \Gamma, K, \delta_{\text{silent}}, \delta_{\text{com}}, c_0)$*, where*

- S_c *is the finite set of states,*
- Σ *is the finite input alphabet, not containing* $\triangleright$ *and* $\triangleleft$*,*
- K *is the finite work alphabet, containing the blank symbol* $\sqcup$*,*
- Γ *is the finite communication alphabet,*
- $\delta_{\text{silent}} : \left(S_c \setminus \{ c_{\text{acc}}, c_{\text{rej}} \}\right) \times (\Sigma \cup \{ \triangleright, \triangleleft \}) \times K \times S_c \times K \times \{ -1, 0, 1 \} \times \{ -1, 0, 1 \} \to \mathcal{C}_{[0,1]}$*, such that, for each fixed* c*,* σ*, and* κ*,* $\sum_{c', \kappa', d_i, d_w} \delta(c, \sigma, \kappa, c', \kappa', d_i, d_w) = 1$*, is the "silent" transition function, as in Definition 1,*
- $\delta_{\text{com}} : \left(S_c \setminus \{ c_{\text{acc}}, c_{\text{rej}} \}\right) \times (\Sigma \cup \{ \triangleright, \triangleleft \}) \times K \times \Gamma \times S_c \times K \times \Gamma \times \{ -1, 0, 1 \} \times \{ -1, 0, 1 \} \to \mathcal{C}_{[0,1]}$*, such that, for each fixed* c*,* σ*,* κ*, and* γ*,* $\sum_{c', \kappa', \gamma', d_i, d_w} \delta(c, \sigma, \kappa, \gamma, c', \kappa', \gamma', d_i, d_w) = 1$*, is the transition function to be employed at steps where a fresh message from the verifier is detected in the communication cell, as described below, and*
- $c_0 \in S_c$ *is the start state.*

The difference between Definitions 3 and 5, which both describe 2PFAs augmented with a communication cell, is caused by the need to follow the convention that a prover only "talks" in response to a communication from the corresponding verifier. As in Definition 4, we specify two different transition functions, δ_{silent} and δ_{com}: $\delta_{\text{com}}(c, \sigma, \kappa, \gamma, c', \kappa', \gamma', d_i, d_w)$ is the probability with which the classical prover will switch to state c', write κ' on the work tape, write γ' in the communication cell, and move the input and work tape heads in directions d_i and d_w, respectively, at a step where it has detected a fresh communication symbol γ while it is in state c and its input and work heads are scanning the symbols σ and κ, respectively. δ_{silent}, which is identical in function to its counterpart in Definition 1, is employed at steps where no "news" has been received from the verifier.

For any verifier V and prover P running on a common input string w, we let $\Pr_{\text{acc}}(V, P, w)$ (resp. $\Pr_{\text{rej}}(V, P, w)$) denote the probability that V accepts (resp. rejects) and halts as a result of the interaction. We will also be considering the possibility that V is "tricked" into running forever without reaching a halting state.

In a standard interactive proof system (IPS) as in [28], the prover's aim is to convince the verifier to accept with high probability that their common input string is a member of some language $\mathtt{L}$, and we say that $\mathtt{L}$ *has an IPS with error bound* ε if there exists a verifier that succeeds in making use of the communication with the prover to accept every member of $\mathtt{L}$, *and* avoiding being tricked into looping on or accepting any non-member of $\mathtt{L}$ with probability at least $1 - \varepsilon$, for some $\varepsilon < \frac{1}{2}$. In the next section, we will show how to modify this setup so that the end result of the dialogue between the verifier and the prover can be interpreted as a verdict about whether the prover is a quantum machine or not. In the remainder of this section, we present a technique that can be used by a small-space prover faced with a similarly bounded verifier in a standard IPS. This technique will be employed by the quantum provers to be described in Section 4.

A *two-head two-way deterministic finite automaton (2DFA(2))* [29] can be visualized as a 2DFA with not one, but two heads. Both heads are positioned on the left endmarker at the beginning of the computation. The transition function governs the two heads separately. At any step of the computation, the next move (i.e., the next state and the movement directions of the two heads) is determined by the current internal state and the symbol pair scanned by the two heads. The class of languages that can be recognized by 2DFA(2)s, designated by $\mathcal{L}(\text{2DFA(2)})$, is known to

contain many interesting nonregular languages. We will concentrate on a subclass associated with a more restricted machine definition.

In the following, we will construct a verifier algorithm for simulating a given 2DFA(2) M on a given input. At each step of the simulation, the verifier, which has only one input head, will receive the symbol which is purportedly scanned by one of the heads of M from the prover. This will give the prover a chance to "lie" about the reading of that head in order to trick the verifier to accept the input string, even if it is not a member of the language recognized by M. We now define a property of heads that will be important in this regard.

Definition 6. *The ith head (H_i) of a 2DFA(2) M is said to be* supersafe[6] *if and only if there exists a 2DFA N_i such that the following are true for any input string w:*

1. *N_i's head follows a finite trajectory (i.e., a sequence of head position indices) T_w, after which N_i halts, when given the input w.*
2. *If a (single-head) verifier simulates M by using its own head to imitate the movements of H_i on the input w and receiving information about the other head from a prover, the trajectory of the verifier's head is guaranteed to be some prefix of T_w (after which the simulation ends by arriving at a halting state of M), regardless of whether the readings of the other head are provided consistently with the input string or are made up arbitrarily at every step by the prover.*

Intuitively, the transition function of such a 2DFA(2) renders the supersafe head's trajectory insensitive to the readings of the other head.

The runtime of a 2DFA(2) M with a supersafe head must have a linear upper bound, since the 2DFA N_i corresponding to the supersafe head is guaranteed to halt for all inputs, any 2DFA running on an input of length n can attain $O(n)$ different configurations, and Definition 6 guarantees that M's runtime is not longer than that of N_i.

As an example, consider the following 2DFA(2) for EQ: The first head, which is supersafe, walks over the input from the left endmarker to the right, and then back towards the left. The second head moves one step to the right for every a that the first head scans during its left-to-right phase, and attempts to move one step to the left for every b scanned by the first head in its right-to-left phase. The machine rejects if the second head fails to return to the left endmarker or attempts to move beyond it in that last phase. Otherwise, it accepts.

The class of languages that can be recognized by 2DFA(2)s with at least one supersafe head will be denoted by $\mathcal{L}(\text{2DFA}(2_s))$. It is easy to see that many other languages (e.g., $\texttt{EQ}_{\texttt{ab}}$, POWER-EQ, PAL, and TWIN) are also in $\mathcal{L}(\text{2DFA}(2_s))$. We do not know if $\mathcal{L}(\text{2DFA}(2)) = \mathcal{L}(\text{2DFA}(2_s))$.

Theorem 1. *For any $\varepsilon > 0$, every language in $\mathcal{L}(\text{2DFA}(2_s))$ has an IPS with error bound ε comprising of a constant-space (classical or quantum) prover and a constant-space verifier that halts with probability at least $1 - \varepsilon$ within expected polynomial time, and may fail to halt with the remaining probability.*

Proof. Let M be a 2DFA(2) with a supersafe head H_1 and another head H_2, recognizing language L. In relation to M and H_1, let N_1 be the corresponding 2DFA described in Definition 6.

Consider the constant-space verifier V described in Figure 1. The setting of the values of parameters m and p in that algorithm will be discussed later. The algorithm consists of m rounds. In each round, the prover P that communicates with V is supposed to report the readings of the head of N_1 at each step of N_1's computation on the input string w, and V randomly decides whether to use this information to simulate M or to check the correctness of P's messages by simulating N_1. If, in any round, V discovers that P is lying about those readings, or a simulation of M ends in rejection, V rejects. Otherwise, it accepts.

[6]This definition corresponds to a restricted version of the concept of *safe* heads, introduced in [14].

On input w:

Do the following for m rounds:
  Move V's input head to the right endmarker and then return it to the left endmarker, thereby giving P time to move its own input head to the left endmarker.
  Send P the message `start new round`.

CHOOSE Randomly pick one of the following simulations (Stage SIM-N_1 or Stage SIM-M) to execute in this round. (In either case, the prover will be asked to send the sequence of symbols that are supposed to be scanned by the head of N_1's computation on input w throughout that stage.)

SIM-N_1 [PROBABILITY: $1-p$] Simulate N_1 on w, checking if the head readings match those reported by P at every step. If a mismatch is detected, *reject*. Otherwise, end this stage when the simulation reaches a halting state of N_1.

SIM-M [PROBABILITY: p] Simulate M's computation on w by using the symbol sent by P as H_1's reading at each step, and using V's head to imitate H_2's trajectory on the input. If the simulation halts with rejection, *reject*. If the simulation halts with acceptance, end this stage.

  Send P the message `this round is over`.
(All rounds were completed without rejection.) *Accept*.

Figure 1. A verifier V for a language recognized by 2DFA(2) M.

V does not trust P, which may lie about H_1's readings in attempts to trick V into accepting or even looping forever without reaching a halting state when the input is not in L. V will therefore rely on P's messages with a very low probability $p \in (0, 0.5)$, and validate that P is indeed transmitting N_1's head readings with the remaining high probability, $1-p$.

Figure 2 depicts an "honest" prover algorithm that leads V to acceptance with probability 1 whenever the input string w is in L: No matter what sequence of random choices is made by V, all rounds will be completed without rejection, since the prover is faithful to N_1, and simulations of M based on its transmissions for the readings of H_1 end in acceptance by M. The runtime of V in this case is polynomially bounded, since the simulations of both N_1 and M will take linear time.

On input w:

Repeat the following in an infinite loop:
  Move the input head to $\triangleright$.
  Wait for V to send the message `start new round`.
  Simulate N_1 on w, sending the head readings to V at each step until V sends the message `this round is over`.

Figure 2. A prover that simulates N_1 on its input and reports its head readings in each round.

It remains to conclude the proof by showing that no prover algorithm P^* can make V fail to reject (i.e., cause it to accept or loop) with probability more than $\varepsilon > 0$, where we can tune ε to be as small as we desire by setting m and p appropriately, and also that the probability of V halting in polynomial expected time can be tuned similarly.

V rejects any prover which does not provide a sequence of purported head readings of N_1 in response to its requests in Stages SIM-N_1 and SIM-M. At those stages, the symbols that P^* provides will either match or not match the actual readings of N_1. In the following analysis, let t_i denote the probability of the event that the symbols provided by P^* match the readings of N_1 when V is in its ith round of verification.

If the input w is not a member of L, one of the following four combinations will occur in each round i of V's processing of P^*'s transmissions:

1. V simulates N_1, and determines that P^*'s transmissions about the head readings are correct. In this case, V will advance to the next round in $O(n)$ steps. The probability of this event is $(1-p)t_i$.
2. V simulates N_1, and determines that P^*'s transmission about the head readings do not match what V sees for itself. In this case, V will reject in $O(n)$ steps. The probability of this event is $(1-p)(1-t_i)$.
3. V simulates M while P^* transmits the correct head readings for N_1. In this case, V will reject in $O(n)$ steps. The probability of this event is pt_i.
4. V simulates M while P^* transmits fictitious head readings for N_1. In this case, V can reach an incorrect conclusion that M accepts w and pass this round, or it can be fooled into "simulating" a nonexistent computation of M forever. The probability of this event is at most $p(1-t_i)$.

Based on this, the probability that V will reject, i.e., $\Pr_{\text{rej}}(V, P^*, w)$, is at least $f(1)$, where the function f is defined recursively as follows:

$$f(i) = \begin{cases} (1-p)t_i f(i+1) + (1-p)(1-t_i) + pt_i & \text{for } i < m \\ (1-p)(1-t_m) + pt_m & \text{for } i = m \end{cases}$$

Consider the following series of observations:

1. $f(i)\Big|_{t_i=0} = 1-p$.
2. $f(i)\Big|_{t_i=1} = \begin{cases} f(i+1) + p(1-f(i+1)) & \text{for } i < m \\ p & \text{for } i = m. \end{cases}$
3. $f(i)$ is linear in terms of t_i, and t_i is between 0 and 1. Thus, $f(i)$ is between the above two extremes. This implies that
$$f(i) \geq \begin{cases} \min\{\, 1-p, f(i+1) + p(1-f(i+1)) \,\} & \text{for } i < m \\ \min\{\, 1-p, p \,\} & \text{for } i = m. \end{cases}$$
4. Observation 3 implies that if some $f(i)$ is greater than or equal to $1-p$, then so are all $f(j)$ for $j < i$, and in particular, $f(1) \geq 1-p$.
5. Finally, for all $p > 0$, there exists an m that forces there to exist an $f(i) \geq 1-p$. To prove this by contradiction, assume that there exists a $p > 0$ such that for all m, $f(i) < 1-p$ (equivalently, $1 - f(i) > p$) holds for all $f(i)$. However, then, for such a p and in conjunction with Observation 3, we get
$$1 - p > f(i) \geq f(i+1) + p(1 - f(i+1))$$
for $i < m$. Applying this inequality repeatedly, and then using the assumption once again, we get
$$f(1) \geq f(m) + \sum_{i=2}^{m} p(1 - f(i)) > f(m) + (m-1)p^2.$$
Regardless of what $p > 0$ is, a large enough m (say, $2\left\lceil \frac{1}{p^2} \right\rceil + 1$) results in $f(1) > 1$, a contradiction.

We therefore conclude that the error bound ε for V failing to reject with P^* can be brought arbitrarily close to 0: For any desired value of ε, setting $p = \varepsilon$ and fixing m to an accordingly large value that exists by the argument above guarantees that $\Pr_{\text{rej}}(V, P^*, w) \geq 1 - \varepsilon$.

The probability that V can be tricked to loop is also bounded by ε. With the remaining probability, V's runtime will be bounded by $O(n)$. □

3. Proofs of Quantumness: A Definition

We now present a framework where a classical verifier in an interactive proof system can distinguish whether the prover it is communicating with is a quantum machine or not. Although we will see that one can presently obtain "unconditional" conclusions about quantumness for very slowly growing space bounds, our definition also accommodates similar protocols between bigger machines by allowing the verifier and the prover larger space bounds.

In the following, we consider a space-bounded verifier (Definition 3) which halts with high probability within polynomial expected runtime, and which can be paired by any quantum or classical prover operating under the same space bound to form an interactive proof system.

Definition 7. *A verifier V is said to* check proofs of quantumness (with halting probability p_h) *if*

- *there exists a number $p_h > \frac{1}{2}$ such that, on any input and paired with any prover, V halts with probability at least p_h within expected polynomial time, and fails to halt with the remaining probability,*
- *V runs within space $s(n)$ for some function c of the input length n, and*

- *there exists a number (the* success gap*)* $\Delta > 0$ *and a quantum prover* P^Q *that runs within space* $s(n)$ *such that, for every classical prover* P^C *that runs within space* $s(n)$*, there exists an infinite set of strings* W_{P^C} *such that*

$$\inf\{\, \Pr_{\text{acc}}(V, P^Q, w) \mid w \in W_{P^C} \,\} - \sup\{\, 1 - \Pr_{\text{rej}}(V, P^C, w) \mid w \in W_{P^C} \,\} \geq \Delta.$$

Intuitively, V expects the prover to prove that it can solve a problem instance encoded in their common input string w correctly. As we will see below, one can start the construction of such a protocol by picking the underlying computational problem to be one that is provably impossible for an $o(\log \log n)$-space PTM, but feasible for a 2QCFA. This will mean that there exist infinitely many input strings (set W_{P^C} in Definition 7) for which a particular classical prover P^C claiming to be able to solve the problem will have to "make up" an answer, which will be incorrect, (and whose purported proof will therefore be "caught" as erroneous) by V with a significant probability. On the other hand, the quantum prover employing the correct algorithm to solve the problem is also supposed to prove its result to V. We will note that not all of the problems with quantum advantage listed in Section 2.1 seem to be associated with such "proofs" that can be generated and communicated between machines with so small resource bounds. For the problems which do accommodate such proofs, V will accept the claim (to quantumness) of a genuine quantum prover running the correct algorithm with considerably higher probability than that of any small-space PTM P^C with the same claim when running on strings in W_{P^C}.[7]

Let us take a moment to consider the ordering of the quantifiers in Definition 7. At first sight, one may wonder whether we can use the same set of strings to distinguish between the "correct" (quantum) prover (which is supposed to succeed on all members of all such "test sets" anyway) and all classical provers, and move the quantifier for the input strings to the left accordingly. This is, however, impossible for the following reason: We will be presenting many example protocols where, as mentioned above, the prover is supposed to first solve a problem that is difficult for small-space PTMs and then generate and transmit a proof of correctness of the obtained answer. In those protocols, once an answer to the original problem is at hand, this second stage of communicating the proof is an "easy" task (for even a 2PFA) to realize. Therefore, if the test set (say, W) is fixed, then for any one of its finite subsets, say, W', there exists a classical prover P' which is "hardwired" to respond to inputs in W' by communicating the correct answers and their proofs to the verifier. Such a prover would succeed with very high probability on members of W', and so the success gap required by the inequality at the end of the definition would not be achieved.

We would like Definition 7 to represent a realistic framework for testing some device for quantumness in a reasonable amount of time. This necessitates us to clarify an aspect about the runtimes of the provers appearing in the interactive proof protocols to be described below. In most of the early work on interactive proof systems, the prover was modeled as a magically powerful entity which could compute anything (including uncomputable functions) in a single step. The limitations that later papers [32,33] did impose on the prover are different than ours in an important respect: Condon and Ladner [32] study provers that are restricted to have polynomial-size strategies, but this still means that any actual machine in the role of the prover would need superpolynomial runtime under standard complexity assumptions. Goldwasser et al. [33] do impose a polynomial time bound on the prover, but their model clearly does not require the prover to run simultaneously with the verifier, whose runtime is significantly shorter. In our model, the understanding is that the prover and the verifier start working simultaneously, at which point they access the input tape for the first time, and the process ends when the verifier halts by accepting or rejecting. In our examples, the verifier will be imposing a polynomial time limit on the prover to declare its answer to the problem instance, so the prover will not be able to run an exponential-time algorithm to completion for many inputs. On the other hand, the small-space restriction introduces a complication that verifiers with $\Omega(\log n)$ memory do not face: Our protocols will allow some malicious provers to fool the corresponding verifier, which is not able to measure the runtime while simultaneously checking a purported proof, to run forever without reaching a verdict, with nonzero probability. This is why Definition 7 requires a gap between the minimum acceptance probability caused by a quantum prover and the maximum probability that any classical prover can trick the verifier either to announce acceptance or to run forever instead of rejecting.

[7] The "parity game" [30,31], which cannot be won by any purely classical couple with probability greater than 0.75, whereas a couple making use of quantum entanglement is ensured to win with probability greater than 0.85, can be seen as another "test setup" where the distinction is based on such a gap between the probabilities with which quantum and classical players can succeed in "winning" (i.e., passing the test).

We will use the following common template when constructing verifiers that check proofs of quantumness: One starts with a promise problem $\mathbf{P} = (\mathbf{P}_{\text{Yes}}, \mathbf{P}_{\text{No}})$ that is impossible for small-space polynomial-time probabilistic machines, but computable by quantum machines within those resource bounds. $\mathbf{P}$ should also have the property that each question's correct answer has an interactive proof where both sides are small-space machines. We build a protocol where a verifier first runs a polynomial-time "clock" to give the prover time to compute and announce the answer to the common input string. The prover is then supposed to prove the correctness of this answer to the verifier with high probability in polynomial time. We say that a verifier built according to this template *checks proofs of quantumness based on* $\mathbf{P}$.

4. Proofs of Quantumness Based on DS-FK Problems

In this section, we will demonstrate a method to convert some of the exponential-time 2QCFA algorithms for the languages in Table 1 to (polynomial-time) tests of quantumness. An important tool to be used in that regard is the following theorem proved by Freivalds and Karpinski [8], building on work by Dwork and Stockmeyer [7].

Theorem 2. *Let* $\mathtt{A}, \mathtt{B} \subseteq \Sigma^*$ *with* $\mathtt{A} \cap \mathtt{B} = \emptyset$. *Suppose there is an infinite set* I *of positive integers and functions* $g(m), h(m)$ *such that* $g(m)$ *is a fixed polynomial in* m, *and for each* $m \in I$, *there is a set* W_m *of words in* Σ^* *such that:*

1. $|w| \leq g(m)$ *for all* $w \in W_m$,
2. *there is a constant* $c > 1$ *such that* $|W_m| \geq c^m$,
3. *for every* $w, w' \in W_m$ *with* $w \neq w'$, *there are words* $u, v \in \Sigma^*$ *such that:*
   (a) $|uwv| \leq h(m), |uw'v| \leq h(m)$, *and*
   (b) *either* $uwv \in \mathtt{A}$ *and* $uw'v \in \mathtt{B}$, *or* $uwv \in \mathtt{B}$ *and* $uw'v \in \mathtt{A}$.

Then, if a PTM with space bound $s(n)$ *separates* $\mathtt{A}$ *and* $\mathtt{B}$, *then* $s(h(m))$ *cannot be in* $o(\log m)$.

Definition 8. *For any language* $\mathtt{L} \subseteq \Sigma^*$, *the padded language* $\mathtt{PAD}(\mathtt{L}) \subseteq \Sigma_\star^*$ *is obtained by appending the corresponding string* $\star^{2^{|x|}-|x|}$ *to each string* $x \in \mathtt{L}$, *where* $\Sigma_\star = \Sigma \cup \{ \star \}$ *for some* $\star \notin \Sigma$. *Formally,*

$$\mathtt{PAD}(\mathtt{L}) = \left\{ xy \;\middle|\; x \in \mathtt{L}, y = \star^{2^{|x|}-|x|} \right\}.$$

Given a string $xy \in \mathtt{PAD}(\mathtt{L})$, we refer to the prefix $x \in \mathtt{L}$ as the *core* (of xy) and the suffix $y \in \{ \star \}^*$ as the *padding* (of xy).

We note that it is impossible for a PTM running in space $o(\log \log n)$ to determine whether its input is in this form with the relationship prescribed in Definition 8 between the length of the core and the padding in polynomial time: The set of strings of the form xy, where $x \in \Sigma^*$ and $y = \star^{2^{|x|}-|x|}$ is nonregular, and no such small-space machine can recognize a nonregular language in polynomial expected time [21].[8] (We do not know if this task of checking for an exponential-length padding in the input is also impossible for small-space polynomial-time QTMs.[9])

To exemplify the way we will use Theorem 2, let us consider the languages $\mathtt{PAD}(\mathtt{PAL})$ and $\mathtt{PAD}(\overline{\mathtt{PAL}})$, where $\mathtt{PAL}$ is the set of palindromes on the alphabet $\Sigma = \{ a, b \}$. In the template of Theorem 2, let I be a set containing only even numbers, $g(m) = m, h(m) = 2^m$, and $W_m = \Sigma^{\frac{m}{2}}$. For every $w, w' \in W_m$, let the corresponding u be the empty string, and the corresponding v be the string $w^R \star^{2^m - m}$. It follows that no PTM with a space bound in $o(\log \log n)$ can separate $\mathtt{PAD}(\mathtt{PAL})$ and $\mathtt{PAD}(\overline{\mathtt{PAL}})$.

$\mathtt{PAL}$ is just one of a number of interesting languages that lead to this kind of impossibility result. Call any separation problem $(\mathtt{A}, \mathtt{B})$ that matches the pattern specified in Theorem 2 with the functions g and h respectively set to be $m \mapsto m$ and $m \mapsto 2^m$ (as in the above example) a *DS-FK problem*.[10] All DS-FK problems are impossible for $o(\log \log n)$-space PTMs. In the following, we note a few more DS-FK problems based on languages we saw in Table 1. λ denotes the empty string.

To see that $(\mathtt{PAD}(\mathtt{TWIN}), \mathtt{PAD}(\overline{\mathtt{TWIN}}))$ is a DS-FK problem, set I to be the set of all odd numbers greater than 1, and $W_m = \{ a, b \}^{\frac{m-1}{2}}$. For any pair of distinct elements $w, w' \in W_m$, let $u = \lambda$, and $v = \# w \star^{2^m - m}$.

[8] In Appendix A, we generalize the result by Dwork and Stockmeyer [21] on the inability of these machines to recognize regular languages in polynomial expected time, showing that this remains true for PTMs that do not necessarily halt with probability 1, as long as the "halting part" of the computation has polynomial expected runtime.

[9] Note that the $\mathtt{POWER}$ language (Table 1) can be recognized by an exponential-time 2QCFA.

[10] From the names of Dwork, Stockmeyer [7], Freivalds, and Karpinski [8].

To see that $(\mathtt{PAD}(\mathtt{MULT}), \mathtt{PAD}(\overline{\mathtt{MULT}}))$ is a DS-FK problem, set I to be the set of all odd numbers greater than 5, and $W_m = \{1\}\{0,1\}^{\frac{m-5}{2}}$. For any pair of distinct elements $w, w' \in W_m$, let $u = \lambda$, and $v = \#1\#w\star^{2^m-m}$.

It is easy to see that $(\mathtt{PAD}(\mathtt{EQ}), \mathtt{PAD}(\overline{\mathtt{EQ}}))$ is *not* a DS-FK problem, by considering the following 2PFA M that performs the separation: On an input of the form xy, where x is the core, M simply runs Freivalds' 2PFA algorithm for recognizing $\mathtt{EQ}$ [20] by treating x as the sole input. The resulting algorithm's runtime is exponential in $|x|$, but only polynomial in the overall input length.

We are now ready to present our first proof-of-quantumness protocol.

Theorem 3. *Let* $\mathtt{L}$ *be any language in* $\mathsf{BQTISP}_*(t(n), s(n)) \cap \mathcal{L}(2DFA(2_s))$ *such that* $t(n) = 2^{O(n)}$, $s(n) = o(\log\log n)$, *and* $(\mathtt{PAD}(\mathtt{L}), \mathtt{PAD}(\overline{\mathtt{L}}))$ *is a DS-FK problem. For any number* $p_h < 1$, *there exists a verifier* $V_{\mathtt{L}}$ *that checks proofs of quantumness based on* $(\mathtt{PAD}(\mathtt{L}), \mathtt{PAD}(\overline{\mathtt{L}}))$ *with halting probability* p_h.

Proof. For any language $\mathtt{L}$ with the properties described in the theorem statement, let $Q_{\mathtt{L}}$ be a QTM that recognizes $\mathtt{L}$ with error bound $\varepsilon_{Q_{\mathtt{L}}}$ within space $s(n) = o(\log\log n)$ and expected runtime 2^{kn}, for some integer $k > 0$ and all sufficiently large input lengths n. Let M and M' be the 2DFA(2)s (with at least one supersafe head) that recognize $\mathtt{L}$ and $\overline{\mathtt{L}}$, respectively.[11] We describe a verifier $V_{\mathtt{L}}$ (Figure 3) and quantum prover $P_{\mathtt{L}}^Q$ (Figure 4) that will be shown to lead $V_{\mathtt{L}}$ to acceptance with high probability for every input satisfying the promise. $V_{\mathtt{L}}$ has parameters named k, k_{p}, ε_{p}, and ε_V that will be set to appropriate values, as described below. Since $P_{\mathtt{L}}^Q$ runs only on the very short core x (of length $m = \log n$) of the input string, its expected runtime will be short enough so that it will be able to run to completion before the "clock" stage of $V_{\mathtt{L}}$ terminates. After the prover announces its answer to the separation problem, it is supposed to engage in a dialogue with $V_{\mathtt{L}}$ to prove that the appropriate 2DFA(2) (M or M', depending on the announced answer) really accepts the input x. We now provide a detailed analysis.

On input xy of length n, such that $x \in \Sigma^*$, $|x| = \log_2 n$, and $y \in \star^*$ for $\star \notin \Sigma$:

CLOCK Run a "clock" (Fact 1) which has an expected runtime in $O(n^{k+1})$, and terminates in fewer than $k_{\mathrm{p}} n^k$ steps with probability at most ε_{p}. At every step of this procedure, ask the prover if it has computed a Yes/No answer for xy, and pass to the next stage if you receive such an answer. *Reject* if the prover has still not sent a claim about xy when the clock times out.

VERIFY If the prover has answered $\boxed{\text{Yes}}$ (resp. $\boxed{\text{No}}$), run the verification algorithm specified in Figure 1 (codified to treat only the x segment on the tape as its input string) to check the membership of x in $\mathtt{L}$ (resp. $\overline{\mathtt{L}}$) with error bound ε_V using the 2DFA(2) M (resp. M') described in the text. *Accept* (resp. *reject*) if that algorithm accepts (resp. rejects).

Figure 3. A verifier based on a DS-FK problem $(\mathtt{PAD}(\mathtt{L}), \mathtt{PAD}(\overline{\mathtt{L}}))$, where $\mathtt{L} \in \mathcal{L}(2\mathrm{DFA}(2_{\mathrm{s}}))$.

On input xy of length n, such that $x \in \Sigma^*$, $|x| = \log_2 n$, and $y \in \star^*$ for $\star \notin \Sigma$:

Run the QTM $Q_{\mathtt{L}}$ (codified to treat only the x segment on the tape as its input string) to decide whether x is in $\mathtt{L}$ or not. At every step during this procedure, give the answer $\boxed{\text{still computing}}$ to each question of the verifier. When $Q_{\mathtt{L}}$ halts, send its result ($\boxed{\text{Yes}}$ or $\boxed{\text{No}}$) to the verifier.
Run the "proof" algorithm specified in Figure 2 (codified to treat only the x segment on the tape as its input string) using the 2DFA associated with the supersafe head of the machine M (or, identically, M') described in the text.

Figure 4. A quantum prover based on $(\mathtt{PAD}(\mathtt{L}), \mathtt{PAD}(\overline{\mathtt{L}}))$, where $\mathtt{L} \in \mathsf{BQTISP}_*(2^{kn}, s(n)) \cap \mathcal{L}(2\mathrm{DFA}(2_{\mathrm{s}}))$.

Consider the behavior of $V_{\mathtt{L}}$ when faced with the prover $P_{\mathtt{L}}^Q$ on some common input string xy on the input alphabet $\Sigma_\star$. $P_{\mathtt{L}}^Q$'s expected runtime on sufficiently long inputs is at most $2^{km} = 2^{k\log n} = n^k$. With probability at least $1 - \varepsilon_{\mathrm{p}}$, $V_{\mathtt{L}}$ will grant $P_{\mathtt{L}}^Q$ at least $k_{\mathrm{p}} \cdot n^k$ steps to announce an answer. By Markov's inequality, $Q_{\mathtt{L}}$'s computation will need more than $k_{\mathrm{p}} \cdot n^k$ steps with probability at most $\frac{n^k}{k_{\mathrm{p}} n^k} = \frac{1}{k_{\mathrm{p}}}$. If it can run to completion, $Q_{\mathtt{L}}$ will return

[11] $\mathcal{L}(2\mathrm{DFA}(2_{\mathrm{s}}))$ is closed under complementation.

the correct answer with probability at least $1 - \varepsilon_{Q_{\mathtt{L}}}$. Since the VERIFY stage of $V_{\mathtt{L}}$ will accept proofs of a correct answer with probability 1, we conclude that $\inf\{ \Pr_{\text{acc}}(V_{\mathtt{L}}, P_{\mathtt{L}}^{Q}, w) \mid w \in \Sigma^{m}\Sigma_{\star}^{2^m - m} \}$ nears at least

$$\left(1 - \varepsilon_{\text{p}}\right) \cdot \left(1 - \frac{1}{k_{\text{p}}}\right) \cdot \left(1 - \varepsilon_{Q_{\mathtt{L}}}\right) \tag{1}$$

as n (and therefore m) grows. Recall from Section 2.1 that there exists a $Q_{\mathtt{L}}$ for any desired small positive value of $\varepsilon_{Q_{\mathtt{L}}}$, and a choice of $\varepsilon_{Q_{\mathtt{L}}}$ sets k to a corresponding value. Note that ε_{p} can be set from the outset to a positive value that is as small as desired by plugging in an appropriate parameter for the clock, and k_{p} can be selected to be as large as necessary. The probability (Expression (1)) that $V_{\mathtt{L}}$ will accept a proper quantum prover can therefore be set to be as near to 1 as desired. $P_{\mathtt{L}}^{Q}$ runs within space $o(\log \log n)$.[12]

Let us now analyze what happens when $V_{\mathtt{L}}$ is faced with some classical $s(n)$-space prover P^{C}. If P^{C} fails to announce a Yes/No answer during the CLOCK stage of $V_{\mathtt{L}}$, it will be rejected by probability 1. Since no $o(\log \log n)$-space PTM can separate $\mathtt{PAD}(\mathtt{L})$ and $\mathtt{PAD}(\overline{\mathtt{L}})$, any algorithm realized by P^{C} to return an answer during the clock stage must answer incorrectly (with probability at least $\frac{1}{2}$) for an infinite[13] set of input strings W'_{P^C}. This incorrect answer will be exposed (with probability at least $1 - \varepsilon_V$, by Theorem 1) when it is checked against the judgment of M (or M') in the VERIFY stage, leading to P^{C} being rejected with probability at least $\frac{1-\varepsilon_V}{2}$ in this case. W'_{P^C} has an infinite subset W_{P^C} of strings that are sufficiently long for the above-mentioned lower bound for the acceptance probability associated by $P_{\mathtt{L}}^{Q}$ to hold. We conclude that, for any such classical prover P^{C},

$$\sup\{ 1 - \Pr_{\text{rej}}(V_{\mathtt{L}}, P^{C}, w) \mid w \in W_{P^C} \} \leq \frac{1 + \varepsilon_V}{2}.$$

ε_V, which also bounds the probability that $V_{\mathtt{L}}$ may be tricked to loop forever, is a value that can be set to be as small as one desires too, and therefore the success gap

$$\inf\{ \Pr_{\text{acc}}(V_{\mathtt{L}}, P_{\mathtt{L}}^{Q}, w) \mid w \in W_{P^C} \} - \sup\{ 1 - \Pr_{\text{rej}}(V_{\mathtt{L}}, P^{C}, w) \mid w \in W_{P^C} \}$$

can be set to be greater than any desired value $\Delta < \frac{1}{2}$.

As established in Theorem 1, the runtime of the VERIFY stage is linear in the length of the core (except in the low-probability case where $V_{\mathtt{L}}$ is deceived into looping), so $V_{\mathtt{L}}$ halts with probability $1 - \varepsilon_V$ in expected time $O(n^{k+1})$. □

Since PAL and TWIN are known to satisfy all the requirements of Theorem 3, the associated problems $(\mathtt{PAD}(\mathtt{PAL}), \mathtt{PAD}(\overline{\mathtt{PAL}}))$ and $(\mathtt{PAD}(\mathtt{TWIN}), \mathtt{PAD}(\overline{\mathtt{TWIN}}))$ are suitable for testing claims to quantumness by small-space machines without relying on unproven assumptions about the impossibility of a particular problem for a specific model of computation.

5. Verification Beyond $\mathcal{L}(\text{2DFA}(2))$

The verifier template described in the proof of Theorem 3 can be modified to support protocols based on problems with different properties. Consider the language SQUARE, which is described in Table 1. For any error bound $\varepsilon_Q > 0$, this language is recognized by a 2QCFA whose expected runtime on all inputs of sufficient length n is at most 2^{kn} for some $k > 0$ corresponding to ε_Q. It is not known whether SQUARE is recognizable by either a 2DFA(2) or a sublogarithmic-space PTM. Let $\mathtt{SUBSQUARE} = \{ a^i b^j \mid j < i^2 \}$. We will describe a protocol whereby a quantum prover can demonstrate its ability to solve the separation problem $(\mathtt{PAD}(\mathtt{SQUARE}), \mathtt{PAD}(\mathtt{SUBSQUARE}))$ to a classical verifier V, depicted in Figure 5.

[12] Note that $P_{\mathtt{L}}^{Q}$ would still run within space $o(\log \log n)$ on inputs satisfying the promise of having exponentially long padding if we allowed $Q_{\mathtt{L}}$ a bigger space budget of $s(n) = o(\log n)$, but Definition 7 stipulates that the prover should respect the common space bound on all possible inputs.

[13] Since a machine that performs the separation answers all inputs correctly with probability greater than $\frac{1}{2}$, any machine T that *fails* to separate $\mathtt{PAD}(\mathtt{L})$ and $\mathtt{PAD}(\overline{\mathtt{L}})$ must be associated with a nonempty set of strings that are answered correctly with probability at most $\frac{1}{2}$. Furthermore, this set must be infinite, since any PTM which acts undesirably in response to only finitely many input strings can be "repaired" without changing its time or space bounds.

The parameters k, ε_{p}, k_{p}, c_F, d_F, p, and h_V in the description of V allow us to tune the success gap of the protocol, as will be explained below. On an input string of the form $a^i b^j \star^*$, this verifier expects a prover to first announce its claim about the input string's membership in the allotted time, and then to repeatedly transmit segments composed of i symbols. One probabilistic branch of V verifies that each segment (in a sufficiently long prefix of this transmission) is indeed of the required length, while another branch uses this "ruler" provided by the prover to determine whether the block of b's in the input consists of i mini-blocks of length i, i.e., has length i^2. For this purpose, V compares the number of segment separator symbols transmitted by the prover in the time it takes V to scan the segment of b's in the input with the number of a's in the input, using the technique invented by Freivalds [20] to build a 2PFA that recognizes $\mathtt{EQ}_{\mathtt{ab}}$. A detailed analysis will follow the description of the quantum prover below.

On input xy of length n, such that $x = a^i b^j$, $j \leq i^2$, $|x| = \log_2 n$, and $y \in \star^*$:

CLOCK Run a "clock" (Fact 1) which has an expected runtime in $O(n^{k+1})$, and terminates in fewer than $k_{\mathrm{p}} n^k$ steps with probability at most ε_{p}. At every step of this procedure, ask the prover (P) if it has computed a Yes/No answer for xy, and pass to the next stage if you receive such an answer. *Reject* if the prover has still not sent a claim about xy when the clock times out.
Move the input head to the right end of the core x and then return it to the left endmarker, thereby giving P time to move its own input head to the left endmarker.
Send P the message $\boxed{\text{start proof}}$.

CHOOSE Randomly pick one of the following two procedures (CHECK-RULER or FREIVALDS) to execute. (In either case, the prover will be expected to transmit a sequence of "$z^{i-1}\#$" segments throughout that stage. V will drive this transmission by requesting each symbol, pausing for one step after each $\#$ to give the prover (see Figure 6) time to move its head to the proper location for starting the next segment.)

CHECK-RULER [PROBABILITY: p] Repeatedly move the head between the two ends of the maximal prefix of a's in the input. On each a, throw a coin with a probability 2^{-h_V} of coming up heads. If that coin comes up heads in all i steps of a walk of the head from one end of the a^i segment to the other, *accept*. If the prover is detected to fail to transmit segments of exactly $i-1$ contiguous z's punctuated by $\#$'s, *reject*.

FREIVALDS [PROBABILITY: $1-p$] Initialize counters C_a and $C_\#$ to 0, and then repeatedly move the input head between the two ends of the core $a^i b^j$. In each pass of the input, perform the following controls parallelly:

Count both the number of a's in the input and the number of $\#$'s sent by P while the input head is scanning b's modulo the parameter c_F. If the two counts are not congruent (mod c_F) at the end of the pass, or if the prover does not send a $\#$ to coincide with the last b that is scanned in this pass, conclude that $j \neq i^2$ and *accept* (resp. *reject*) if the answer previously announced by P agrees (resp. disagrees) with this conclusion.

Throw a fair coin for each a that the input head scans in the input (call these "the group of a coins") and for each $\#$ sent by P while the input head is scanning a b ("the $\#$ coins"). If *all* coins in one of these groups turn up heads during a complete (left-to-right or right-to-left) sweep of the core by the input head, this is a "goal" scored by that group. If exactly one group scores a goal during a sweep, this is a "win" for that group. If this sweep is a win for the group of a (resp. $\#$) coins, increment the counter C_a (resp. $C_\#$) by 1. If the total number of wins reaches d_F, terminate the loop and check the counter values. If both counter values are nonzero, *accept* (resp. *reject*) if the answer sent by P in the CLOCK stage was $\boxed{\text{Yes}}$ (resp. $\boxed{\text{No}}$). If a counter value is zero, *reject* (resp. *accept*) if the answer sent by P in the CLOCK stage was $\boxed{\text{Yes}}$ (resp. $\boxed{\text{No}}$).

Figure 5. A verifier V based on $(\mathtt{PAD}(\mathtt{SQUARE}), \mathtt{PAD}(\mathtt{SUBSQUARE}))$.

On input xy, where $x = a^i b^j$, $y \in \star^*$, and $|x| = \log_2 |xy|$:

Run the 2QCFA $Q_{\texttt{SQUARE}}$ (codified to treat only the x segment on the tape as its input string) to decide whether $x \in \texttt{SQUARE}$ or not. At every step during this procedure, give the answer $\boxed{\text{still computing}}$ to each question of the verifier. When $Q_{\texttt{SQUARE}}$ halts, send its result ($\boxed{\text{Yes}}$ or $\boxed{\text{No}}$) to the verifier.
Move the input head to the first a in the input.
Wait for V to send the message $\boxed{\text{start proof}}$.
Repeatedly move the head between the two ends of the block of a's on the input tape. On each pass, perform the following actions:
Skip the first a. For all the remaining a's, send a z to the verifier upon its symbol request when the head is on this a. For the non-a symbol that you scan in the input tape when you walk off the block, send a $\#$ to the verifier upon its symbol request.

Figure 6. A quantum prover based on $(\texttt{PAD}(\texttt{SQUARE}), \texttt{PAD}(\texttt{SUBSQUARE}))$.

The first stage of the quantum prover P^Q in Figure 6 runs the 2QCFA algorithm $Q_{\texttt{SQUARE}}$ for recognizing $\texttt{SQUARE}$ on the core of the form $a^i b^j$ in the same fashion as the prover in Figure 4. Since the CLOCK stage of the verifier in Figure 5 is identical with that of Figure 3, the first stage of P^Q will have sufficient time to finish $Q_{\texttt{SQUARE}}$ and report a correct Yes/No answer with a probability that can be set to be as close to 1 as one desires by choosing an appropriately low-error version of $Q_{\texttt{SQUARE}}$, and plugging in the values of k, k_{p}, and ε_{p} accordingly, by the analysis we presented in the proof of Theorem 3. In the remainder of its execution, the prover responds to the symbol requests from the verifier by sending an infinitely long sequence $z^{i-1}\#z^{i-1}\#z^{i-1}\#\cdots$ made up of segments of length i. Note that the "proof" transmitted by P^Q does not depend on the answer that it announces earlier.

In the CHOOSE stage of Figure 5, V will choose to execute the control in the CHECK-RULER stage with probability p. Since P^Q transmits the proper symbol sequence dictated by the protocol, V's CHECK-RULER stage will accept with probability 1 within expected runtime $i2^{h_V i}$ when faced with P^Q. If V chooses to execute the FREIVALDS stage,[14] V simply runs a modified version of Freivalds' algorithm for $\texttt{EQ}_{\texttt{ab}}$, which has been codified to compare the number of a's in the core with the number of $\#$'s transmitted by P^Q while V's input head is on the b-block of the core. Since P^Q is honest to its mission, it will transmit the same number of $\#$'s during every sweep of the core. As proven by Freivalds [20], one can set the parameters c_F and d_F to guarantee that the FREIVALDS stage arrives at the correct decision about whether the two numbers being compared are equal or not with probability $1 - \varepsilon_F$, for any desired value of $\varepsilon_F > 0$. That stage rejects P^Q only if V's conclusion about the core disagrees with the answer previously sent by P^Q. We have already established that we can make that answer's correctness probability as close to 1 as we wish, and the parameters c_F and d_F can be set to tune the probability of a disagreement with that correct answer to a value below any desired positive bound as long as the correct ruler is transmitted. We conclude that $\inf\{\, \Pr_{\text{acc}}(V, P^Q, w) \mid w \in W \,\}$ can be arranged to be arbitrarily close to 1 for an infinite set W containing all padded strings that are sufficiently long.

Note that, if it chooses to execute the FREIVALDS stage, V bases its decision on the assumption that the prover P that it is communicating with will send a proper sequence of segments, each of length i. The runtime of Freivalds' algorithm, which V executes in this case, supplies a bound on the number of symbols that V will request from P. That algorithm's expected runtime is $O(m2^r)$, where m is the length of the string it is working on, r is the number of coins it tosses during a single pass of that string, and the parameters c_F and d_F determine the constant of proportionality "hidden" by the big-O notation [21]. In this case, the promise of the problem guarantees that $m \leq i + i^2$ and $r \leq 2i$. We can thefore say that, for sufficiently long inputs, this procedure has expected runtime under $2^{k_F i}$, for a sufficiently large number k_F that depends on the desired ε_F. Consider a parameter $K > 1$. The probability of the runtime of the FREIVALDS stage exceeding $K2^{k_F i}$ is at most $\frac{1}{K}$, by Markov's inequality. Since $i > 0$, $\frac{1}{K}$ is also an upper bound for the probability that V will request more than $iK2^{k_F i}$ symbols from the prover during this stage.

Let us now consider a classical prover P^C interacting with V. Note that V will reject any prover that fails to return a Yes/No answer, or to respond to a symbol request promptly, with probability 1. Assume that P^C has reported a Yes/No answer within the time allotted by the CLOCK stage of V. We will analyze P^C's chances of being accepted in the cases where it adopts one of the following strategies about which symbols to transmit in response to the verifier's subsequent requests:

[14] This happens with probability $1 - p$.

Strategy 1: Transmit a sequence which fails to match the "expected" pattern $(z^{i-1}\#)^+$ in at least one position among the first $iK2^{k_F i}$ symbols

Strategy 2: Transmit a sequence whose first $iK2^{k_F i}$ symbols match the "expected" pattern $(z^{i-1}\#)^+$

If P^C employs Strategy 1 and V chooses (with probability p) to execute the CHECK-RULER stage, V will reject the prover, unless that stage terminates before P^C attempts to transmit the first "defect" in the pattern. In this stage, V repeatedly sweeps the block of a's in the input from end to end, and decides to halt and accept with probability $2^{-h_V i}$ after each such sweep. The probability that this stage will perform more than $K2^{k_F i}$ sweeps (and therefore will request more than $iK2^{k_F i}$ symbols from the prover) is $\left(1-2^{-h_V i}\right)^{K2^{k_F i}}$. So the probability p_1 that P^C will be accepted by employing Strategy 1 is at most $1-p\left(1-2^{-h_V i}\right)^{K2^{k_F i}}$.

If P^C employs Strategy 2, it will be accepted if one of the following events occur:

- V selects the CHECK-RULER stage, which accepts P^C. The probability of this event is at most p.
- V selects the FREIVALDS stage, which reaches the same conclusion about the core as previously announced by P^C, after requesting more than $iK2^{k_F i}$ symbols from the prover. Since P^C can "lie" after some point during its transmission to direct V to reach a conclusion consistent with its previous claim, an upper bound for the probability of this event is $(1-p)\left(\frac{1}{K}\right)$.
- V selects the FREIVALDS stage, which reaches the same conclusion about the core as previously announced by P^C, after requesting no more than $iK2^{k_F i}$ symbols from the prover. The probability of this event is not more than $(1-p)p_{\text{agree}}$, where we define p_{agree} as the probability that those two conclusions agree.

Assuming that P^C is unable to separate PAD(SQUARE) and PAD(SUBSQUARE), there will be a set $W'_{P^C} \subseteq W$ containing infinitely many input strings for which it will send the correct answer with probability at most $\frac{1}{2}$ during the CLOCK stage of V. For any such input, p_{agree}, which is the sum of two terms corresponding to the events where P^C and V agree on the correct and wrong answer, respectively, is at most

$$\frac{1}{2}(1-\varepsilon_F)+\frac{1}{2}(\varepsilon_F)=\frac{1}{2},$$

where we recall that ε_F is the (very small) probability that V comes to the wrong conclusion about the core as a result of running Freivalds' algorithm. Therefore, the probability p_2 that P^C will be accepted when it employs Strategy 2 is at most $p+(1-p)\left(\frac{1}{K}\right)+\frac{1-p}{2}$ for infinitely many inputs.

As one of the two "pure" strategies that we considered must yield the highest acceptance probability possible for P^C,[15] this probability is bounded by

$$\max\left\{\,1-p\left(1-2^{-h_V i}\right)^{K2^{k_F i}},\, p+(1-p)\left(\frac{1}{K}\right)+\frac{1-p}{2}\,\right\}.$$

One wishes this probability to be as low as possible, so consider setting the value of K to a very large number, forcing p_2 to be below a value that is as near to $\frac{1+p}{2}$ as one desires. For any given value of K, one can then set h_V to a correspondingly large value that will force p_1 to be as near to $1-p$ as desired for all sufficiently long strings that form an infinite subset W_{P^C} of W'_{P^C}.[16] We fix p to the value $\frac{1}{3}$ that minimizes $\max\left\{\,1-p,\,\frac{1+p}{2}\,\right\}$, and conclude that the probability that P^C can be accepted by V can be pulled down to any value above $\frac{2}{3}$, obtaining a success gap

$$\inf\left\{\,\Pr_{\text{acc}}(V,P^Q,w)\;\middle|\;w\in W_{P^C}\,\right\}-\sup\left\{\,1-\Pr_{\text{rej}}(V,P^C,w)\;\middle|\;w\in W_{P^C}\,\right\}$$

that can be set to be greater than any desired value $\Delta<\frac{1}{3}$.

Note that V runs in expected polynomial time only on input strings satisfying the promise that the core's length is logarithmic in the length of the input,[17] and is guaranteed to halt with probability 1, unlike the verifiers of Theorem 3.

[15] No probabilistic mixture of the two strategies can yield a higher acceptance probability.

[16] k_F has already been fixed by our choice of c_F and d_F. We make use of the fact that $\lim_{x\to\infty}\left(1-2^{-Ax}\right)^{2^{Bx}}=1$ when $A>B>0$.

[17] This is in contrast to the verifiers of Theorem 3, which satisfy the runtime requirement stated in Definition 7 for all possible input strings.

V can be said to distinguish quantum provers from classical ones only on the condition that no 2PFA can separate $\texttt{PAD}(\texttt{SQUARE})$ and $\texttt{PAD}(\texttt{SUBSQUARE})$. If that assumption is not true, a classical prover can achieve the same performance as the 2QFA in Figure 6. Of course, V can also be seen as a framework enabling a party who has invented *any* fast (classical or quantum) constant-space algorithm for solving the separation problem $(\texttt{PAD}(\texttt{SQUARE}), \texttt{PAD}(\texttt{SUBSQUARE}))$ to demonstrate this to another party without revealing the source code.

6. Concluding Remarks

As we noted in Section 2.1, there exist recognition problems which are known to be solvable in polynomial expected time by 2QCFAs and at least exponential expected time by 2PFAs. This proven power difference between classical and quantum models suggests a variant of the protocol described in Theorem 3, where the underlying problem is simply the recognition of such a language. However, neither Dwork and Stockmeyer's result [21] establishing that any small-space PTM recognizing a nonregular language must run in superpolynomial expected time, nor our generalization (in Appendix A) of that theorem to machines which do not necessarily halt with probability 1 is strong enough to guarantee that the resulting verifier will reject every classical prover with higher probability than the genuine quantum prover for that problem. Those theorems do not preclude a classical machine (say, a 2PFA) which recognizes $\texttt{EQ}$ and runs within a polynomial time bound with probability $(n-1)/n$, and spends exponential time with probability $1/n$ for inputs of length n. If such a machine P^C exists, it will be able to perform the recognition task with very high probability within some polynomially large time given by the verifier, and the argument used to demonstrate the success gap in the proof of Theorem 3 will not work for a verifier faced with P^C. This line of thought about the search for more proof-of-quantumness protocols raises the following questions:

1. Is there a 2PFA that recognizes a nonregular language and runs within a polynomial time bound with probability $1-o(1)$ as a function of the input length?
2. Do there exist interactive proof systems (like the ones presented in Theorem 1) for languages other than $\texttt{PAL}$ and $\texttt{TWIN}$ in Table 1?
3. Which, if any, languages in Table 1 (in addition to $\texttt{PAL}$, $\texttt{TWIN}$, $\texttt{MULT}$, and "word problems," which are closely connected to $\texttt{PAL}$) yield DS-FK problems when one considers separating their padded versions from those of their complements?
4. Is $\mathcal{L}(\text{2DFA}(2)) = \mathcal{L}(\text{2DFA}(2_s))$?

In Section 1, we mentioned the recent result [6] that every language in BQL has an IPS through which a quantum logspace prover can prove the membership of a string in that language to a classical logspace verifier.[18] Although our work can be seen as a step toward a similar framework for the small-space case, the construction in [6] requires space to be at least logarithmic in time, and we conjecture that the answer to the following question is negative:

5. Is there a general small-space IPS framework (e.g., for all languages in $\mathsf{BQTISP}_*(t(n), s(n))$ where $t(n)$ is some polynomial in n and $s(n) \in o(\log\log n)$) through which a quantum $s(n)$-space prover can prove the membership of a string in a language that it can recognize to a classical verifier with the same space bound?

Author Contributions
A.C.C.S. and M.U.G. contributed equally to all aspects of this work. Both authors have read and agreed to the published version of the manuscript.

Funding
This research received no external funding.

Conflicts of Interest
The authors declare no conflicts of interest.

Data Availability Statement
This is a theoretical study, and no new data (other than the detailed definitions and proofs presented) were created.

Acknowledgments
We are grateful to Abuzer Yakaryılmaz and Wei Zhan for their helpful answers to our questions. We also thank the anonymous reviewers for their comments.

[18] In fact, the protocol described in [6] is one-way, that is, the prover simply sends the verifier a single proof string, with no further interaction.

Appendix A. A Generalization of Dwork and Stockmeyer's Theorem

Theorem A1. *Let M be a PTM which recognizes a nonregular language such that for any input w of length n, M runs within space $o(\log\log n)$, halts with probability $h_w \leq 1$ within expected time $T(n)$, and fails to halt with the remaining probability $1 - h_w$. Then, for every $b < 1$,*

$$\log\log T(n) \geq (\log n)^b$$

for infinitely many n. In particular, $T(n)$ is not bounded above by any polynomial in n.

Proof. We start with a quick recap of Dwork and Stockmeyer's proof of the case ([21, Theorem 5.1]) where the PTM halts with probability 1:

For any language $\mathtt{L}$ and number $n > 0$, two strings w and w' are said to be *n-dissimilar*, written $w \nsim_{\mathtt{L},n} w'$, if $|w| \leq n$, $|w'| \leq n$, and there exists a *distinguishing string* v with $|wv| \leq n$, $|w'v| \leq n$, and $wv \in \mathtt{L} \iff w'v \notin \mathtt{L}$. Let $N_{\mathtt{L}}(n)$ be the maximum k such that there exist k distinct strings that are pairwise $\nsim_{\mathtt{L},n}$. It is known [34] that, if $\mathtt{L}$ is nonregular, then $N_{\mathtt{L}}(n) \geq \frac{n+3}{2}$ for infinitely many n.

It is convenient to adopt the following new conventions regarding M: M starts in a special state c_1 (which will only be used for the initial right-to-left pass) with the input head on the right endmarker $\lhd$, moves the input head to the left endmarker without moving the work tape head, and switches to a different state when it arrives at $\rhd$. M keeps track of the rightmost non-blank symbol on the work tape throughout its execution, and performs a subroutine that "cleans" the work tape by replacing all non-blank symbols by blanks and positioning the work head on the leftmost end of the tape immediately before it accepts or rejects.

Let us call an instantaneous description of the collection of the internal state, work tape content, and work head position of M a *megastate* of M. (The familiar notion of a *configuration* of M then corresponds to a pair consisting of a megastate of M and the position of its input tape head.) Since M has space complexity $o(\log\log n)$, $c(n)$, the number of distinct megastates that it can attain while running on an input of length n, grows more slowly than $(\log n)^a$ for any constant $a > 0$. We impose some ordering from 1 to $c(n)$ on the megastates of M, with megastate 1 corresponding to c_1 and a "clean" work tape containing all blank symbols with the work head positioned on the left end. Megastates $c(n) - 1$ and $c(n)$ correspond to the reject and accept states, respectively, both with similarly clean work tapes.

As Dwork and Stockmeyer describe [21], there exists a Markov chain $P_{M,xy}$ with $2c(n)$ states that models the computation of M on the concatenated string xy for any two strings $x, y \in \Sigma^*$, where $|xy| = n$. For $1 \leq j \leq c(n) - 1$, state j of $P_{M,xy}$ corresponds to M being in the configuration with megastate j and the input head on the last symbol of $\rhd x$, and state $c(n) + j - 1$ corresponds to M being in the configuration with megastate j and the input head on the first symbol of $y\lhd$. (State 1 of $P_{M,xy}$ thus corresponds to a configuration that M will be in with probability 1 shortly after the beginning of its computation.) Importantly, state $2c(n) - 1$ of $P_{M,xy}$ corresponds to a disjunction of rejection, infinite loop with the input head never leaving the region $\rhd x$, and infinite loop within the region $y\lhd$. State $2c(n)$ of $P_{M,xy}$ corresponds to acceptance. This Markov chain is "faithful" to M, in the sense that each transition probability of $P_{M,xy}$ equals the probability of the (multi-step) transition between the corresponding configurations of M when running on xy. The probability that $P_{M,xy}$ is absorbed in state $2c(n)$ when started in state 1 equals the probability that M accepts xy. Since M halts with certainty in the case in [21] (where $h_w = 1$ for all w), the chain is absorbed in state $2c(n) - 1$ with the remaining probability. The relationship between the computation of M and $P_{M,xy}$ guarantees that the Markov chain's expected time to absorption (into one of those two states) is at most $T(n)$ as well.

One then assumes that $T(n)$ is smaller (i.e., M runs faster) than dictated by the lower bound in the theorem statement. This assumption and the aforementioned lower and upper bounds for $N_{\mathtt{L}}(n)$ and $c(n)$, respectively, lead to the conclusion that, for the language $\mathtt{L}$ recognized by M and for sufficiently large n, there exist two pairwise $\nsim_{\mathtt{L},n}$ strings w and w', with distinguishing string v, such that the Markov chains $P_{M,wv}$ are $P_{M,w'v}$ are so close (according to a notion of closeness defined in [21]) to each other that M must accept either both or none of the strings wv and wv', contradicting their n-dissimilarity, and concluding the proof. The assumption that $T(n)$ (i.e., the absorption times for both Markov chains) is "small" plays a critical role in this final step.

We now show how to modify the proof above to also work for the case where M may run forever on any input string w with some nonzero probability $1 - h_w$. Call a non-halting configuration C "looping" if M's probability of halting eventually is 0 when it is started from C. Note that M attains at least one looping configuration with nonzero

probability while running on input w if and only if M halts with probability less than 1 on w. Since M recognizes a language, there is a probability greater than $\frac{1}{2}$ that its computation will consist of a finite sequence of non-looping configurations followed by a halting configuration. By the theorem statement, the expected length of such a sequence is at most $T(n)$. A looping computation can occur with probability less than $\frac{1}{2}$, and consists of a finite sequence of non-looping configurations, followed by an infinite sequence of looping configurations.

If M can loop with nonzero probability on an input string xy, the corresponding Markov chain $P_{M,xy}$ constructed using the procedure described above is no longer guaranteed to have a finite, let alone "small," expected absorption time. The given construction does map infinite computations of M where the input head does not leave one of the $\triangleright x$ and $y\triangleleft$ regions to finite paths that end in state $2c(n) - 1$ in the chain. However, if M shuttles its input head back and forth forever between the two regions, the corresponding Markov chain also "runs" forever.

We solve this problem by adding a "postprocessing" step in the construction of $P_{M,xy}$. Let $p_{i,j}$ denote the probability of the transition from state i to state j set by the previously mentioned procedure. We simply "rewire" the chain by diverting every transition to a state corresponding to a looping configuration to the trap state $2c(n) - 1$:

For each state j that corresponds to a looping configuration of M:
  For each i:
    $p_{i,2c(n)-1} \leftarrow p_{i,2c(n)-1} + p_{i,j}$
    $p_{i,j} \leftarrow 0$

The modified chain still models the acceptance probability of M faithfully. All non-halting computations of M are modeled by finite paths in the chain. Note that these paths correspond to sequences of non-looping configurations of M, which also appear in halting computations, so their lengths are accounted for in the expected runtime $T(n)$ of the "halting part" of the machine. We conclude that the modified chain's expected absorption time is also bounded by $T(n)$. The rest of the proof for the general case is identical to that of the case for $h_w = 1$. $\square$

References

1. Z. Brakerski, V. Koppula, U. Vazirani and T. Vidick (2020). "Simpler proofs of quantumness", in S. T. Flammia (ed.), *15th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2020)*, volume 158 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 8:1–8:14, Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
2. P.W. Shor (1997). "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer". *SIAM Journal on Computing*, 26: 5, 1484–1509.
3. U. Mahadev (2022). "Classical verification of quantum computations". *SIAM Journal on Computing*, 51: 4, 1172–1229.
4. O. Regev (2009). "On lattices, learning with errors, random linear codes, and cryptography". *Journal of the ACM*, 56: 6.
5. W. Zhan (2023). "Randomness and quantumness in space-bounded computation". Ph.D thesis, Princeton University.
6. U. Girish, R. Raz, and W. Zhan (2024). "Quantum logspace computations are verifiable", in *2024 Symposium on Simplicity in Algorithms (SOSA)*, pp. 144–150.
7. C. Dwork, and L. Stockmeyer (1992). "Finite state verifiers I: The power of interaction". *Journal of the ACM*, 39: 4, 800–828.
8. R. Freivalds and M. Karpinski (1994). "Lower space bounds for randomized computation", in *Automata, Languages and Programming. ICALP 1994*, Lecture Notes in Computer Science 820, Springer.
9. A. Ambainis and J. Watrous (2002). "Two-way finite automata with quantum and classical states". *Theoretical Computer Science*, 287: 1, 299–311.
10. A. C. Cem Say and A. Yakaryılmaz (2017). "Magic coins are useful for small-space quantum machines". *Quantum Information and Computation*, 17: 11–12, 1027–1043.
11. Z. Remscrim (2020). "The power of a single qubit: Two-way quantum finite automata and the word problem", in A. Czumaj, A. Dawar, and E. Merelli (eds), *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, vol. 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 139:1–139:18, Dagstuhl, Germany. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. https://eccc.weizmann.ac.il/report/2019/107/.

12. A. Ambainis and A. Yakaryılmaz (2021). “Automata and quantum computing”, in J.-É. Pin (ed.), *Handbook of Automata Theory*, pp. 1457–1493. Zürich, Switzerland: European Mathematical Society Publishing House.
13. A.C. Cem Say and A. Yakaryılmaz. (2014). “Finite state verifiers with constant randomness”. *Logical Methods in Computer Science*, 10: 3.
14. M. Utkan Gezer and A. C. Cem Say (2022). “Constant-space, constant-randomness verifiers with arbitrarily small error”. *Information and Computation*, 288, 104744.
15. M. Utkan Gezer, Ö. Dolu, N. Ersoy, and A. C. Cem Say (2023). “Real-time, constant-space, constant-randomness verifiers”. *Theoretical Computer Science*, 976, 114155.
16. M. Utkan Gezer and A. C. Cem Say (2024). “P has polynomial-time finite-state verifiers.” https://arxiv.org/abs/2306.09542.
17. A. Yakaryılmaz and A. C. Cem Say (2010). “Languages recognized by nondeterministic quantum finite automata”. *Quantum Information and Computation*, 10: 9, 747–770.
18. A. Yakaryılmaz and A. C. Cem Say (2010). “Succinctness of two-way probabilistic and quantum finite automata”. *Discrete Mathematics & Theoretical Computer Science*, 12: 4.
19. E. Bernstein and U. Vazirani (1997). “Quantum complexity theory”. *SIAM Journal on Computing*, 26: 1411–1473.
20. R. Freivalds (1981). “Probabilistic two-way machines”, in *Proceedings of the International Symposium on Mathematical Foundations of Computer Science*, pp. 33–45.
21. C. Dwork and L. Stockmeyer (1990). “A time complexity gap for two-way probabilistic finite-state automata”. *SIAM Journal on Computing*, 19: 6, 1011–1023.
22. M. A. Nielsen and I. L. Chuang (2002). *Quantum Computation and Quantum Information*. Cambridge: Cambridge University Press.
23. J. Watrous (2003). “On the complexity of simulating space-bounded quantum computations”. *Computational Complexity*, 12, 48–84.
24. Z. Remscrim (2021). “Lower bounds on the running time of two-way quantum finite automata and sublogarithmicspace quantum Turing machines”, in J. R. Lee (ed.), *12th Innovations in Theoretical Computer Science Conference (ITCS 2021)*, volume 185 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 39:1–39:20, Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik. https://arxiv.org/abs/2003.09877.
25. P. Turakainen (1969). “On languages representable in rational probabilistic automata”. *Annales Academiæ Scientiarum Fennicæ*, (439).
26. Ioan Macarie (1993).” Closure properties of stochastic languages”. Technical Report 441, University of Rochester.
27. Paavo Turakainen (1982). “Rational stochastic automata in formal language theory”, in *Discrete Mathematics, volume 7 of Banach Center Publications*. PWN.
28. A. Shamir (1992). “IP = PSPACE”. *Journal of the ACM*, 39: 4, 869–877.
29. M. Holzer, M. Kutrib, and A. Malcher (2011). “ Complexity of multi-head finite automata: Origins and directions”. *Theoretical Computer Science*, 412: 1–2, 83–96.
30. S. Arora and B. Barak. (2009). *Computational Complexity: A Modern Approach*. New York: Cambridge University Press.
31. J. F. Clauser, M. E. Horne, A. Shimony, and R. A. Holt (1969). “Proposed experiment to test local hidden-variable theories”. *Physical Review Letters*, 23, 880–884.
32. A. Condon and R. Ladner (1995). “Interactive proof systems with polynomially bounded strategies”. *Journal of Computer and System Sciences*, 50: 3, 506–518.
33. S. Goldwasser, Y. T. Kalai, and G. N. Rothblum (2015). “Delegating computation: Interactive proofs for muggles”. *Journal of the ACM*, 62: 4.
34. J. Shallit and Y. Breitbart. (1996). “Automaticity I: Properties of a measure of descriptional complexity”. *Journal of Computer and System Sciences*, 53, 10–25.