Electronic Colloquium on Computational Complexity, Report No. 21 (2003)

QMA = PP implies that PP contains PH

M. N. Vyalyi
Institute for Quantum Information
California Institute of Technology
vyalyi@mccme.ru

April 3, 2003

Abstract

We consider possible equality **QMA** = **PP** and give an argument against it. Namely, this equality implies that **PP** contains **PH**. The argument is based on the strong form of Toda's theorem and the strengthening of the proof for inclusion $\mathbf{QMA} \subseteq \mathbf{PP}$ due to Kitaev and Watrous.

Keywords: complexity class, quantum computation, gap functions

Now many quantum analogs of classical complexity classes are known. The first example was the class **BQP** consisting of functions that are computable by polynomial-time quantum algorithms. Quantum algorithms can be described by quantum Turing machines as well as by uniform families of quantum circuits as shown by Yao [14].

Quantum interactive proof systems provide an another important example of quantum analog to classical complexity classes. It was shown by Watrous [11] that **PSPACE** has three-message quantum interactive proof systems. Later Kitaev and Watrous [8] proved that the class **QIP**(3) (three-message quantum interactive proof systems) coincides with the class of **QIP**(poly) (the number of messages is polynomial). Almost nothing is known about the class **QIP**(2). The smallest class in this hierarchy, **QIP**(1), was initially studied by Kitaev under the name **BQNP** and was considered as a quantum analog of **NP**. In fact, there is no interaction between verifier and prover in the case of one-message protocol. It seems more appropriate to address this class as a quantum analog of the class **MA** which is a probabilistic analog of **NP** ant the smallest class in Arthur–Merlin games hierarchy introduced in [1]. Therefore this class is now referred as **QMA**. Kitaev proved $\mathbf{QMA} \subseteq \mathbf{P}^{\#\mathbf{P}} \subseteq \mathbf{PSPACE}$. Later Kitaev and Watrous proved stronger result $\mathbf{QMA} \subseteq \mathbf{PP}$.

In this paper we consider possible equality $\mathbf{QMA} = \mathbf{PP}$. We will show that this equality is hardly possible because it would imply the inclusion $\mathbf{QMA} = \mathbf{PP} \supseteq \mathbf{PH}$. It is believed that this inclusion does not hold. As a motivation to that belief the relativized arguments can be applied. Beigel [2] constructed an oracle A such that $\mathbf{P}^{\mathbf{NP}^A} \not\subseteq \mathbf{PP}^A$.

ISSN 1433-8092

Two key ingredients for our result are Toda's theorem and arithmetic closure properties of **GapP** functions. Toda [10] proved that $\mathbf{P}^{\#\mathbf{P}} \supseteq \mathbf{PH}$. Moreover, the reduction algorithm uses one query to #**P** oracle only. This property is important to our proof.

The **GapP** functions were invented by Fenner, Fortnow and Kurtz [4]. The class **GapP** is the closure of #**P** under subtraction. More natural definition of **GapP** uses a notion of counting machine. *A counting machine* is a non-deterministic Turing machine running in polynomial time and finishing at either accepting or rejecting halting state. Given an input word x a counting machine produces a gap $g_M(x)$. *The gap* is the difference between the number of accepting computation paths and the number of rejecting computation paths.

Using gaps, it is easy to define many complexity classes such as **PP**, $\oplus$**P** and others. One of them is the class **AWPP**. Fortnow and Rogers [6] proved **BQP** $\subseteq$ **AWPP** and constructed such an oracle A that $\mathbf{P}^A = \mathbf{BQP}^A = \mathbf{AWPP}^A$ but the polynomial hierarchy is infinite. Li proved that **AWPP** is **PP**-low, i.e. $\mathbf{PP}^{\mathbf{AWPP}} = \mathbf{PP}$. This proof is sketched in [6]. **PP**-lowness of **AWPP** implies that if **BQP** = **PP** then **BQP** = **PP** $\supseteq$ **PH** due to Toda's theorem which implies that $\mathbf{P}^{\mathbf{PP}} \supseteq \mathbf{PH}$. Recently Fenner [3] simplified the definition of **AWPP** by establishing the amplification property for this class.

PP-lowness of **QMA** is unknown and it is doubtful. In this work we show that **QMA** is contained in some subclass of **PP** wider than **AWPP**. We denote this subclass by $\mathbf{A_0PP}$ to stress that it is to some extent an one-side analog of **AWPP**. This class $\mathbf{A_0PP}$ contains also the another quantum non-deterministic class **QNP** which coincides with $\mathbf{co\text{-}C_=P}$ [5, 13]. The amplification property for the $\mathbf{A_0PP}$ is obtained easily. We cannot prove **PP**-lowness of the $\mathbf{A_0PP}$ and cannot find out an oracle relative to which $\mathbf{A_0PP}$ collapses to **P**. Instead we propose the simple straightforward argument to show that $\mathbf{A_0PP} = \mathbf{PP}$ implies **PP** $\supseteq$ **PH**. In this argument we rely on the mentioned above property of Toda's reduction. In our construction we replace the oracle query by a guess and checking the correctness of this guess by some **PP**-machine. The amplification property of the $\mathbf{A_0PP}$ is important at this point.

1 Gap functions and gap-definable classes

In this section we reproduce definitions and facts about gap functions and gap-definable classes that will be used below. Then we define yet another complexity class $\mathbf{A_0PP}$. Unfortunately we cannot identify it with previously defined classes.

We will use the following properties of **GapP** functions.

1. **FP** $\subseteq$ **GapP**. **FP** is the class of functions computable in polynomial time by a deterministic Turing machine.

2. #**P** $\subseteq$ **GapP**. A #**P** function $f(x)$ counts the number of accepting paths for some counting machine M.

3. If $g_1, g_2 \in$ **GapP** then $-g_1 \in$ **GapP**, $g_1 + g_2 \in$ **GapP**, $g_1 g_2 \in$ **GapP**.

4. If $g(x) \in \mathbf{GapP}$, $f(x) \in \mathbf{FP}$, $f(x) > 0$ and $f(x) = O(|x|^{O(1)})$ then $(g(x))^{f(x)} \in \mathbf{GapP}$.

These facts are easy to prove. For more properties of **GapP** functions see [4].

The multiplication of gaps is achieved by concatenation of counting machines and XORing their halting states. An accepting state corresponds to 0 and the rejecting state corresponds to 1. More precisely, suppose a counting machine M_1 produces the gap g_1 and a counting machine M_2 produces the gap g_2. The machine M producing the gap $g_1 g_2$ imitates M_1 at the first stage of computation and stores the halting state of M_1. Then it imitates M_2. At the end of computation M accepts iff the halting states of M_1 and M_2 are the same; otherwise M rejects. This multiplying procedure can be iterated to obtain a product of polynomially many gaps. Also the XORing of acceptances can be applied along a a certain branch of computation to produce a gap in a form of exponential-size sum of polynomially sized products.

Let's recall the standard definition of the class **PP**.

Definition 1. *$L \in \mathbf{PP}$ iff there exists a counting machine M running in polynomial time such that*

- *if $x \in L$ then $g_M(x) > 0$;*
- *if $x \notin L$ then $g_M(x) \leq 0$.*

For our purposes it is convenient to use a slightly different definition of the **PP**.

Lemma 1. *$L \in \mathbf{PP}$ if there exist functions $g(x) \in \mathbf{GapP}$ and $t(x) \in \mathbf{FP}$ such that*

- *if $x \in L$ then $g(x) > t(x)$;*
- *if $x \notin L$ then $g(x) \leq t(x)$.*

Now we define a new counting class $\mathbf{A_0PP}$. In the next section we will show that this class contains **QMA**.

Definition 2. *$L \in \mathbf{A_0PP}$ iff there exist functions $g(x) \in \mathbf{GapP}$ and $T(x) \in$* **FP** (a threshold function) *such that*

- *if $x \in L$ then $g(x) > T(x)$;*
- *if $x \notin L$ then $0 \leq g(x) < \frac{1}{2}T(x)$.*

The inclusion $\mathbf{A_0PP} \subseteq \mathbf{PP}$ follows immediately from the definition. It also easy to obtain the inclusions $\mathbf{co\text{-}C_=P} \subseteq \mathbf{A_0PP}$, $\mathbf{AWPP} \subseteq \mathbf{A_0PP}$. Recall that $L \in \mathbf{co\text{-}C_=P}$ iff there exists a function $g \in \mathbf{GapP}$ such that

- $x \in L$ implies $g(x) \neq 0$;
- $x \notin L$ implies $g(x) = 0$.

Given such a function $g(x)$ we can take the function $2g(x)^2 \in \mathbf{GapP}$ and the threshold $T(x) = 1$ to conclude that $L \in \mathbf{A_0PP}$.

As for **AWPP**, it was proved by Fenner [3] that $L \in \mathbf{AWPP}$ iff there exist functions $g \in \mathbf{GapP}$ and $f \in \mathbf{FP}$ such that

- $x \in L$ implies $2/3 < g(x)/f(x) \leq 1$;
- $x \notin L$ implies $0 \leq g(x)/f(x) < 1/3$.

Taking $g(x)$ and $\lceil 2f(x)/3 \rceil$ as a threshold function we conclude that $L \in \mathbf{A_0PP}$.

We require in Definition 2 that a threshold ratio should be at least 2. It is easy to amplify the threshold ratio to an exponent by using the mentioned above property 4 of **GapP** functions.

Lemma 2. *$L \in \mathbf{A_0PP}$ iff for any polynomial r there exist functions $g \in \mathbf{GapP}$ and $T(x) \in \mathbf{FP}$ such that*

- *if $x \in L$ then $g(x) > T(x)$;*
- *if $x \notin L$ then $0 \leq g(x) < 2^{-r(|x|)}T(x)$.*

Similar to the **AWPP**, the class $\mathbf{A_0PP}$ can be characterized by thresholds in the form $2^{p(|x|)}$ where $p(\cdot)$ is a polynomial.

Lemma 3. *$L \in \mathbf{A_0PP}$ iff there exist a polynomial p and a function $g(x) \in$ **GapP** such that*

- *if $x \in L$ then $g(x) > 2^{p(|x|)}$;*
- *if $x \notin L$ then $0 \leq g(x) < \frac{1}{2}2^{p(|x|)}$.*

Proof. It is clear that the conclusion of the Lemma implies $L \in \mathbf{A_0PP}$.

Let's now prove the opposite. Consider a language $L \in \mathbf{A_0PP}$. Assume that functions g, T satisfy Definition 2. Since $g(x)$ is the gap of some counting machine (running in polynomial time), there exists a polynomial p such that $|g(x)| < \frac{1}{4}2^{p(|x|)}$ for all x.

Let $g'(x) = \frac{4}{3}\lfloor \frac{2^{p(|x|)}}{T(x)} \rfloor g(x)$. We claim that the polynomial $2p$ and the function $(g'(x))^2$ satisfy the conclusion of the Lemma 3. Indeed, suppose that $x \in L$. Then $g(x) > T(x)$ and we have

$$\frac{g'(x)}{2^{p(|x|)}} = \frac{4}{3}\left\lfloor \frac{2^{p(|x|)}}{T(x)} \right\rfloor \frac{g(x)}{2^{p(|x|)}} > \frac{4}{3}\left(\frac{2^{p(|x|)}}{T(x)} - 1\right)\frac{g(x)}{2^{p(|x|)}} > \frac{4}{3} - \frac{4}{3}\frac{g(x)}{2^{p(|x|)}} > 1. \quad (1)$$

If $x \notin L$ then $g(x) < \frac{1}{2}T(x)$ and we have

$$\frac{g'(x)}{2^{p(|x|)}} = \frac{4}{3}\left\lfloor \frac{2^{p(|x|)}}{T(x)} \right\rfloor \frac{g(x)}{2^{p(|x|)}} \leq \frac{4}{3}\frac{2^{p(|x|)}}{T(x)}\frac{g(x)}{2^{p(|x|)}} < \frac{2}{3}. \quad (2)$$

By squaring the inequalities (1) and (2) we come to the conclusion. □

2 QMA vs $\mathbf{A_0PP}$

We choose the standard basis (the Shor's basis) for quantum circuits. It consists of operators T, H, K where

$$T\colon |a,b,c\rangle \mapsto |a,b,c\oplus ab\rangle; \quad H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}; \quad K = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}. \tag{3}$$

It is well-known that this basis provides universal quantum computation (see books [7, 9] for details).

To define the **QMA** we use uniform families of quantum circuits. Such a family is defined by a function that maps a binary word x (input) to a description U_x of a quantum circuit (in the Shor basis). Informally, in the case of **QMA** computation this circuit determines verifier actions on an input word x. (Note that the word x can be used in the description of $U(x)$.) The circuit U_x acts on the space $\mathcal{P}\otimes\mathcal{V}$, where $\mathcal{P} = \left(\mathbb{C}^2\right)^{\otimes n}$ is the space of the quantum prover qubits and $\mathcal{V} = \left(\mathbb{C}^2\right)^{\otimes m}$ is the space of the quantum verifier qubits. Let $p(x)$ be the maximum of accepting probability over all possible prover messages $|\xi\rangle$. It can be expressed as

$$p(x) = \max_{|\xi\rangle\in\mathcal{P}} \langle\xi|\otimes\langle 0^m|U_x^\dagger \Pi^a U_x|\xi\rangle\otimes|0^m\rangle. \tag{4}$$

Here we assume that the first qubit contains 1 if the verifier accepts and 0 if the verifier rejects. The operator Π^a is the projector to the subspace $\mathcal{L}_a = \mathbb{C}\big(\{|1y\rangle\}_{y\in\{0,1\}^{n-1+m}}\big)$ of accepting final states. So, we come to the following definition.

Definition 3. $L \in$ **QMA** *iff there exists a polynomial time computable function* $x \mapsto U_x$ *mapping binary words to descriptions of quantum circuits such that*

- *if* $x \in L$ *then* $p(x) > 2/3$*;*
- *if* $x \notin L$ *then* $p(x) < 1/3$*.*

Theorem 1. QMA $\subseteq$ $\mathbf{A_0PP}$*.*

To relate **QMA** with counting complexity classes we use the following interpretation of (4) due to Kitaev and Watrous (see also [7]): $p(x)$ is the maximal eigenvalue of the operator

$$A = \operatorname{Tr}_{\mathcal{V}} V(x), \quad V(x) = U_x^\dagger \Pi^{(a)} U_x (I_{\mathcal{P}} \otimes |0^m\rangle\langle 0^m|). \tag{5}$$

The maximal eigenvalue $\lambda_{\max}$ of a positive operator can be estimated by the trace of (sufficiently large) degree of this operator. For the operator A we have

$$\lambda_{\max}^d \le \operatorname{Tr} A^d = \sum_{j=1}^{2^n} \lambda_j^d \le 2^n \lambda_{\max}^d. \tag{6}$$

We will assume that $d = n+1$ and will apply the bounds (6) to distinguish the cases $\lambda_{\max} < 1/3$ and $\lambda_{\max} > 2/3$. Note that

$$\begin{aligned} \lambda_{\max} < 1/3 \quad &\text{implies} \quad \operatorname{Tr} A^d < \frac{1}{2}\left(\frac{2}{3}\right)^d, \\ \lambda_{\max} > 2/3 \quad &\text{implies} \quad \operatorname{Tr} A^d > \left(\frac{2}{3}\right)^d. \end{aligned} \tag{7}$$

Let $h(x)$ be the total number of Hadamard gates H in the circuit U_x. Our choice of the basis for quantum circuits guarantees that the trace of A^d is a rational with the denominator $2^{-dh(x)}$: $\operatorname{Tr} A^d = a(x)2^{-dh(x)}$, $a(x) \in \mathbb{Z}$.

Lemma 4. $a(x) \in \mathbf{GapP}$.

Proof. Let $s(x)$ be a size of U_x. The operator $V(x)$ can be expressed in the form

$$V(x) = 2^{-h(x)} V_1 V_2 \ldots V_{2s+2} \tag{8}$$

where $V_k \in \{T, K, \sqrt{2}H, \Pi^{(a)}, I_{\mathcal{P}} \otimes |0^m\rangle\langle 0^m|\}$. Note that matrix elements $(V_k)_{(\alpha,\gamma),(\beta,\delta)} \in \{0, +1, -1, +i, -i\}$ ($\alpha, \beta \in \{0,1\}^n$; $\gamma, \delta \in \{0,1\}^m$; $k \in [1, 2s{+}2]$) and that the value of $(V_k)_{(\alpha,\gamma),(\beta,\delta)}$ is computable in polynomial time. So, we have

$$\bigl(V(x)\bigr)_{(\alpha,\gamma),(\beta,\delta)} = 2^{-h(x)} \sum \ldots (V_k)_{(\alpha_k,\gamma_k),(\alpha_{k+1},\gamma_{k+1})} \cdots \tag{9}$$

where summation is taken over all sequences $\{(\alpha_k, \gamma_k)\}$ such that $1 \le k \le 2s{+}3$, $\alpha_k \in \{0,1\}^n$, $\gamma_k \in \{0,1\}^m$, $\alpha_1 = \alpha$, $\gamma_1 = \gamma$, $\alpha_{2s+3} = \beta$, $\gamma_{2s+3} = \delta$.

Taking partial trace we get an expression for a matrix element of A:

$$A_{\alpha,\beta} = 2^{-h(x)} \sum_{\gamma} \bigl(V(x)\bigr)_{(\alpha,\gamma),(\beta,\gamma)}. \tag{10}$$

For matrix elements of A^d we have

$$(A^d)_{\alpha,\beta} = 2^{-dh(x)} \sum \ldots A_{\alpha_k,\alpha_{k+1}} \cdots \tag{11}$$

where summation is taken over all sequences $\{\alpha_k\}$ such that $1 \le k \le d+1$, $\alpha_k \in \{0,1\}^n$, $\alpha_1 = \alpha$, $\alpha_{d+1} = \beta$. For the $a(x)$ we obtain:

$$a(x) = 2^{dh(x)} \sum_{\alpha} \bigl(A^d\bigr)_{\alpha,\alpha}. \tag{12}$$

Thus, $a(x)$ is expressed as the exponential size sum of polynomially sized products of numbers taken from the set $\{0, \pm 1, \pm i\}$. The number of factors in each summand is $d(2s(x)+2)$. Summands are indexed by sequences $(\alpha_{jk}, \gamma_{jk})$, $1 \le j < d+1$, $1 \le k \le 2s+3$, $\alpha_{jk} \in \{0,1\}^n$, $\beta_{jk} \in \{0,1\}^m$. Each summand can be calculated in polynomial time.

Now it is easy to construct a counting machine M producing the gap $a(x)$. At the first stage of the computation the machine makes $2^{(n+m)d(2s(x)+3)}$ branches indexed by the sequences $\{(\alpha_{jk}, \gamma_{jk})\}$. Along each branch the machine computes the value of the summand in (12) indexed by the same sequence. If the value is ± 1 then the machine produces a gap according to this value. Otherwise it produces a zero gap. □

Thus, using this Lemma and the relations (7) for any language L from the **QMA** we can construct a **GapP** function $a(x)$ such that

- if $x \in L$ then $a(x) > 2^{dh(x)} \left(\frac{2}{3}\right)^d$;
- if $x \notin L$ then $a(x) < \frac{1}{2} 2^{dh(x)} \left(\frac{2}{3}\right)^d$.

This completes the proof of Theorem 1.

3 $\mathbf{A_0PP}$ vs PP

The class $\mathbf{A_0PP}$ looks very powerful. Is it coincide with **PP**? We cannot answer this question. But we can give an argument against the positive answer. Namely, we will prove the following theorem.

Theorem 2. *If* $\mathbf{A_0PP} = \mathbf{PP}$ *then* $\mathbf{A_0PP} \supseteq \mathbf{PH}$.

As mentioned above, to prove Theorem 2 we need the strong form of Toda's theorem: $\mathbf{P}^{\#\mathbf{P}[1]} \supseteq \mathbf{PH}$. In other words, any language L in the polynomial hierarchy is recognizable by a deterministic polynomial-time $\#\mathbf{P}$-oracle machine M that makes only one oracle query. Let $f(y) \in \#\mathbf{P}$ be the oracle answer on the query y.

Now we show how to recognize the language L by some (very restrictive!) **PP** machine M' that queries a **PP**-oracle g. The machine must obey the following conditions:

- along any computation path it makes just one query to the oracle;
- it is promised that the only one oracle's answer is "yes".

Note that possible values of $f(y)$ range from 0 to $2^{q(|x|)}$, $q(\cdot)$ is a polynomial. At first, the machine M' makes $2^{q(|x|)}$ branches indexed by possible values j of $f(y)$. Then the machine M' makes an oracle query about the sign of expression

$$g(y, j) = (f(y) - j - 1)(j - 1 - f(y)) \in \mathbf{GapP}. \tag{13}$$

It's easy to see that $g(y, j) > 0$ iff $f(y) = j$. If the answer is "yes" then M' assumes that $j = f(y)$ and imitates the behavior of M after the oracle query. If M accepts then M' produces a gap 1. Along all other branches M' produces a zero gap.

Thus, if $x \in L$ the machine M' produces the gap 1. Otherwise it produces the gap 0. It is obvious from the description of M' that it satisfies the aforementioned conditions.

To finish the proof of Theorem 2 let's suppose that $\mathbf{A_0PP} = \mathbf{PP}$. Due to Lemma 2 and Lemma 3 there exist a **GapP** function $\tilde{g}(y, j)$ and a polynomial p such that

$$\begin{aligned} &g(y,j) > 0 \ \Rightarrow\ \tilde{g}(y,j) > 2^{p(\ell)}, \\ &g(y,j) \le 0 \ \Rightarrow\ 0 \le \tilde{g}(y,j) < 2^{-q(|x|)} 2^{p(\ell)}, \end{aligned} \tag{14}$$

where $\ell = |y| + q(|x|)$. It is clear that $2^{p(\ell)} \in \mathbf{FP}$. By G we denote a counting machine such that G produces a gap $\tilde{g}(y, j)$ on the input (y, j).

Now we construct a **PP** machine M'' that recognizes the language L. The machine M'' operates similar to M'. But it replaces an oracle query by imitation of the machine G and produces a gap $\tilde{g}(y, j)$. This gap is multiplied by a gap produced by M' at the end of computation.

Let us calculate the gap produced by M'':

$$\begin{aligned} &x \in L \ \Rightarrow\ g_{M''}(x) > 2^{p(\ell)}, \\ &x \notin L \ \Rightarrow\ g_{M''}(x) < 2^{q(|x|)} 2^{-q(|x|)} 2^{p(\ell)} = 2^{p(\ell)}. \end{aligned} \tag{15}$$

Applying Lemma 1 we get $L \in \mathbf{PP}$.

Corollary 1. *If* $\mathbf{QMA} = \mathbf{PP}$ *then* $\mathbf{QMA} = \mathbf{PP} \supseteq \mathbf{PH}$.

This corollary follows immediately from Theorems 2 and 1.

Corollary 2. *If* $\mathbf{co\text{-}C_{=}P} = \mathbf{PP}$ *then* $\mathbf{co\text{-}C_{=}P} = \mathbf{PP} \supseteq \mathbf{PH}$.

References

[1] L. Babai. Trading group theory for randomness. *Proceedings of ACM STOC'17*, pages 421–429, 1985.

[2] R. Beigel. Perceptrons, PP, and the Polynomial Hierarchy. *Computational Complexity* (4), 1994, pages 339–349.

[3] S. Fenner. PP-lowness and a simple definition of AWPP. *Theory of Computing Systems*. Volume 36 (2003) pages 199-212. See also ECCC Report TR02-036, 2002.

[4] S. Fenner, L. Fortnow, S. Kurtz. Gap-definable counting classes. *J. of Comp. and Sys. Sci.* (48), 1994, pages 116–148.

[5] S. Fenner, F. Green, S. Homer, R. Pruim. Determining acceptance possibility for a quantum computation is hard for the polynomial hierarchy. *Proc. of the Royal Society London* Ser. A (455), 1999, pages 3953–3966.

[6] L. Fortnow, J. Rogers. Complexity limitations on Quantum Computation. *Proc. 13th Conference on Computational Complexity*, 1998, pages 202–209. E-version: cs.CC/9811023

[7] A. Kitaev, A. Shen, M. Vyalyi. *Classical and Quantum Computation.* AMS, 2002.

[8] A. Kitaev, J. Watrous. Parallelization, amplification, and exponential time simulation of quantum interactive proof systems, *Proceedings of ACM STOC*, pages 608–617, 2000.

[9] M. A. Nielsen, I. I. Chuang. *Quantum computation and quantum information.* Cambridge Univ. Press, 2000.

[10] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. on Comput.* (20) 1991, pages 865–877.

[11] J. Watrous. PSPACE has constant-round quantum interactive proof systems. *Proceedings of IEEE FOCS*, pp. 112-119, 1999.

[12] J. Watrous. Succinct quantum proofs for properties of finite groups, *Proceedings of IEEE FOCS*, pages 537–546, 2000. E-version: cs.CC/0009002.

[13] T. Yamakami, A. C-C. Yao. $\mathrm{NQP}_{\mathrm{C}} = \text{co-}\mathrm{C}_{=}P$. *Information Processing Letters* (71) 1999, pages 63–69. E-version: quant-ph/9812032.

[14] A. C-C. Yao. Quantum circuit complexity. *Proceedings of IEEE FOCS'34*, 1993. P. 352.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'