Electronic Colloquium on Computational Complexity, Report No. 130 (2012)

Public-qubits versus private-coins*

Abuzer Yakaryılmaz†
University of Latvia, Faculty of Computing, Raina bulv. 19, Rīga, LV-1586, Latvia
abuzer@lu.lv

October 3, 2012

Keywords: quantum complexity theory, Turing-equivalence, interactive proof systems, Arthur-Merlin games, space-bounded verifiers, finite state verifiers, NP, NEXP

Abstract

We introduce a new public quantum interactive proof system, namely qAM, by augmenting the verifier with a fixed-size quantum register in Arthur-Merlin game. We focus on space-bounded verifiers, and compare our new public system with private-coin interactive proof (IP) system in the same space bounds. We show that qAM systems not only can simulate all known space-bounded private-coin protocols, but also implements some protocols that are either not implementable by IP systems or currently not known to be simulated by IP systems. More specifically, we show that for any Turing-recognizable language, there exists a constant-space weak-qAM system (the nonmembers do not need to be rejected with high probability), and, different from the classical case, our protocol has perfect-completeness (the members are accepted exactly). For strong proof system, where the nonmembers must be rejected with high probability, we show that the known space-bounded private-coin protocols can also be simulated by qAM systems with the same space bound. In case of constant-space and log-space IP systems, the best known lower bounds are ASPACE(n) and EXP, respectively. We obtain better lower bounds for the corresponding qAM systems: Each language in NP has a constant-space (exp-time) qAM system, there is an NEXP-complete language having a constant-space qAM system, and each language in NEXP has a log-space qAM system.

1 Introduction

Interactive proof (IP) systems and Arthur-Merlin (AM) proof systems were introduced by Goldwasser, Micali, and Rackoff [13] and Babai [3], respectively. In the case of time-bounded verifiers, it was shown that the class of languages having a polynomial-time IP or AM system is identical to PSPACE [23]. In the case of space-bounded verifiers, IP systems are more powerful than AM systems for any space-bound [5, 6, 11]. e.g. the class of languages having a logarithmic-space AM system is identical to P, and the best known lower bounds for constant-space and logarithmic-space IP systems are $\mathsf{ASPACE(n)}$ and EXP, respectively. Note that $\mathsf{P} \subsetneq \mathsf{ASPACE(n)}$ and the relation between NP and $\mathsf{ASPACE(n)}$, on the other hand, is still open. It was also shown that [10] for any Turing-recognizable language, there exits a constant-space *weak*-IP system (the verifier does not need to halt with high probability for the nonmembers of the corresponding language).

*This is an extended version of some contents in [27, 25].

†The author was partially supported by FP7 FET-Open project QCS.

ISSN 1433-8092

There are many different definitions of quantum interactive proof (QIP) systems [17, 16, 24, 1, 18]. In the case of time-bounded verifiers, similar to the classical case, the class of languages having a polynomial-time QIP system was shown to be identical to PSPACE [15]. In the case of space-bounded verifiers, the only published work belongs to Nishimura and Yamakami [20]. Their results, unfortunately, are model-dependent, and so do not reflect the full power of QIP systems since they use some restricted models of quantum automata as the verifiers.

In this paper, we introduce a new proof system by augmenting the verifier with a fixed-size[1] quantum register in Arthur-Merlin game, namely *qAM*, and we focus on space-bounded verifiers. Our proof system is the first public space-bounded QIP system, and we present the first non-trivial results on space-bounded QIP systems. We show how a fixed-size quantum register leads to unexpected increase in the computational power of a public proof system, in which the prover can have complete information about the computation of the verifier. Therefore, we compare our public proof system with the private-coin IP systems. First of all, we show that there exists a constant-space weak-qAM protocol for any Turing-recognizable language. In the classical case, a similar result is known for constant-space weak private-coin protocols [10]. However, our protocol is not only public but also has perfect-completeness. Secondly, we show that for any known $s(n)$ space-bounded private-coin protocol, there exists an equivalent $s(n)$ space-bounded qAM protocol, where $s(n) \in O(1) \cup \Omega(\log(n))$ is space-constructible. Lastly, we obtain better lower bounds for qAM proof system in constant and logarithmic space bounds: Each language in NP has a constant-space qAM system that runs in exponential expected time, there is an NEXP-complete language having a constant-space qAM system, and each language in NEXP has a log-space qAM system.

2 Preliminaries

For any string x, $|x|$ is the length of x and $x[j]$ is its j^{th} symbol, where $1 \leq j \leq |x|$. "#" is the blank symbol.

We assume that the reader is familiar with deterministic, nondeterministic, and alternating Turing machines, (DTM, NTM, and ATM, respectively,) and their time- and space-bounded complexity classes $\mathcal{X}\mathsf{TIME}$ and $\mathcal{X}\mathsf{SPACE}$, where $\mathcal{X}$ is "D", "N", and "A", respectively; and, the standard time complexity classes: P, NP, EXP, and NEXP.

In the following part, we provide the necessary background, based on [11, 7], for the proof systems. For a detailed survey on space-bounded interactive proof systems, we refer the reader to [7]. An interactive proof system (IPS) consists of a prover (P) and a verifier (V). The verifier is a (resource-bounded) probabilistic Turing machine having a read-only input tape, a read/write work tape, and a source of random bits. Each head has two-way access to its tape. The states of the verifier are partitioned into reading, communication, and halting (accepting or rejecting) states, and it has a special communication cell for communicating with the prover, where the capacity of the cell is finite.

The one-step transitions of the verifier can be described as follows. When in a reading state, the verifier firstly flips an unbiased coin and then determines its next configuration based on the symbol under the tape heads, the state, and the outcome of the coin flip. When in a communication state, the verifiers writes a symbol on the communication cell with respect to the current state. Then, in response, the prover writes a symbol in the cell. Based on the state and the symbol written by prover, the verifier defines the next state of the verifier.

The prover P is specified by a prover transition function, which determines the response of the prover to the verifier based on the input and the verifier's communication history until then. Note

[1]The size of the register does not depend on the length of the input.

that this function does not need to be *computable*.

For a given input x, the probability that (P, V) accepts (rejects) x is the cumulative accepting (rejecting) probabilities taken over all branches of the verifier. The prover-verifier pair (P, V) is an IPS for $\mathtt{L}$ with error probability $\epsilon < \frac{1}{2}$ if

1. for all $x \in \mathtt{L}$, the probability that (P, V) accepts x is greater than $1 - \epsilon$,
2. for all $x \notin \mathtt{L}$, and all provers P^*, the probability that (P^*, V) rejects x is greater than $1 - \epsilon$.

These conditions are known as completeness and soundness, respectively. Now, we define some variants of IP systems by restricting and/or relaxing the above conditions.

The prover-verifier pair (P, V) is an *weak-IPS for* $\mathtt{L}$ *with error probability* $\epsilon < \frac{1}{2}$ if we relax the soundness condition (2) as follows:

$2'$. for all $x \notin \mathtt{L}$, and all provers P^*, the probability that (P^*, V) accepts x is at most ϵ.

The prover-verifier pair (P, V) is a *(weak or not) IPS having perfect completeness* for $\mathtt{L}$ if we restrict the completeness condition (1) as follows:

$1'$. for all $x \in \mathtt{L}$, the probability that (P, V) accepts x is exactly equal to 1.

An Arthur-Merlin (AM) proof system (or a public-coin IPS) is a special case of IPS such that after each coin toss, the outcome is automatically written on the communication cell, and so the prover can have complete information about the computation of the verifier.

We will use $\mathsf{IP}(\cdot)$ and $\mathsf{AM}(\cdot)$ to represent the space-bounded complexity classes for IP and AM systems, respectively. Note that the space bound is always defined on the verifiers. The ones having perfect-completeness will be shown by $\mathsf{IP_1}(\cdot)$ and $\mathsf{AM_1}(\cdot)$, respectively. We will use prefix "weak-" to represent their "weak" versions.

Some known facts related to our results on AM and IP systems are given below. (We also refer the reader to Appendix A for the details of some private-coin protocols.)

Fact 1. *[9, 5, 11] For any space-constructible $s(n) = \Omega(\log n)$,*

$$\mathsf{weak\text{-}AM(s(n))} = \mathsf{AM(s(n))} = \mathsf{ASPACE(s(n))}.$$

Moreover, $\mathsf{weak\text{-}AM(1)} \subsetneq \mathsf{weak\text{-}AM(log)} = \mathsf{AM(log)} = \mathsf{P}$.

Fact 2. *[10] Any Turing recognizable language is in* $\mathsf{weak\text{-}IP(1)}$.

Fact 3. *[11]* $\mathsf{DTIME(2^{O(n)})} = \mathsf{ASPACE(O(n))} \subseteq \mathsf{IP_1(1)}$.

Fact 4. *[5, 11] For any space-constructible $s(n) = \Omega(\log(n))$,*

$$\mathsf{DTIME(2^{2^{O(s(n))}})} = \mathsf{ASPACE(2^{O(s(n))})} \subseteq \mathsf{IP_1(s(n))}.$$

Fact 5. *[10] For any space-constructible $s(n) = \Omega(\log(n))$,*

$$\mathsf{IP_1(s(n))} \subseteq \mathsf{IP(s(n))} \subseteq \mathsf{ATIME(2^{2^{2^{O(s(n))}}})}.$$

As seen from the above facts, IP systems are more powerful than AM systems under the same space bounds. In the case of weak-soundness, the power of the IP systems with finite-state verifiers becomes Turing-equivalent, which can never be possible for a space-bounded AM system.

We will shortly show that AM systems using a fixed-size quantum register not only can simulate the known private-coin protocols (Facts 3 and 4) under the same space bounds but also implement some protocols which are not known to be simulated by IP systems under certain space bounds. Moreover, in the case of weak-soundness, our new proof system can also be Turing-equivalent even restricting to perfect-completeness, which can never be a case for a space-bounded IP systems due to the following theorem (Theorem 1).

Theorem 1. *For any space-constructible $s(n) \in \Omega(\log(n))$,*

$$\mathsf{IP_1(s(n))} \subseteq \mathsf{weak\text{-}IP_1(s(n))} \subseteq \mathsf{ASPACE(2^{2^{O(s(n))}})}.$$

Proof. See Appendix B. □

Note that Theorem 1 improves the previously known upper bound (Fact 5) for space-bounded IPS having perfect-completeness.

3 qAM

In this section, we give the definition of our new AM system, namely qAM. A qAM (*q*uantum Arthur-Merlin) proof system is an AM system where the verifier additionally has a fixed size quantum register. (The small "q" indicates that the verifier has a "very small" (possible the smallest) quantum resource.) The reading state of the new verifier is as follows:

> A superoperator, determined by the current state and the symbol(s) under the tape head(s), is applied to the quantum register, and the outcome of the operator is automatically written on the communication cell in order to satisfy *the complete information requirement*. Then, the next configuration is determined based on the the current state, the symbol(s) under the tape head(s), and the observed outcome. For any deterministic transition, the verifier applies an identity operator on the register. We refer the reader to Figure 1 for the details of superoperators.

Note that the verifier no longer needs a classical random source: For example, the superoperator $\mathcal{E} = \left\{E_{h_1} = \frac{1}{2}I, E_{h_2} = \frac{1}{2}I, E_{t_1} = \frac{1}{2}I, E_{t_2} = \frac{1}{2}I\right\}$ always produces the outcomes *head* ("h_1" or "h_2) and *tail* ("t_1" or "t_2") with probability $\frac{1}{2}$. We will use $\mathsf{qAM}(\cdot)$ and $\mathsf{qAM_1}(\cdot)$ to represent qAM counterparts of $\mathsf{AM}(\cdot)$ and $\mathsf{AM_1}(\cdot)$, respectively. Prefix "weak-" is also applicable to $\mathsf{qAM}(\cdot)$ and $\mathsf{qAM_1}(\cdot)$.

Knowledgeable readers will have noticed that, when the verifier is restricted to use constant space, the qAM system is actually the quantum counterpart of the finite automaton with both nondeterministic and probabilistic states of Condon et. al. [8], and that if we remove the communication with the prover as well we end up with a finite automaton with quantum and classical states (2qcfa) of Ambainis and Watrous [2]. Thus, in the framework of Dwork and Stocmeyer [11], $\mathsf{qAM(1)}$ corresponds to $\mathsf{AM(2qcfa)}$.

4 Main results

In our qAM protocols, we use some non-unitary transformations to implement our main tasks. We define our superoperators based on these transformations. Let $E_1, \ldots, E_k$ be some of these

The most general quantum operator is a superoperator, which generalizes stochastic and unitary operators and also includes measurement. Formally, a superoperator $\mathcal{E}$ is composed by a finite number of operation elements, $\mathcal{E} = \{E_1, \ldots, E_k\}$, satisfying that

$$\sum_{i=1}^{k} E_i^\dagger E_i = I, \tag{1}$$

where $k \in \mathbb{Z}^+$ and the indices are the measurement outcomes. When a superoperator, say $\mathcal{E}$, is applied to the quantum register in state $|\psi\rangle$, i.e. $\mathcal{E}(|\psi\rangle)$, we obtain the measurement outcome i with probability $p_i = \langle \widetilde{\psi_i} | \widetilde{\psi_i} \rangle$, where $|\widetilde{\psi_i}\rangle$, *the unconditional state vector*, is calculated as $|\widetilde{\psi_i}\rangle = E_i|\psi\rangle$ and $1 \leq i \leq k$. (Note that using unconditional state vector simplifies calculations in many cases.) If the outcome i is observed ($p_i > 0$), the new state of the system is obtained by normalizing $|\widetilde{\psi_i}\rangle$, which is $|\psi_i\rangle = \frac{|\widetilde{\psi_i}\rangle}{\sqrt{p_i}}$. Moreover, as a special operator, the quantum register can be initialized to a predefined quantum state. This initialize operator, which has only one outcome, is denoted $\acute{\mathcal{E}}$. In this paper, the entries of quantum operators are defined by rational numbers. Thus the probabilities of the outcomes are always rational numbers.

Figure 1: The details of superoperators

transformations. We can obtain a superoperator $\mathcal{E}$ based on them by defining some additional transformations $E_{k+1}, \ldots, E_{k+k'}$ such that

$$\mathcal{E} = \left\{ \frac{1}{d}E_1, \ldots, \frac{1}{d}E_k, \frac{1}{d}E_{k+1}, \ldots, \frac{1}{d}E_{k+k'} \right\}$$

satisfies Condition 1 (in Figure 1) for a convenient $d > 1$, where $k' > 0$. (We refer the reader to [26, 28] for similar procedures.) We call $\left\{\frac{1}{d}E_1, \ldots, \frac{1}{d}E_k\right\}$ *the main operation elements* and $\left\{\frac{1}{d}E_{k+1}, \ldots, \frac{1}{d}E_{k+k'}\right\}$ *the auxiliary operation elements.* Moreover, in our protocols, the computation continues on the quantum register only when the outcomes of some main operation elements are observed. On the other hand (when the outcome of an auxiliary operation element is observed), the current computation on the quantum register is always terminated/restarted with discarding the current content of the register. Therefore, the details of the auxiliary operation elements can be omitted from the description of the protocols. We use this simplification in our qAM protocols except the first protocol given in the following subsection (Section 4.1).

4.1 A constant-space exp-time qAM protocol for `KNAPSACK`

We begin with a simple constant-space (exp-time) qAM protocol for the well-known NP-complete language `KNAPSACK`, which is the collection of all strings of the form $S\$a_1\$\ldots\$a_n\$$ such that S and the a_i's are numbers in binary ($1 \leq i \leq n$), and there exists a set $I \subseteq \{1, \ldots, n\}$ satisfying $\sum_{i \in I} a_i = S$, where $n > 0$. This protocol will also be used by some other protocols given in the last subsection (Section 4.4).

Lemma 1. `KNAPSACK` $\in \mathsf{qAM}_1(1)$*, where the protocol runs in exponential expected time.*

Proof. We assume that the input to be of the form $S\$a_1\$\ldots\$a_n\$$, where S, the a_i's are numbers in binary ($1 \leq i \leq n$), and $n > 0$. (If not, the input is immediately rejected.) The input is written between two # symbols on the input tape and its head is not allowed to cross these boundaries.

The main idea is that the verifier scans the input from left to right in an infinite loop and firstly encodes S, and then subtracts the encoding of each of the a_i's selected by the prover, in some amplitudes of the states on the quantum register. And, at the end of the loop (round), the verifier tests whether the result is zero or not (described later). Since our encoding procedure works by reducing the amplitude with a constant in each step, the process can successfully be ended with

$$\begin{array}{ll}
\text{Part 2:} & \mathcal{E}_0 = \left\{ E_f = \frac{1}{3}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}, E_{i_1} = \frac{1}{3}\begin{pmatrix} 2 & 0 & -2 \\ 2 & 0 & 2 \\ 0 & 2 & 0 \end{pmatrix}, E_{i_2} = \frac{1}{3}\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \right\} \\
& \mathcal{E}_1 = \left\{ E_f = \frac{1}{3}\begin{pmatrix} 1 & 0 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}, E_{i_1} = \frac{1}{3}\begin{pmatrix} 2 & -1 & 0 \\ 1 & 0 & 2 \\ 1 & 0 & -2 \end{pmatrix}, E_{i_2} = \frac{1}{3}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix} \right\} \\
& \mathcal{E}_\$ = \left\{ E_f = \frac{1}{3}\mathcal{I}, E_{i_1} = \frac{1}{3}2\mathcal{I}, E_{i_2} = \frac{1}{3}2\mathcal{I} \right\} \\
\text{Part 3:} & \mathcal{E}'_0 = \left\{ E_f = \frac{1}{3}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{pmatrix}, E_{i_1} = \frac{1}{3}\begin{pmatrix} 2 & 2 & 0 \\ 2 & -2 & 0 \\ 0 & 0 & 2 \end{pmatrix}, E_{i_2} = \frac{1}{3}\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \right\} \\
& \mathcal{E}'_1 = \left\{ E_f = \frac{1}{3}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 2 \end{pmatrix}, E_{i_1} = \frac{1}{3}\begin{pmatrix} 2 & 0 & -1 \\ 1 & 2 & 0 \\ 1 & -2 & 0 \end{pmatrix}, E_{i_2} = \frac{1}{3}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{pmatrix} \right\} \\
\text{Part 4:} & \mathcal{E}'_\$ = \left\{ E_f = \frac{1}{3}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{pmatrix}, E_{i_1} = \frac{1}{3}\begin{pmatrix} 0 & -1 & 1 \\ 2 & 1 & -1 \\ 2 & -1 & 1 \end{pmatrix}, E_{i_2} = \frac{1}{3}\begin{pmatrix} 0 & 2 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, E_{i_3} = \frac{1}{3}\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \right\} \\
& \mathcal{E}''_\$ = \left\{ E_f = \frac{1}{3}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, E_{i_1} = \frac{1}{3}\begin{pmatrix} 2 & -2 & 0 \\ 2 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix} \right\} \\
\text{Part 5:} & \mathcal{E}_\# = \left\{ E_a = \frac{1}{3}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, E_r = \frac{1}{3}\begin{pmatrix} 0 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{pmatrix}, E_{i_1} = \frac{1}{3}\begin{pmatrix} 2 & 0 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & 3 \end{pmatrix} \right\}
\end{array}$$

Figure 2: The details of superoperators used by the qAM protocol for `KNAPSACK`

an exponentially small probability depending on the length of the input. Therefore, a new round is initiated with remaining huge probability.

The quantum register has 3 states, $\{q_1, q_2, q_3\}$. Each round is composed by five parts, described below. The details of superoperators applied in each round are given in Figure 2. The actions associated to each outcome are as follows: (i) The input head is moved forward if outcome "f" is observed, (ii) the input is accepted (rejected) if outcome "a" ("r") is observed, and (iii) a new round is initiated if outcome "i_j" ($1 \leq j \leq 3$) is observed.

1. The finite register is initialized on symbol #: $|\psi_0\rangle = (1 \;\; 0 \;\; 0)^T$.
2. S is encoded into the amplitudes of $|q_2\rangle$: $\mathcal{E}_\sigma$ is applied to the quantum register when reading $\sigma \in \{0,1\}$. Then, $\mathcal{E}_\$$ is applied to the quantum register when reading \$.
3. Each a_i ($1 \leq i \leq n$) is encoded into the amplitude of $|q_3\rangle$: $\mathcal{E}'_\sigma$ is applied to the quantum register when reading $\sigma \in \{0,1\}$.
4. If an a_i ($1 \leq i \leq n$) is *selected* by the prover on symbol \$, it is subtracted from the number represented by the amplitude of $|q_2\rangle$: $\mathcal{E}'{}_\$$ is applied to the quantum register. If it is not selected, $\mathcal{E}''_\$$ is applied to the quantum register. Note that, the amplitude of $|q_3\rangle$ is set to 0 after each of these transformations.
5. The decision is given on the right end-marker #: $\mathcal{E}_\#$ is applied to the quantum register when reading the right end-marker #.

Let x be the input and T be the cumulative sum of selected a_i's by the prover. Then, the state of the register before reading the right end-marker # becomes

$$|\widetilde{\psi_{|x|}}\rangle = \left(\frac{1}{3}\right)^{|x|} \begin{pmatrix} 1 \\ S - T \\ 0 \end{pmatrix}.$$

After applying $\mathcal{E}_\#$, the input is rejected with probability

$$\left(\frac{1}{3}\right)^{2|x|+2} (3S - 3T)^2,$$

which is at least $9\left(\frac{1}{3}\right)^{2|x|+2}$ if $S \neq T$, and is exactly equal to 0 if $S = T$. On the other hand, the input is always accepted with probability $\left(\frac{1}{3}\right)^{2|x|+2}$. Therefore, if $x \in$ `KNAPSACK`, there exists a prover such that it is accepted exactly, and if $x \notin$ `KNAPSACK`, whatever the prover says, it is rejected with a probability at least $\frac{9}{10}$. The error bound can be reduced to any desired value by using conventional probability amplification techniques. Since the protocol is always terminated with a probability at least $\left(\frac{1}{3}\right)^{2|x|+2}$ in each round, the expected running time is $2^{O(n)}$. $\square$

4.2 Constant-space weak-qAM systems are Turing-equivalent

We continue with a constant-space weak-qAM protocol having perfect completeness for any given Turing-recognizable language. We present our result by giving *a new public protocol* simulating a given DTM. Contrary to the classical case, our simulation technique can also be applicable to the case of strong-soundness. *In classical case, to show universality of constant-space weak-IP systems, a simulation of two-way finite automaton with two-counters was given [10]. Since the complexity classes are defined by Turing machines, this technique does not seem to be applicable to the case of strong-soundness. (See also Appendix A).* Therefore, after making certain modifications (and combining with the qAM protocol for `KNAPSACK` in some cases), we obtain some other space-bounded qAM protocols. Note that, all of our qAM protocols have *perfect-completeness.* (Remember that two-sided bounded error is necessary for the universality of constant-space weak-IP systems due to Theorem 1.)

Theorem 2. *Any Turing recognizable language is in* $\mathsf{weak\text{-}qAM}_1(1)$.

Proof. Let `L` be a Turing recognizable language and $\mathcal{D}$ be a single-tape DTM recognizing `L`. We will construct a weak-qAM proof system (P, V) for `L` with perfect completeness, where V is a finite state verifier.

We begin with some details of $\mathcal{D}$. Q containing q_1 (the initial state), q_a (the accepting state), and q_r (the rejecting state) is the set of states, Γ containing $\#$ is the tape alphabet, and $\Gamma' = Q \cup \Gamma$, called configuration alphabet, where Q and Γ are disjoint sets and $\$ \notin \Gamma$. Note that Γ' contains at least 5 elements. Any configuration of $\mathcal{D}$ is of the form uqv ($\mathcal{D}$ is in q and the tape head is on the leftmost symbol of v), where $q \in Q$ and $uv \in \#(\Gamma)^*\#$. The unnecessary blank symbols are always dropped from the descriptions of configurations. For a given input string x, the initial configuration is represented as $q_1\#x\#$.

The main protocol is executed in an infinite loop and each iteration (round) is composed by the following: (i) The verifier requests the computation (a sequence of configurations starting from the initial configuration) of $\mathcal{D}$ on the given input, say x, from the prover. (ii) Against the cheating provers, the verifier checks the correctness of the computation and rejects x if it detects a defect in the computation. (iii) When it encounters a halting configuration, the verifier mimics the decision of this (halting) configuration.

Let w be the string obtained from the prover in a single round. The verifier expects w as $c_1\$\$c_2\$\$c_3\$\$\cdots$, where (P1) c_i's ($i > 0$) are some configurations of $\mathcal{D}$, (P2) c_1 is the initial configuration, and (P3) c_{i+1} is the successor of c_i in one step for any $i > 0$. (We use double \$ for pedagogical reasons.) Note that w can be an infinite string. The verifier can check P1 and P2 deterministically, and x is rejected immediately if one of them fails. Therefore, in the following part, we assume that w satisfies both P1 and P2 and any configuration separator is always "\$\$".

The non-trivial part is to check P3 for each $i > 0$, i.e. whether c_{i+1} is identical to $\mathtt{next}(c_i)$, where $\mathtt{next}(c_i)$ is the single-step successor of c_i. This is where the quantum register comes into play. The idea behind is to encode $\mathtt{next}(c_i)$ and c_{i+1} into the amplitudes of two states on the register, and then to subtract them, and to reject x with the resulting amplitude.[2] We call this procedure *successor-check*. The i^{th} successor-check compares $\texttt{next}(c_i)$ and c_{i+1}. As will be detailed below, whenever $c_{i+1} = \texttt{next}(c_i)$ (for all defined $i > 0$), the input is never rejected by a successor-check. Thus, once a halting configuration is obtained from the prover, the *only decision* is made based on it. Otherwise, i.e. $\exists i > 0$ s.t. $c_{i+1} \neq \texttt{next}(c_i)$, x is rejected by the i^{th} successor-check. We show that such a reject probability is sufficiently greater than any accept probability in a single round.

In the remaining part, we will give the details of a single round and the analyses of the protocol. The verifier requests the computation of $\mathcal{D}$ on x symbol by symbol from the prover, and it uses a 3-symbol buffer to parallelly encode the legal successor of the presently scanned configuration. The verifier uses a 4-state quantum register, $q_1, \ldots, q_4$, which is set to $|\psi_{1,0}\rangle = (1 \quad 0 \quad 0 \quad 0)^T$ at the beginning of each round. The configurations are encoded in base-m, where $m = |\Gamma'| + 1$. Each symbol of Γ' is associated with a different positive integer. Thus, any configuration c_i $(i > 0)$ can be represented by a $|c_i|$-length number in base-m. We use the same symbol for both the encoded string/symbol and its encoding. The verifier applies one superoperator per symbol. When the outcome of an auxiliary operation elements is observed, the verifier terminates the current round and initiates a new round. The tasks implemented by the main operation elements reduce the amplitudes with $\frac{1}{d} < 1$. Therefore, after applying each superoperator, a new round is initiated with some probability. In other words, a round can continue only with a small probability. The complete details of the superoperators and the related operations are given at the end of the proof due to their technicalities.

Let l_i be the length of $c_1\$\$c_2\$\$\cdots c_i\$\$$ $(i > 0)$. By processing $c_1\$\$$, i.e. a series of superoperators $\mathcal{E}_{1,1}, \ldots, \mathcal{E}_{1,|c_1\$\$|}$ are applied to the register, $\texttt{next}(c_1)$ is encoded into the amplitudes of $|q_2\rangle$. Then, the quantum state becomes

$$|\widetilde{\psi_{2,0}}\rangle = \left(\frac{1}{d}\right)^{l_1} \begin{pmatrix} 1 \\ \texttt{next}(c_1) \\ 0 \\ 0 \end{pmatrix}.$$

Note that unconditional state vectors facilitate the calculations since the probabilities can be calculated directly. Similarly, by processing $c_2\$$, c_2 and $\texttt{next}(c_2)$ are encoded into the amplitudes of $|q_3\rangle$ and $|q_4\rangle$, respectively:

$$|\widetilde{\psi_{2,|c_2\$|}}\rangle = \left(\frac{1}{d}\right)^{l_2-1} \begin{pmatrix} 1 \\ \texttt{next}(c_1) \\ c_2 \\ \texttt{next}(c_2) \end{pmatrix}.$$

After processing one more \$, the first successor-check is finalized: The corresponding superoperator has two main operation elements. The first one is responsible for comparing $\texttt{next}(c_1)$ and c_2, and subtracts the amplitudes of $|q_2\rangle$ and $|q_3\rangle$. Its outcome is observed with probability

$$\left(\frac{1}{d}\right)^{2l_2} (\texttt{next}(c_1) - c_2)^2,$$

which is also the rejecting probability of the first successor-check. The second main operation element is determined conditionally. If $\texttt{next}(c_2)$ is an accepting (a rejecting) configuration, then it selects only the amplitude of $|q_1\rangle$, the outcome of which is observed with probability $\left(\frac{1}{d}\right)^{2l_2}$. This

[2] In fact, the encoding part can also be implemented by a probabilistic system but the aforementioned *subtraction* is not possible in classical systems!

probability is also the accepting (rejecting) probability of the round. Since the computation is terminated due to the outcomes of both operation elements, the round does not continue in this case. If $\texttt{next}(c_2)$ is not a halting configuration, the round continues with the next successor-check. Thus the quantum state becomes

$$|\widetilde{\psi_{3,0}}\rangle = \left(\frac{1}{d}\right)^{l_2} \begin{pmatrix} 1 \\ \texttt{next}(c_2) \\ 0 \\ 0 \end{pmatrix}.$$

One can easily verify that if the prover is cheating about c_2, the input is rejected with a probability at least $m^2 \left(\frac{1}{d}\right)^{2l_2}$ since both $\texttt{next}(c_1)$ and c_2 end with a # and they must be disagree on at least one digit. If the prover is honest, the decision is given only if $\texttt{next}(c_2)$ is a halting configuration, whose probability is at least m^2 smaller than the above rejecting probability. The other successor-checks are executed exactly in the same way (check the end of the proof).

The analysis of the protocol is as follows. If $x \in L$, then P always sends the correct computation of $\mathcal{D}$ on x to V, say $c_1\$\$c_2\$\$\cdots\$\$c_t\$\$$ such that $\texttt{next}(c_t)$ is an accepting configuration. Then V never rejects but accepts x with probability $\left(\frac{1}{d}\right)^{2l_t}$ in each round. So, x is accepted exactly by V.

If $x \notin L$, then the only case in which V accepts x is that P^* send a computation of some nonhalting configurations $c_1\$\$c_2\$\$\cdots\$\$c_{t'}\$\$$ such that $\texttt{next}(c_{t'})$ is an accepting configuration, where $t' > 1$. Since $x \notin L$, there must be an i ($1 \leq i < t'$) such that $\texttt{next}(c_i) \neq c_{i+1}$. Therefore, x is rejected with probability at least m^2 times greater than the accepting probability in a single round. In other words, the overall accepting probability can be bounded above by $\frac{1}{m^2+1}$. This bound can be easily reduced to any desired value by selecting a greater m value.

Now, we give the omitted details of the superoperators and the related operations below. Remember that l_i is the length of $c_1\$\$c_2\$\$\cdots c_i\$\$$ and the quantum state is set to

$$|\psi_{1,0}\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

at the beginning of each round. As described before, we omit the details of each auxiliary operation element, and a new round is initiated when the outcome of such an operation element is observed.

For each symbol of $w_1 = c_1\$\$$, the verifier applies $|w_1|$ superoperators, $\mathcal{E}_{1,1}, \ldots, \mathcal{E}_{1,|w_1|}$, to the register, some of them can be the same. The aim is to encode $\texttt{next}(c_1)$ into the amplitudes of $|q_2\rangle$. For each $j \in \{1, \ldots, |c_1| - 1\}$, the main operation element of $\mathcal{E}_{1,j}$ is as follows:

$$\frac{1}{d} \begin{pmatrix} 1 & 0 & 0 & 0 \\ \texttt{next}(c_1)[j] & m & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

For $\mathcal{E}_{1,|c_1|}$ and $\mathcal{E}_{1,|c_1\$|}$, we have the following cases, each of which can be deterministically determined and handled by using 3-symbol buffer:

- If $|\texttt{next}(c_1)| = |c_1| - 1$, the main operation elements of $\mathcal{E}_{1,|c_1|}$ and $\mathcal{E}_{1,|c_1\$|}$ are as follows:

$$\frac{1}{d} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \text{ and } \frac{1}{d} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

respectively, since the encoding of $\texttt{next}(c_1)$ is finished by superoperator $\mathcal{E}_{1,|c_1|-1}$.

- If $|\texttt{next}(c_1)| = |c_1|$, the main operation elements of $\mathcal{E}_{1,|c_1|}$ and $\mathcal{E}_{1,|c_1\$|}$ are as follows:

$$\frac{1}{d}\begin{pmatrix} 1 & 0 & 0 & 0 \\ \# & m & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \text{ and } \frac{1}{d}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

  respectively, since the encoding of $\texttt{next}(c_1)$ is finished by superoperator $\mathcal{E}_{1,|c_1|}$. (Note that the last symbol of any configuration is a blank symbol.)

- If $|\texttt{next}(c_1)| = |c_1| + 1$, the main operation elements of $\mathcal{E}_{1,|c_1|}$ and $\mathcal{E}_{1,|c_1\$|}$ are as follows:

$$\frac{1}{d}\begin{pmatrix} 1 & 0 & 0 & 0 \\ \texttt{next}(c_1)[|c_1|] & m & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \text{ and } \frac{1}{d}\begin{pmatrix} 1 & 0 & 0 & 0 \\ \# & m & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

  respectively, since the encoding of $\texttt{next}(c_1)$ is finished by superoperator $\mathcal{E}_{1,|c_1|+1}$.

The main operation elements of $\mathcal{E}_{1,|c_1\$\$|}$ is as follows:

$$\frac{1}{d}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Thus, after applying superoperators $\mathcal{E}_{1,j}$'s ($1 \leq j \leq |w_1|$), the state vector of the register becomes

$$|\widetilde{\psi_{2,0}}\rangle = \left(\frac{1}{d}\right)^{l_1} \begin{pmatrix} 1 \\ \texttt{next}(c_1) \\ 0 \\ 0 \end{pmatrix}. \tag{2}$$

We continue with the block $w_2 = c_2\$\$$. In fact, the tasks implemented in this part are the same for any other block $w_i = c_i\$\$$ ($i > 2$). Similar to the above case, for each symbol of w_2, the verifier applies $|w_2|$ superoperators, $\mathcal{E}_{2,1}, \ldots, \mathcal{E}_{2,|w_2|}$, to the register, some of which can be the same. Remember that before starting to apply any operator, the unconditional state vector is $|\widetilde{\psi_{2,0}}\rangle$ (Equation 2). The aims in this block (w_2) are as follows:

1. By processing $c_2\$$,
   (a) to encode c_2 into the amplitudes of $|q_3\rangle$ and
   (b) to encode $\texttt{next}(c_2)$ into the amplitudes of $|q_4\rangle$.
2. By processing the second \$,
   (a) to finalize the 1st successor-check,
   (b) to accept or reject the input if $\texttt{next}(c_2)$ is an accepting or a rejecting configuration, respectively; and, to start 3rd successor-check if $\texttt{next}(c_2)$ is not a halting configuration.

The details of superoperators to encode c_2 and $\texttt{next}(c_2)$ are similar to the ones given above. For each $j \in \{1, \ldots, |c_2| - 1\}$, the main operation element of $\mathcal{E}_{2,j}$ is as follows:

$$\frac{1}{d}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ c_2[j] & 0 & m & 0 \\ \texttt{next}(c_2)[j] & 0 & 0 & m \end{pmatrix}.$$

For $\mathcal{E}_{2,|c_2|}$ and $\mathcal{E}_{2,|c_2\$|}$, we have the following cases:

- If $|\texttt{next}(c_2)| = |c_2| - 1$, the main operation elements of $\mathcal{E}_{2,|c_2|}$ and $\mathcal{E}_{2,|c_2\$|}$ are as follows:

$$\frac{1}{d}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \# & 0 & m & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \text{ and } \frac{1}{d}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

  respectively, since the encoding of $\texttt{next}(c_2)$ is finished by the superoperator $\mathcal{E}_{2,|c_2|-1}$.

- If $|\texttt{next}(c_2)| = |c_2|$, the main operation elements of $\mathcal{E}_{2,|c_2|}$ and $\mathcal{E}_{2,|c_2\$|}$ are as follows:

$$\frac{1}{d}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \# & 0 & m & 0 \\ \# & 0 & 0 & m \end{pmatrix} \text{ and } \frac{1}{d}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

  respectively, since the encoding of $\texttt{next}(c_2)$ is finished by the superoperator $\mathcal{E}_{2,|c_2|}$.

- If $|\texttt{next}(c_2)| = |c_2| + 1$, the main operation elements of $\mathcal{E}_{2,|c_2|}$ and $\mathcal{E}_{2,|c_2\$|}$ are as follows:

$$\frac{1}{d}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \# & 0 & m & 0 \\ \texttt{next}(c_2)[|c_2|] & 0 & 0 & m \end{pmatrix} \text{ and } \frac{1}{d}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \# & 0 & 0 & m \end{pmatrix},$$

  respectively, since the encoding of $\texttt{next}(c_2)$ is finished by superoperator $\mathcal{E}_{2,|c_2|+1}$.

Thus, before applying $\mathcal{E}_{2,|c_2\$\$|}$, the state vector becomes as follows:

$$|\widetilde{\psi_{2,|c_2\$|}}\rangle = \left(\frac{1}{d}\right)^{l_2-1}\begin{pmatrix} 1 \\ \texttt{next}(c_1) \\ c_2 \\ \texttt{next}(c_2) \end{pmatrix}. \qquad (3)$$

Operator $\mathcal{E}_{2,|c_2\$\$|}$ has two main operation elements. This first one is responsible to finalize the 1th successor-check:

$$\frac{1}{d}\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

The associated action of this operation element is *to reject the input.* Therefore, the input is rejected with probability

$$\left(\frac{1}{d}\right)^{2l_2} (\texttt{next}(c_1) - c_2)^2 ,$$

which is zero if the check succeeds ($\texttt{next}(c_1) = c_2$) and is at least

$$\left(\frac{1}{d}\right)^{2l_2} m^2$$

if the check fails ($\texttt{next}(c_1) \neq c_2$). Since the last symbol of $\texttt{next}(c_1)$ and c_2 are the identical, the value of $|\texttt{next}(c_1) - c_2|$ can be at least m.

The second main operation element is determined by the type of $\texttt{next}(c_2)$:

- If it is an accepting (a rejecting) configuration, then the following operation element is applied:

$$\frac{1}{d}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

  The associated action of this is *to accept (reject) the input.* Therefore, the input is accepted (rejected) with probability

$$\left(\frac{1}{d}\right)^{2l_2}.$$

  Note that the round is certainly terminated in this case.

- If it is not a halting configuration, then the following operation element is applied:

$$\frac{1}{d}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

  The associated action of this is to continue the round, and so *to initiate* the 3^{rd} successor-check. The state vector becomes

$$|\widetilde{\psi_{3,0}}\rangle = \left(\frac{1}{d}\right)^{l_2} \begin{pmatrix} 1 \\ \texttt{next}(c_2) \\ 0 \\ 0 \end{pmatrix}.$$

Note that if the prover never sends \$\$ symbols, then no decision is given.

The tasks for block $w_3 = c_3\$\$$ is exactly the same as for block $w_2 = c_2\$\$$, and the tasks for block $w_4 = c_4\$\$$ is exactly the same as for block $w_3 = c_3\$\$$, and so on. Therefore we can generalize it for a generic block $w_i = c_i\$\$$, where $i \geq 2$. The state vector is

$$|\widetilde{\psi_{i,0}}\rangle = \left(\frac{1}{d}\right)^{l_{i-1}} \begin{pmatrix} 1 \\ \texttt{next}(c_{i-1}) \\ 0 \\ 0 \end{pmatrix}$$

at the beginning. The aims are as follows:

1. By processing $c_i\$$,
   (a) to encode c_i into the amplitudes of $|q_3\rangle$ and
   (b) to encode $\texttt{next}(c_i)$ into the amplitudes of $|q_4\rangle$.
2. By processing the second \$,
   (a) to finalize the $(i-1)^{st}$ successor-check,
   (b) to accept or reject the input if $\texttt{next}(c_i)$ is an accepting or a rejecting configuration, respectively; and, to start the $(i+1)^{st}$ successor-check if $\texttt{next}(c_i)$ is not a halting configuration.

When the $(i-1)^{st}$ successor-check is finalized, the input is rejected with probability

$$\left(\frac{1}{d}\right)^{2l_i} (\texttt{next}(c_{i-1}) - c_i)^2,$$

which can be at least

$$\left(\frac{1}{d}\right)^{2l_i} m^2$$

if $\texttt{next}(c_{i-1}) \neq c_i$. If $\texttt{next}(c_i)$ is an accepting (a rejecting) configuration, then the input is accepted (rejected) with probability

$$\left(\frac{1}{d}\right)^{2l_i}.$$

Otherwise, the $(i+1)^{st}$ successor-check is initialized with the state vector

$$\widetilde{|\psi_{i,0}\rangle} = \left(\frac{1}{d}\right)^{l_i} \begin{pmatrix} 1 \\ \texttt{next}(c_i) \\ 0 \\ 0 \end{pmatrix}.$$

□

One of the remarkable properties of our protocol is that the quantum register can be in a *superposition* of two successor-checks, which is one of the fundamental and distinctive properties of quantum computation. In fact, private-coin protocols can also "imitate" this phenomenon: The verifier privately selects odd- or even-numbered successor-checks, and so, *from the viewpoint of the prover*, the verifier seems to be in a superposition of two successor-checks (such as, with probability $\frac{1}{2}$). However, in our protocol, the verifier *really* is in a superposition and it is independent from any prover. This is indeed why our protocol works in a public setting. Therefore, our protocol can also be seen as a new and elegant evidence on how the superposition phenomenon can become useful in terms of complexity theory.

The main corollary of Theorem 2 is that weak-qAM systems can be more powerful than weak-IP systems:

Corollary 1. *For any space bound $s(n)$,* $\mathsf{weak\text{-}IP_1(s(n))} \subsetneq \mathsf{weak\text{-}qAM_1(1)}$.

4.3 qAM systems can simulate the known private-coin protocols

Now, we back to the protocols in which the computation always halts with high probability. (Note that, our weak-protocol may run forever in some cases, i.e., the simulated machine may run forever on some inputs, a cheating prover never sends $$ after sending a few valid configurations, etc.) In our weak-protocol, the input head of the verifier is never used after checking the first configuration. In fact, it can be used to check whether the length of a configuration sent by the prover is linear. Thus, the verifier can force the prover to send linear-size configurations. So, it can be easily obtained that $\mathsf{DSPACE(n)} \subseteq \mathsf{qAM_1(1)}$, i.e. the prover sends the computation of a linear-space DTM.

Lemma 2. $\mathsf{DSPACE(n)} \subseteq \mathsf{qAM_1(1)}$.

If the prover additionally provides nondeterministic choices, this result is extended to $\mathsf{NSPACE(n)} \subseteq \mathsf{qAM_1(1)}$, i.e. the prover sends the computation of a linear-space NTM, in which nondeterministic choices are determined by the prover. Although it is not trivial as these two results, we can go one step further: $\mathsf{ASPACE(n)} \subseteq \mathsf{qAM_1(1)}$, i.e. the prover sends the computation of a linear-space ATM, in which nondeterministic choices are determined by the prover and universal choices are determined by the verifier. This idea was firstly given by Reif [22] for the simulation of a space-bounded ATM by a private ATM, and then used by Condon and Ladner [9], Condon [5], and Dwork and Stockmeyer [11] for some other similar simulations. We embed the simulation idea of Dwork and Stockmeyer [11] into our weak-protocol by making some certain modifications.

Theorem 3. $\mathsf{DTIME(2^{O(n)})} = \mathsf{ASPACE(n)} \subseteq \mathsf{qAM_1(1)}$.

Proof. Let $\mathtt{L}$ be a language in $\mathsf{ASPACE(n)}$. Then, there exists a single-tape ATM $\mathcal{A}$ recognizing $\mathtt{L}$. The components of $\mathcal{A}$ is similar to $\mathcal{D}$ given in the proof of Theorem 2. (Note that, for an ATM, any nonhalting state is labelled as either existential or universal.) We make some additional assumptions on $\mathcal{A}$ as given in Dwork and Stockmeyer [11] (see also Appendix A). Tape alphabet Γ, apart from $\#$, contains another special symbol ¢. The input, say x, is given as ¢x¢, ¢ is only overwritten by ¢, the head is not allowed to leave the area between the cent symbols, and any non-cent symbol is only overwritten by a non-cent symbol. Thus, any configuration of $\mathcal{A}$ is of the form uqv such that $q \in Q$ and $uv \in ¢(\Gamma \setminus \{¢\})^{|x|}¢$. We assume that $\mathcal{A}$ makes an existential move after a universal one and vice versa. Moreover, each such a transition leads to exactly two branches. In order to fix the running time in each branch, we assume that $\mathcal{A}$ has some additional deterministic states (i.e. universal states with no branching) to keep a counter to guarantee that the decision is given after making exactly $2^{c|x|}$ branching transitions. So, we can group non-halting states of $\mathcal{A}$ into three groups:

1. existential states,
2. universal states leading to two transitions, and
3. universal states leading to one transition.

The third ones are called deterministic states to prevent confusion. So, only the second ones are called universal states. We also assume that each counter operation takes the same amount of steps, which is only dependent on the length of input ($|x|$). So the computation tree of $\mathcal{A}'$ on x, denoted by $\mathcal{T}_{\mathcal{A}}(x)$, has $2^{2^{c|x|}}$ leafs, and each path from the root to a leaf has the same depth, where c is an appropriate constant. If the leaf is an accepting (rejecting) configuration, then the path is called accepting (rejecting) path.

Now, we will describe a qAM proof system (P, V) for `L` with perfect completeness based the qAM system given in the proof of Theorem 2 after making some modifications, where V is a finite state verifier.

The first modification is that the verifier requests a computation path of $\mathcal{T}_{\mathcal{A}}(x)$, i.e. a path from the root to one of the leafs, from the prover in each round. The format of the computation is the same:

$$c_1\$\$c_2\$\$\cdots c_i\$\$c_{i+1}\$\$\cdots.$$

However, before a successor-check, say the i^{th} one, the verifier should know which branch it follows after c_i since the verifier encodes $\texttt{next}(c_i)$ during processing $c_i\$\$$. Therefore, the verifier should know the follow-up branch before obtaining $c_i\$\$$. Similarly, the prover should be aware of this branch before sending c_{i+1}. As mentioned earlier, any existential branch is determined by the prover and any universal branch is determined by the verifier. One of the convenient ways is that before communicating on $c_i\$\$$, both parties exchange their decisions *about which branch is followed after* c_i. Remark that if c_i is a deterministic branch, then both choices become useless, if it is an existential (a universal) one, the choice of the prover (the verifier) becomes useless. The verifier makes its choice with equal probability. Therefore, it applies a superoperator having the following two main operation elements

$$\left\{ \underbrace{\frac{1}{d}I}_{l}, \underbrace{\frac{1}{d}I}_{r} \right\},$$

where outcome "l" ("r") represents the left-branch (right-branch) and I is the identity operator.

The second modification is that each configuration length is *deterministically* checked by the verifier to be equal to $|x| + 2$ by help of the input head. We denote this property as P4. If not, the computation is terminated and the input is rejected immediately.

The third modification is about the acceptance probability, which will be dramatically smaller than the accepting probability in the original protocol. We describe why the original strategy does not work with an example.

> Remember that if a round ends with an accepting (a rejecting) configuration in the original protocol, then the input is accepted (rejected) with a probability calculated by the amplitude of $|q_1\rangle$. Suppose that the prover sends a computation of $\mathcal{A}$ satisfying P1-P4. Then if the verifier follows the original strategy, the input is rejected (accepted) with the same probability at the end of each round since the length of each path is equal in this case. If the prover follows a nondeterministic strategy of $\mathcal{A}$ that leads to exactly one rejecting leaf, then the input is accepted with a high probability although the input must be rejected with high probability due to this subtree.

Our new acceptance strategy is as follows. The verifier uses an additional state, q_5, on the register. When a new round is initiated, the state vector is immediately set to

$$\left(\frac{1}{d}\right)\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

Then, the amplitude of $|q_5\rangle$ is multiplied by

$$\left(\frac{1}{2d}\right)$$

for each configuration in the computation. The input is accepted similar to the original protocol but by the amplitudes of $|q_5\rangle$. Note that after making b branches, the ratio of the amplitude of $|q_1\rangle$ to the amplitude of $|q_5\rangle$ is 2^b, and so, any rejecting probability calculated by the amplitude of $|q_1\rangle$ is 4^b times greater than any accepting probability calculated by the amplitude of $|q_1\rangle$.

Now, we can analyse our modified protocol. If $x \in L$, then (honest) P always sends a valid computation of $\mathcal{A}$ on x, and the input is always accepted with a (constant) non-zero probability in each round. Therefore, x is accepted by V exactly.

If $x \notin L$, there exists always at least one rejecting path whichever nondeterministic strategy is followed. As known form the original protocol, if the prover (P^*) sends some configurations violating any of P1-P4, then the rejecting probability is always sufficiently greater than the accepting one in a single round. Therefore, suppose that the prover sends the configurations satisfying P1-P4. Let p be the probability of rejecting x at the end of a rejecting path. So, the accepting probability of x at the end of an accepting path is equal to

$$p(\frac{1}{4})^{2^{c|x|}}.$$

Since there can be at most $2^{2^{c|x|}} - 1$ accepting paths, a single rejecting probability is sufficiently greater than the overall accepting probability. $\square$

Theorem 4. *For any space-constructible $s(n) \in \Omega(\log(n))$,*

$$\mathsf{DTIME}(2^{2^{\mathsf{O(s(n))}}}) = \mathsf{ASPACE}(2^{\mathsf{O(s(n))}}) \subseteq \mathsf{qAM_1(s(n))}.$$

Proof. In this case, the verifier can force the prover to send $2^{O(s(n))}$-size configurations by using its classical work tape. The remainder follows from the proof of Theorem 3. $\square$

Due to Fact 1 and Theorem 4, we can follow that qAM systems (having perfect-completeness) are *strictly* more powerful than any AM system under the same space bound.

Corollary 2. *For any space bound $s(n)$,* $\mathsf{AM(s(n))} \subseteq \mathsf{weak\text{-}AM(s(n))} \subsetneq \mathsf{qAM_1(s(n))}$.

4.4 Better lower bounds for qAM systems

In this section, we use the idea given at the beginning of Section 4.3 in a different way. That is, similar to the simulation of linear-space DTMs, constant-space qAM systems can also simulate linear-space (deterministic) reductions. Thus, by combining this kind of simulation with the qAM protocol for `KNAPSACK`, we can obtain the following interesting result.

Theorem 5. *Any language linear-space reducible to* `KNAPSACK` *is in* $\mathsf{qAM_1(1)}$. *Moreover, if the reduction takes $t(n)$-steps, the expected running time of the protocol is $2^{O(n \cdot t(n))}$.*

Proof. The proof combines Lemma 1 and Lemma 2. Let `L` be a language linear-space reducible to `KNAPSACK`. Then, there exists a linear-space DTM $\mathcal{D}$ that outputs a string u for a given input string, say x, such that $x \in$ `L` iff $u \in$ `KNAPSACK`.

We can design a constant-space qAM protocol for `L` as follows. Similar to previous protocols, the main protocol is executed in an infinite loop. The prover sends the computation of $\mathcal{D}$ on x to the verifier, and also says "selected" or "not selected" after each \$ symbol of the output string u (except the first \$ symbol). The verifier runs two protocols in parallel. The first one checks the correctness of the computation of $\mathcal{D}$, and rejects the input if it detects a defect in the computation. (This protocol never accepts the input.) The second one simulates the qAM protocol for `KNAPSACK` on the output string (see Lemma 1). If $x \in \mathtt{L}$, the first protocol never rejects the input and the second protocol always accepts the input with some probability in each round by the help of the (honest) prover. If $x \notin \mathtt{L}$, we have two main cases. If the prover lies about the computation of $\mathcal{D}$ on x in a single round, then the first protocol rejects the input with some probability, which can be sufficiently bigger than any accept probability given by the second protocol. If the prover does not lie, then the second protocol rejects the input with some probability, which is certainly bigger than any accepting probability. Since the maximum length of a valid computation is at most $O(n \cdot t(n))$, the computation is always terminated with a probability at least $\left(\frac{1}{2}\right)^{O(n \cdot t(n))}$ in each round, the expected running time of the protocol is $2^{O(n \cdot t(n))}$. □

It is an open question whether $\mathsf{NP} \subseteq \mathsf{IP}(1)$ [11]. On the other hand, we can show that any language in NP has a constant-space exp-time qAM system having perfect-completeness.

Theorem 6. $\mathsf{NP} \subseteq \mathsf{qAM_1}(1)$, *where the protocol runs in exponential expected time.*

Proof. The proof follows from Theorem 5 and the fact that any language in NP is log-space (and so also poly-time) reducible to `KNAPSACK` [21]. □

Another surprising result about constant-space qAMs is as follows.

Theorem 7. $\mathsf{qAM_1}(1)$ *contains an NEXP-complete language.*

Proof. It is not hard to show that `SUCCINCT 0-1 KNAPSACK` (see Figure 3), a NEXP-complete language [14], can be reduced to `KNAPSCAK` via linear-space reduction. So, there is a constant-space qAM system having perfect-completeness for this language due to Theorem 5. □

The language `SUCCINCT 0-1 KNAPSACK` is composed by strings $a^m\#a^k\#\theta$ such that θ is a Boolean formula with variables $x_1, \ldots, x_{m+k}$ satisfying that the string $b\#a_1\#\cdots\#a_{2^k-1}$ defined based on θ (described below) is a member of `KNAPSCAK`, i.e. $\sum_{i=1}^{2^k-1} a_i z_i = b$ for some $z_1, \ldots, z_{2^k-1} \in \{0,1\}$, where b and each a_i are precisely 2^m bits (leading 0s permitted) binary numbers.

The string $b\#a_1\#\cdots\#a_{2^k-1}$ is defined based on θ as follows. Let $(i)_{2,k}$ be the k-bit binary representation of i.

- The i^{th} bit of b is the value of θ by setting $x_1, \ldots, x_{k+m}$ to the digits of $0^k(i)_{2,m}$ in order.
- The i^{th} bit of a_j is the value of θ by setting $x_1, \ldots, x_{k+m}$ to the digits of $(j)_{2,k}(i)_{2,m}$ in order.

Figure 3: The definition of `SUCCINCT 0-1 KNAPSACK` [14]

If the verifier uses logarithmic space, then qAM systems can also simulate poly-space (deterministic) reductions. Thus, we can extend Theorem 5 as follows.

Theorem 8. *Any language poly-space reducible to* `KNAPSACK` *is in* $\mathsf{qAM_1}(\mathsf{log})$. *Moreover, if the reduction takes $t(n)$-steps, the expected running time of the protocol is $2^{O(poly(n) \cdot t(n))}$.*

The best-known lower bound for $\mathsf{IP}(\mathsf{log})$ is EXP. By Theorem 8, we obtain a better lower bound for log-space qAM systems.

Theorem 9. $\mathsf{NEXP} \subseteq \mathsf{qAM_1}(\mathsf{log})$.

Proof. Any language in NEXP is log-space reducible to `SUCCINCT 0-1 KNAPSACK` [14], which is linear-space reducible to `KNAPSACK`. Thus, any language in NEXP is poly-space reducible to `KNAPSACK`. The proof follows from Theorem 8. □

Acknowledgements. We would like to thank Andris Ambainis and A. C. Cem Say for their many helpful comments on some drafts of this paper; H. Gökalp Demirci and John Watrous for their many helpful comments on the subject matter of this paper; and, Matthew Hague and Lane A. Hemaspaandra for kindly answering our questions.

A A review of previous private-coin protocols

In this part, we review the private-coin protocols for obtaining the results given in Facts 2 and 3, which we will refer as *the weak-protocol* and *the strong-protocol*, respectively. (Since Fact 4 is a generalization of Fact 3 [11], we omit its details.) The main idea behind both protocols for a given language is to simulate a fixed machine recognizing the given language: The prover sends to the verifier the computation of the simulated machine on a given input, which is a sequence of configurations starting from the initial configuration, and the verifier tries to verify the correctness of this sequence and then gives a decision with respect to the halting configuration sent by the prover. During the verification procedure, the following tasks can easily be checked deterministically: (i) each configuration has a correct format, and (ii) the sequence starts with the initial configuration and ends with a halting configuration. In the latter protocol, the length of each configuration is also checked to be at most linear. On the other hand, to check whether the prover sends a valid next configuration after each one is a non-trivial task. We will call this task *successor-check* and this is indeed where the private coin-flips come into play.

Since a single computation requires many adjunctive successor-checks and each of them contains many (private) coin-flips, a decision on the input can only be given with a very small probability after passing a single computation. Therefore, the prover sends the computation repeatedly in an infinite loop. Depending on the machine and resources of the verifier, either the verifier can always halt the computation with high probability or the protocol can run forever in some cases. For example, if the simulated machine never halts on the input and the prover honestly sends the corresponding computation, the verifier can never halt and give a decision.

Now, we give some protocol specific details. We begin with the weak-protocol of Condon and Lipton [10]. In his seminal paper [12], Freivalds presented a two-way probabilistic finite automaton (pfa) recognizing language

$$\mathtt{FRE} = \{a^{n_1}b^{n_1}a^{n_2}b^{n_2}\cdots a^{n_k}b^{n_k} \mid n_1,\ldots,n_k > 0, k > 0\}$$

with bounded error. Based on Freivalds' algorithm, Condon and Lipton proposed a private-coin protocol such that if a prover sends a member of FRE repeatedly to a one-way pfa verifier, then the verifier detects the memberships of the input with high probability. If the prover sends some nonmembers of FRE repeatedly to the same verifier, then the verifier gives a decision of rejection with high probability. Two-way finite automata with two-counters (2D2CA) are Turing-equivalent [19], and their main configuration elements are the contents of the counters, which can be encoded unary. Then successor-checks on a computation of a 2D2CA can be implemented by the protocol given by Condon and Lipton. Let L be a Turing-recognizable language and $\mathcal{D}$ be the 2D2CA recognizing it. For the members of L, the honest verifier sends finite valid configurations, and so the verifier accepts the input with high probability. For the nonmembers of L, the verifier can only accept if the last configuration of the computation is an accepting one. This means that the computation contains at least one defect, and so the probability of rejection is greater than the probability of acceptance due to the defect. Since a prover may never sends a halting configuration, the protocol has a weak-soundness.

In the case of the strong-protocol of Dwork and Stockmeyer [11], the simulated machine is an $O(n)$ space-bounded ATM, say $\mathcal{A}$. In order to simplify the proof, some inessential assumptions are made on $\mathcal{A}$: Roughly, each existential or universal transition leads to exactly two branches, there is no consecutive two existential or universal branching, $\mathcal{A}$ uses only $|x|+2$ space, $\mathcal{A}$ always halts exactly after $2^{c|x|}$ branching steps by keeping a counter, and so $\mathcal{A}$ never enters a loop, where x is a given input and c is an appropriate constant. Note that the computation tree of $\mathcal{A}$ on x has $2^{2^{c|x|}}$ leafs. The prover repeatedly sends the computation of some paths of this tree, in which the nondeterministic choices are made by the prover and the universal choices are made by the verifier, i.e. the verifier flips a coin and sends the outcome to the verifier. The configurations on a computation are separated by \$'s as

$$\cdots \$ c_i \$ c_{i+1} \$ c_{i+2} \$ \cdots ,$$

where $i > 0$. Since each configuration of $\mathcal{A}$ on x is fixed length $(|x|+3)$, the verifier can implement the successor-check by deterministically comparing the symbols at distance $|x|+4$ by using its input head.[3] If there is not exactly one \$ between two symbols, then the input is rejected by the verifier. The private part of the successor-check is that the verifier always selects the first symbol randomly and repeats this selection after each comparison. That is, if $l > 0$ is the index of a selected symbol, then it is compared with $(l+|x|+4)^{th}$ symbol, and then a new symbol with an index between $(l+|x|+4)+1$ and $(l+|x|+4)+|x|+4$ is selected. Since the protocol is private, the prover can never know which two symbols are compared. Thus, any defect on the computation can always be detected by the verifier with some probability, which is sufficiently greater than any accept probability. If there is no defect, the accepting probability is still very small so that a single rejecting probability on a leaf can always dominate the cumulative accepting probabilities from all the other leafs. (We omit the details here but the same issue is also detailed in the proof of Theorem 3.) Therefore, for the strings accepted by $\mathcal{A}$, none of successor-check produces a rejecting probability, and so the verifier accepts the input exactly by help of a honest prover. For the strings rejected by $\mathcal{A}$, if the input is not rejected by any successor-check, the input is rejected at least one path which is sufficient to dominate all the accepting decisions. Note that, if the prover never sends a halting configuration, the verifier detects infinitely many defects, which are sufficient to reject the input with probability 1, by using its input head.

As seen from the details, the private methods used by the protocols are different. In the former one, the verifier privately collects statistical evidence from over the computations to decide their correctness. Moreover, two-sided error is necessary in this case due to Theorem 1. In the latter case, the verifier can force the prover to send linear size configurations and then detects the defects directly. The latter protocol has also perfect-completeness.

B The proof of Theorem 1

The proof of $\mathsf{IP_1(s(n))} \subseteq \mathsf{weak\text{-}IP_1(s(n))} \subseteq \mathsf{ASPACE}(2^{2^{\mathsf{O(s(n))}}})$ follows from Lemma 3 (see below) and [4], where $s(n) \in \Omega(\log(n))$ is space-constructible.

We begin with a definition: The prover-verifier pair (P,V) is an *unbounded-error IPS having perfect completeness* for L if it has a perfect-completeness, i.e. satisfying condition $(1')$, and has the following soundness condition:

$2''$. for all $x \notin L$, and all provers P^*, the probability that (P^*,V) accepts x is less than 1.

[3]If the simulated machine is $s(n) \in \omega(n)$ space-bounded, then $\log(s(n))$ space-bounded verifier is sufficient for a similar check, where s is a space-constructible function.

The classes defined by unbounded-error IPS having perfect completeness will be shown by $\mathsf{IP}_{1,<1}(\cdot)$.

Lemma 3. *For any space-constructible $s(n) \in \Omega(\log(n))$,*

$$\mathsf{IP}_{1,<1}(\mathsf{s(n)}) \subseteq \mathsf{DTIME}(2^{2^{2^{\mathsf{O(s(n))}}}}).$$

Proof. Let $\mathtt{L} \in \mathsf{IP}_{1,<1}(\mathsf{s(n)})$. Then there exists an unbounded-error IPS having perfect-completeness (P, V) for $\mathtt{L}$. We will show that a DTM can recognize $\mathtt{L}$ by using triple-exponential time in $s(n)$.

Let x be an input string. Without loss of generality, we assume that the communication alphabet contains exactly two symbols $\{0, 1\}$. We represent the configuration set of V on x as $\mathcal{C}_V(x)$, whose size is exponential in $s(|x|)$. We classify the configurations into five groups:

1. `read`: the ones in a reading state
2. `comm-0`: the ones in a communication state ready to write 0 on the communication cell
3. `comm-1`: the ones in a communication state ready to write 1 on the communication cell
4. `acc`: the ones in an accepting state
5. `rej`: the ones in a rejecting state

The computation tree of (P', V) on x can be infinite for a prover P'. On the other hand, *having perfect completeness* allow us to build a finite computation tree that concisely represents the computation of V on x and its communications with all possible provers (P^*). First of all, we do not need to keep the probabilities of the configurations since the input is either accepted with probability 1 or with a probability less than 1. (We do not need the cumulative sums of the accepting and rejecting probabilities.) Secondly, the length of any halting path must be bounded by a certain number steps. If the computation does not halt, then V must enter an infinite loop. An infinite loop, *in our case*, can either contributes to a halting path or independent from the other parts of the computation. We visualize both cases in Figure 4. The former case can be seen as a part of the halting path since the probability of being in the loop approaches to zero and the halting path is re-traversed with the decreasing probability after each cycle. However, the probability of being in the loop remains the same in latter case, and so it should be replaced with a rejecting path if detected.

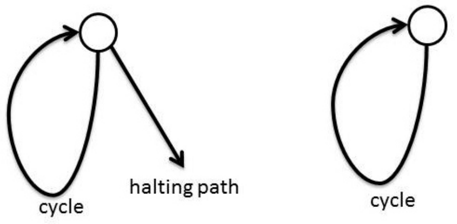

Figure 4: Two cases of infinite loops

We denote our finite tree $\mathcal{T}_V(x)$. (Note that we do not specify any prover since this tree represent all possible communication scenarios.) The structure and evaluation of $\mathcal{T}_V(x)$ is similar to the computation tree of an ATM. The main difference of $\mathcal{T}_V(x)$ is that a node can take three different values instead of two values, which are originally true and false. We give the details of how $\mathcal{T}_V(x)$ can be constructed below.

Since the protocol is private, we keep the configurations that follow the same communication strategy together. Therefore, each node of $\mathcal{T}_V(x)$ represents a subset of $\mathcal{C}_V(x)$. The root represents the initial configuration. We have four different types of inner nodes, i.e.

READ-COMM, COMM-01, COMM-0, and COMM-1,

and three different types of leafs, i.e.

ACC, REJ, and LOOP.

A READ-COMM node, say RC, is a node that contains at least one `read` configuration and may contain some `comm-0` or `comm-1` configurations. The child node(s) of RC is (are) determined as follows: Each `read` configuration in RC divides into at most two child configurations in a single step.

- If the child is an `acc` (a `rej`) configuration, then an ACC (a REJ) leaf is created and connected to the RC.
- If the child is a `read` configuration which is identical to a `read` configuration in RC, then a LOOP leaf is created and connected to RC.
- In any other case, each child should be one of a `read` (not in RC), a `comm-0`, or a `comm-1` configuration. All these children with the previous `comm-0` or `comm-1` configurations form a new node and connected to RC.

The type of the new node given in the last item is determined conditionally. If its depth (in $\mathcal{T}_V(x)$) exceeds a certain number ($2^{|\mathcal{C}_V(x)|}$ – the total number of all subsets of $\mathcal{C}_V(x)$), it becomes a LOOP leaf since it must be a repetition of a previous node along the same path.

Suppose that the depth of the new node does not exceed this number. If it contains at least one `read` configuration, then it becomes a READ-COMM node again. If it contains both `comm-0` and `comm-1` configurations, then it becomes a COMM-01 node. If it contains only some `comm-0` (`comm-1`) configurations, then it becomes a COMM-0 (COMM-1) node.

For each COMM-01 node, say C01, two new nodes are created and connect to C01. One of them becomes a COMM-0 node that contains all `comm-0` configurations of C01. The other one becomes a COMM-1 node that contains all `comm-1` configurations of C01. Both COMM-0 and COMM-1 are the communication nodes. We give the details for a COMM-0, say C0, node. (The situation is exactly the same for a COMM-1 node.) Let c be a configuration in C0. In c, the verifier writes 0 on the communication cell, and then receives 0 or 1. Since we consider all possible communications, there are two next configurations evolved from c in a single step, say c_0 and c_1. If c_0 (c_1) is an `acc` or a `rej` configuration, then an ACC or a REF leaf is created, respectively, and connected to the C0; or if c_0 (c_1) is a `comm-0` configuration which is identical to a `comm-0` configuration in C0, then a LOOP leaf is created and connected to C0. Otherwise, all c_0's and c_1's are collected into two new nodes and then connected to C0. The types of the new nodes are determined as explained above. We completed how $\mathcal{T}_V(x)$ can be constructed.

Now, we describe how $\mathcal{T}_V(x)$ can be evaluated. We associate each inner node with "$\wedge$" or "$\vee$" operator: READ-COMM and COMM-01 are associated with "$\wedge$" (universal choice), and COMM-0 and COMM-1 are associated with "$\vee$" (nondeterministic choice). These operators determines the value of a node from the values of its children. There are three types of values: true, false, and loop[depth], where depth is a numeric value. Obviously, any ACC (REJ) leaf takes the value of true (false). Any LOOP leaf takes the value of loop with a depth value that represent the smallest depth of the repeated node.

If the computation tree just contains true and false values, then its evaluation becomes trivial (exactly the same as alternation). The non-trivial part is how to include the loop into the evaluation. Suppose that a node associated with "$\wedge$" has $k > 0$ child/children, whose values are represented by $v_1, \ldots, v_k$. The value of the node can be calculated as follows:

$$v_1 \wedge (v_2 \wedge \cdots (v_{k-2} \wedge (v_{k-1} \wedge v_k))),$$

where binary relation "$\wedge$" is defined as follows:

$\wedge$	true	false	loop[d_1]
true	true	false	true
false	false	false	false
loop[d_2]	true	false	loop[$min\{\mathsf{d}_1, \mathsf{d}_2\}$]

Since it is a universal choice, any false value beats all the other values. Any true value beats any loop value, since this loop contributes the accepting path. Between two loop values, we select the one having smaller depth. The value of a "$\vee$" node can be calculated in a similar way with its specific rules:

$\vee$	true	false	loop[d_1]
true	true	true	true
false	true	false	loop[d_1]
loop[d_2]	true	loop[d_2]	loop[$min\{\mathsf{d}_1, \mathsf{d}_2\}$]

Since it is an existential (nondeterministic) choice, any true value beats all the other values. Any loop value beats any false value, since the corresponding loop may contribute an accepting path in a lower depth. Between two loop values, we again select the one having smaller depth. The last thing about the evaluation is that if a node takes value of loop and the depth of the loop refers to this node, then the value of the node is changed to false since this loop does not contribute any accepting path. The value of the root is the value of the tree.

In case of $x \in \mathtt{L}$, we know that there exists a nondeterministic strategy (corresponding to the communication with P) on the tree such that it does not lead to any rejecting path, and any infinite loop must contribute some accepting paths. Therefore, the value of the root is set to true.

In case of $x \notin \mathtt{L}$, we know that for every nondeterministic strategy (corresponding to the communication with P^*), there must be a nonzero rejecting path or a nonzero looping path (not contributing any accepting path) that dominates all the other opponent values during the evaluation. Therefore, the value of the root is set to false.

A DTM can construct and evaluate $\mathcal{T}_V(x)$ in a straightforward way. Since the depth of the tree is $2^{|\mathcal{C}_V(x)|}$, the total running time of the DTM is double-exponential in $|\mathcal{C}_V(x)|$. Since $|\mathcal{C}_V(x)|$ is exponential in $s(|x|)$, then the total running time becomes triple-exponential in $s(|x|)$. $\square$

References

[1] Dorit Aharonov and Tomer Naveh. Quantum NP – A survey. Technical report, 2002. arXiv:0210077.

[2] Andris Ambainis and John Watrous. Two–way finite automata with quantum and classical states. *Theoretical Computer Science*, 287(1):299–311, 2002.

[3] László Babai. Trading group theory for randomness. In *STOC'85: Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 421–429, 1985.

[4] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

[5] Anne Condon. *Computational Models of Games.* MIT Press, 1989.

[6] Anne Condon. Space-bounded probabilistic game automata. *Journal of the ACM*, 38(2):472–494, 1991.

[7] Anne Condon. *Complexity Theory: Current Research*, chapter The complexity of space bounded interactive proof systems, pages 147–190. Cambridge University Press, 1993.

[8] Anne Condon, Lisa Hellerstein, Samuel Pottle, and Avi Wigderson. On the power of finite automata with both nondeterministic and probabilistic states. *SIAM Journal on Computing*, 27(3):739–762, 1998.

[9] Anne Condon and Richard E. Ladner. Probabilistic game automata. *Journal of Computer and System Sciences*, 36(3):452–489, 1988.

[10] Anne Condon and Richard J. Lipton. On the complexity of space bounded interactive proofs (extended abstract). In *FOCS'89: Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 462–467, 1989.

[11] Cynthia Dwork and Larry Stockmeyer. Finite state verifiers I: The power of interaction. *Journal of the ACM*, 39(4):800–828, 1992.

[12] Rūsiņš Freivalds. Probabilistic two-way machines. In *Proceedings of the International Symposium on Mathematical Foundations of Computer Science*, pages 33–45, 1981.

[13] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC'85: Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 291–304, 1985.

[14] Matthew Hague and Anthony Widjaja Lin. Model checking recursive programs with numeric data types. In *CAV*, volume 6806 of *LNCS*, pages 743–759, 2011.

[15] Rahul Jain, Zhengfeng Ji, Sarvagya Upadhyay, and John Watrous. QIP = PSPACE. *Journal of the ACM*, 58(6):30, 2011.

[16] Alexei Kitaev. Quantum NP, January 1999. Talk at AQIS'99: Second Workshop on Algorithms in Quantum Information Processing.

[17] Emanuel Knill. Quantum randomness and nondeterminism. Technical Report arXiv:quant-ph/9610012, 1996.

[18] Chris Marriott and John Watrous. Quantum Arthur-Merlin games. *Computational Complexity*, 14(2):122–152, 2005.

[19] Marvin Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, 1967.

[20] Harumichi Nishimura and Tomoyuki Yamakami. An application of quantum finite automata to interactive proof systems. *Journal of Computer and System Sciences*, 75(4):255–269, 2009.

[21] Christos H. Papadimitriou. *Computational Complexity.* Addison Wesley, 1994.

[22] John H. Reif. The complexity of two-player games of incomplete information. *Journal of Computer and System Sciences*, 29(2):274–301, 1984.

[23] Adi Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, 1992.

[24] John Watrous. PSPACE has constant-round quantum interactive proof systems. In *FOCS'99: Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, pages 112–119, 1999.

[25] Abuzer Yakaryılmaz. Turing-equivalent automata using a fixed-size quantum memory. Technical Report arXiv:1205.5395, 2012.

[26] Abuzer Yakaryılmaz and A. C. Cem Say. Languages recognized by nondeterministic quantum finite automata. *Quantum Information and Computation*, 10(9&10):747–770, 2010.

[27] Abuzer Yakaryılmaz and A. C. Cem Say. NP has log-space verifiers with fixed-size public quantum registers. Technical Report arXiv:1101.5227, 2011.

[28] Abuzer Yakaryılmaz and A. C. Cem Say. Unbounded-error quantum computation with small space bounds. *Information and Computation*, 279(6):873–892, 2011.