On the Complexity of Quantum ACC

Frederic Green
Department of Mathematics and Computer Science
Clark University, Worcester, MA 01610
fgreen@black.clarku.edu

Steven Homer
Computer Science Department
Boston University, Boston, MA 02215
homer@cs.bu.edu

Christopher Pollett
Department of Mathematics
University of California, Los Angeles, CA
cpollett@willow.math.ucla.edu

Abstract

For any $q > 1$, let MOD_q be a quantum gate that determines if the number of 1's in the input is divisible by q. We show that for any $q, t > 1$, MOD_q is equivalent to MOD_t (up to constant depth). Based on the case $q = 2$, Moore [8] has shown that quantum analogs of AC$^{(0)}$, ACC$[q]$, and ACC, denoted QAC$_{wf}^{(0)}$, QACC$[2]$, QACC respectively, define the same class of operators, leaving $q > 2$ as an open question. Our result resolves this question, proving that QAC$_{wf}^{(0)}$ = QACC$[q]$ = QACC for all q. We also develop techniques for proving upper bounds for QACC in terms of related language classes. We define classes of languages EQACC, NQACC and BQACC$_{\mathbf{Q}}$. We define a notion of $\log$-planar QACC operators and show the appropriately restricted versions of EQACC and NQACC are contained in P/poly. We also define a notion of $\log$-gate restricted QACC operators and show the appropriately restricted versions of EQACC and NQACC are contained in TC$^{(0)}$. To do this last proof, we show that TC$^{(0)}$ can perform iterated addition and multiplication in certain field extensions. We also introduce the notion of a polynomial-size tensor graph and we show that families of such graphs can encode the amplitudes resulting from applying an arbitrary QACC operator to an initial state.

1. Introduction

Advances in quantum computation in the last decade have been among the most notable in theoretical computer science. This is due to the surprising improvements in the efficiency of solving several fundamental combinatorial problems using quantum mechanical methods in place of their classical counterparts. These advances led to considerable efforts in finding new efficient quantum algorithms for classical problems and in developing a complexity theory of quantum computation.

While most of the original results in quantum computation were developed using quantum Turing machines, they can also be formulated in terms of quantum circuits, which yield a more natural model of quantum computation. For example, Shor [10] has shown that quantum circuits can factor integers more efficiently than any known classical algorithm for factoring. And quantum circuits have been shown (see Yao [16]) to provide a universal model for quantum computation.

In the classical setting, small depth circuits are considered a good model for parallel computing. Constant-depth circuits, corresponding to constant parallel time, are of central importance. For example, constant-depth circuits of AND, OR and NOT gates of polynomial size (called AC$^{(0)}$ circuits) can add and subtract binary numbers. The class ACC extends AC$^{(0)}$ by allowing modular counting gates. The class TC$^{(0)}$, consisting of constant-depth threshold circuits, can compute iterated multiplication.

In studying quantum circuits, it is natural to consider the power of small depth circuit families. Quantum circuit models analogous to the central classical circuit classes have recently been studied by Moore and Nilsson [7] and Moore [8]. They investigated the properties of classes of quantum operators QAC$_{wf}^{(0)}$, QACC$[q]$, and QNC defined to be analogous to and to contain their classical counterparts. This paper is a contribution to this line of research.

For example, a quantum analog of AC$^{(0)}$, defined by Moore and denoted QAC$_{wf}^{(0)}$, is the class of families of operators which can be built out of products of constantly many layers consisting of polynomial-sized tensor products of one-qubit gates (analogous to NOT's), Toffoli gates (analogous to AND's and OR's) and fan-out gates[1]. An analog of ACC$[q]$ (i.e., ACC circuit families only allowing Mod_q gates) is QACC$[q]$, defined similarly to $\text{QAC}_{wf}^{(0)}$, but replacing the fan-out gates with quantum Mod_q gates (which we denote as MOD_q). QACC is the same class but we allow MOD_q gates for every q. Moore [8] proves the surprising result $\text{QAC}_{wf}^{(0)}=$ QACC$[2]$ = QACC. This is in sharp contrast to the classical result of Smolensky [13] that says $\text{ACC}^{(0)}[q] \neq \text{ACC}^{(0)}[p]$ for any pair of distinct primes q, p, which implies that for any prime p, $\text{AC}^{(0)} \subset \text{ACC}^{(0)}[p] \subset$ ACC. This result showed that parity gates are as powerful as any other mod gates in QACC, but left open the complexity of MOD_q gates for $q > 2$.

In [8], Moore conjectured that QACC $\neq$ QACC$[q]$ for odd q. In this paper, we provide the missing ingredients to show that in fact QACC= QACC$[q]$ for any $q \geq 2$. Moore's result showed that parity is as good as any other MOD_q gate; our result further shows that any MOD_q gate is as good as any other. The main technical contribution is the application of the Quantum Fourier Transform (using complex q^{th} roots of unity), and encodings of base q digits using qubits.

We also develop methods for proving upper bounds for language classes related to QACC. Our methods result in upper bounds for restricted QACC circuits. Roughly speaking, we show that QACC is no more powerful than P/Poly provided that a layer of "wire-crossings" in the QACC operator can be written as log many compositions of Kronecker products of controlled-not gates. We call this class $\text{QACC}_{pl}^{\log}$, where the "pl" is for this planarity condition. We show if one further restricts attention to the case where the number of multi-line gates (gates whose input is more than 1 qubit) is log-bounded then the circuits are no more powerful than $\text{TC}^{(0)}$. We call this class $\text{QACC}_{gates}^{\log}$. These results hold for arbitrary complex amplitudes in the QACC circuits.

To be more precise, it is necessary to show how a class of operators in QACC can define a language, as usually considered in complexity theory. In this paper, we define classes of languages EQACC, NQACC, and BQACC based on the expectation of observing a certain state after applying the QACC operator to the input state. For example, the class NQACC corresponds to the case where x is in the language if the expectation of the observed state after applying the QACC operator is non-zero. This is analogous to the definition of the class NQP in Fenner et al. [5].

In this paper, we show that $\text{NQACC}_{gates}^{\log}$is in $\text{TC}^{(0)}$ and $\text{NQACC}_{pl}^{\log}$is in P/poly. Although the proof uses some of the techniques developed by Yamakami and Yao [14] to show that $\text{NQP}_C = \text{co-C}_=\text{P}$, the small depth circuit case presents technical challenges not present in their setting. In particular, given a QACC operator built out of layers $M_1, \ldots, M_t$ and an input state $|x, 0^{p(n)}\rangle$, we must show that a $\text{TC}^{(0)}$ circuit can keep track of the amplitudes of each possible resulting state as each layer is applied. After all layers have been applied, the $\text{TC}^{(0)}$ circuit then needs to be able to check that the amplitude of one possible state is non-zero. Unfortunately, there could be exponentially many states with non-zero amplitudes after applying a layer. To handle this problem we introduce the idea of a "tensor-graph," a new way to represent a collection of states. We can extract from these graphs (via $\text{TC}^{(0)}$ or P/poly computations) whether the amplitude of any particular vector is non-zero.

The exponential growth in the number of states is one of the primary obstacles to proving that all of NQACC is in $\text{TC}^{(0)}$ (or even P/Poly), and thus the tensor graph formalism represents a significant step towards such an upper bound. The reason the bounds apply only in the restricted cases is that although tensor graphs can represent any QACC operator, in the case of operators with layers that might do arbitrary permutations, the top-down approach we use to compute a desired amplitude from the graph no longer seems to work. We feel that it is likely that the amplitude of any vector in a tensor graph can be written as a polynomial product of a polynomial sum in some extension algebra of the ones we work with in this paper, in which case it is quite likely it can be evaluated in $\text{TC}^{(0)}$.

Another important obstacle to obtaining a $\text{TC}^{(0)}$ upper bound is that one needs to be able to add and multiply a polynomial number of complex amplitudes that may appear in a QACC computation. We solve this problem. It reduces to adding and multiplying polynomially many elements of a certain transcendental extension of the rational numbers. We show that in fact $\text{TC}^{(0)}$ is closed under iterated addition and multiplication of such numbers (Lemma 4.1 below). This result is of independent interest, and our application of tensor-graphs and these closure properties of $\text{TC}^{(0)}$ may prove useful in further investigations of small-depth quantum circuits.

We now discuss the organization of the rest of this paper. In the next section we introduce the definitions and notations we use in this paper. Then in the following section we prove QACC$[q]$ = QACC. Finally, in the last section, we prove the $\text{TC}^{(0)}$ and P/poly upper bounds for the restricted classes discussed above.

2. Preliminaries

In this section we define the gates used as building blocks for our quantum circuits. Classes of operators built out of these gates are then defined. We define language classes that can be determined by these operators and give a couple definitions from algebra. Lastly, some closure properties of $\text{TC}^{(0)}$ are described.

[1]The subscript "wf" in the notation denotes "with fan-out." The idea of fan-out in the quantum setting is subtle, as will be made clearer later in this paper. See Moore [8] for a more in-depth discussion.

Definition 2.1

By a one-qubit gate *we mean an operator from the group* $U(2)$.

Let $U = \begin{pmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{pmatrix} \in U(2)$. $\wedge_m(U)$ *is defined as:* $\wedge_0(U) = U$ *and for* $m > 0$, $\wedge_m(U)$ *is*

$$\wedge_m(U)(|\vec{x}, y\rangle) = \begin{cases} u_{y0}|\vec{x}, 0\rangle + u_{y1}|\vec{x}, 1\rangle & \text{if } \wedge_{k=1}^m x_k = 1 \\ |\vec{x}, y\rangle & \text{otherwise} \end{cases}$$

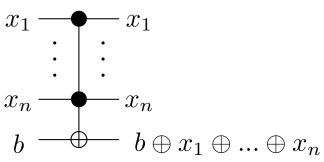
Let $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. *A* Tofolli gate *is a* $\wedge_m(X)$ *gate for some* $m \geq 0$. *A* controlled-not *gate is a* $\wedge_1(X)$ *gate.*

An (m-)spaced controlled-not gate *is an operator that maps* $|y_1, \ldots, y_m, x\rangle$ *to* $|x \oplus y_1, y_2 \ldots, y_m, x\rangle$ *or* $|y_1, \ldots, y_m, x\rangle$ *to* $|x, y_1 \ldots, y_{m-1}, y_m \oplus x\rangle$

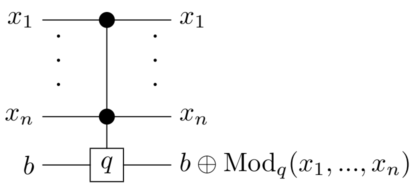
An (m-ary) fan out gate F *is an operator that maps from* $|y_1, \ldots, y_m, x\rangle$ *to* $|x \oplus y_1, \ldots, x \oplus y_m, x\rangle$.

A $\mathrm{MOD}_{q,r}$ *gate is an operator that maps* $|y_1, \ldots, y_m, x\rangle$ *to* $|y_1, \ldots, y_m, x \oplus (\sum y_i \bmod q \equiv r)\rangle$.

We use the following graphical notation for parity (i.e., MOD_2) or, in the case of $n = 1$, for controlled-not:



and for MOD_q:



As discussed in [8], the no-cloning theorem of quantum mechanics makes it difficult to directly fan out qubits in constant depth (although constant fan-out is no problem). Thus it is necessary to define the operator F as in the above definition; refer to [8] for further details. Also, in the literature it is frequently the case that one says a given operator M on $|y_1, \ldots, y_m\rangle$ can be written as a tensor product of certain gates. What is meant is that there is an permutation operator Π (a map $|y_1, \ldots, y_m\rangle$ to $|y_{\pi(1)}, \ldots, y_{\pi(m)}\rangle$ for some permutation π) such that

$$M|y_1, \ldots y_m\rangle = \Pi \otimes_j^n M_j \Pi^{-1}|y_1, \ldots y_m\rangle$$

where the M_i's are our base gates, i.e., those gates for which no inherent ordering on the y_i is assumed *a priori*. Since it is important to keep track of such details in our upper bounds proofs, we will always use Kronecker products of the form $\otimes_j^n M_j$ without unspoken permutations. Nevertheless, being able to do permutation operators (not conjugation by a permutation) intuitively allows our circuits to simulate classical wire crossings. To handle permutations, we allow our circuits to have controlled-not layers. A *controlled-not layer* is a gate which performs, in one step, controlled-not's between an arbitrary collection of disjoint pairs of lines in its domain. That is, it performs $\Pi \otimes_j^n \wedge_1(X)\Pi^{-1}$ for some permutation operator Π. Moore Nilsson [7] show that any permutation can be written as a finite product of controlled-not layers. We say a controlled-not layer is *log-depth* if it can be written as the composition of log many matrices each of which is the Kronecker product of identities and spaced controlled-not gates.

$M^{\otimes n}$ is the n-fold Kronecker product of M with itself. The next definitions are based on Moore [8].

Definition 2.2

QAC$^{(k)}$ *is the class of families* $\{F_n\}$, *where* F_n *is in* $U(2^{n+p(n)})$, p *a polynomial, and each* F_n *is writable as a product of* $O(\log^k n)$ *layers, where a* layer *is a Kronecker product of one-qubit gates and Toffoli gates or is a controlled-not layer. Also for all* n *the number of distinct types of one qubit gates used must be fixed.*

QACC$^{(k)}[q]$ *is the same as QAC*$^{(k)}$ *except we also allow* MOD_q *gates. QACC*$^{(k)} = \cup_q$*QACC*$^{(k)}[q]$.

QAC$^{(k)}_{wf}$ *is the same as QAC*$^{(k)}$ *but we also allow fan-out gates.*

QACC is defined as QACC$^{(0)}$ *and QACC*$[q]$ *is defined as QACC*$^{(0)}[q]$. *QACC*$^{\log}_{pl}$ *is QACC restricted to log-depth controlled not layers. QACC*$^{\log}_{gates}$ *is QACC restricted so that the total number of multi-line gates in all layers is log-bounded.*

If $\mathcal{C}$ *is one of the above classes, then* $\mathcal{C}_K$ *are the families in* $\mathcal{C}$ *with coefficients restricted to* K.

Let $\{F_n\}$ *and* $\{G_n\}$, $G_n, F_n \in U(2^n)$ *be families of operators. We say* $\{F_n\}$ *is* QAC$^{(0)}$ reducible *to* $\{G_n\}$ *if there is a family* $\{R_n\}$, $R_n \in U(2^{n+p(n)})$ *of QAC*$^{(0)}$ *operators augmented with operators from* $\{G_n\}$ *such that for all* n, $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$, *there is a setting of* $z_1, ..., z_{p(n)} \in \{0,1\}$ *for which* $\langle \mathbf{y}|F_n|\mathbf{x}\rangle = \langle \mathbf{y}, \mathbf{z}|R_n|\mathbf{x}, \mathbf{z}\rangle$. *Operator families are* QAC$^{(0)}$ equivalent *if they are QAC*$^{(0)}$ *reducible to each other. If* $\mathcal{C}_1$ *and* $\mathcal{C}_2$ *are families of QAC*$^{(0)}$ *equivalent operators, we write* $\mathcal{C}_1 = \mathcal{C}_2$.

We refer to the z_i's above as "auxiliary bits" (called "ancillae" in [8]). Note that in proving QAC$^{(0)}$ equivalence, the auxiliary bits must be returned to their original values in a computation.

It follows for any $\{F_n\} \in \text{QAC}^{(0)}$ that F_n is writable as a product of finite number of layers. Moore [8] shows $\text{QAC}^{(0)}_{wf} = \text{QACC}[2] = \text{QACC}$. Moore [8] places no restriction on the number of distinct types of one-qubit gates used in a given family of operators. We do this so that the number of distinct amplitudes which appear in matrices in a layer is fixed with respect to n. This restriction arises implicitly in the quantum Turing machine case of the upper bounds proofs in Fenner, et al. [5] and Yamakami and Yao [14]. Also, it seems fairly natural since in the classical case one builds circuits using a fixed number of distinct gate types. Our classes are, thus, more "uniform" than Moore's. We now define language classes based on our classes of operator families.

Definition 2.3 *Let $\mathcal{C}$ be a class of families of $U(2^{n+p(n)})$ operators where p is a polynomial and $n = |x|$.*

1. *E·$\mathcal{C}$ is the class of languages L such that for some $\{F_n\} \in \mathcal{C}$ and $\{\langle \vec{z}_n|\} = \{\langle z_{n,1}, \ldots, z_{n,n+p(n)}|\}$ a family of states, $m := |\langle \vec{z}_n|F_n|x, 0^{p(n)}\rangle|^2$ is 1 or 0 and $x \in L$ iff $m = 1$.*
2. *N·$\mathcal{C}$ is the class of languages L such that for some $\{F_n\} \in \mathcal{C}$ and $\{\langle \vec{z}_n|\}$ a family of states, $x \in L$ iff $|\langle \vec{z}_n|F_n|x, 0^{p(n)}\rangle|^2 > 0$.*
3. *B·$\mathcal{C}$ is the class of languages L so that for some $\{F_n\} \in \mathcal{C}$ and $\{\langle \vec{z}|\}$, $x \in L$ if $|\langle \vec{z}_n|F_n|x, 0^{p(n)}\rangle|^2 > 3/4$ and $x \notin L$ if $|\langle \vec{z}_n|F_n|x, 0^{p(n)}\rangle|^2 < 1/4$.*

It follows E·$\mathcal{C}$ $\subseteq$ N·$\mathcal{C}$ and E·$\mathcal{C}$ $\subseteq$ B·$\mathcal{C}$. We frequently will omit the '·' when writing a class, so E·QACC is written as EQACC. Let $|\Psi\rangle := F_n|x, 0^{p(n)}\rangle$. Notice that $|\langle \vec{z}_n|F_n|x, 0^{p(n)}\rangle|^2 = \langle \Psi|P_{|\vec{z}_n\rangle}|\Psi\rangle$, where $P_{|\vec{z}_n\rangle}$ is the projection matrix onto $|\vec{z}_n\rangle$. We could allow in our definitions measurements of up to polynomially many such projection observables and not affect our results below. However, this would shift the burden of the computation in some sense away from the QACC operator and instead onto preparation of the observable.

Next are some variations on familiar definitions from algebra.

Definition 2.4 *Let $k > 0$. A subset $\{\beta_i\}_{1 \le i \le k}$ of $\mathbf{C}$ is* linearly independent *if $\sum_{i=1}^{k} a_i \beta_i \neq 0$ for any $(a_1, \ldots, a_k) \in \mathbf{Q}^k - \{\vec{0}^k\}$. A set $\{\beta_i\}_{1 \le i \le k}$ is* algebraically independent *if the only $p \in \mathbf{Q}[x_1, \ldots, x_k]$ with $p(\beta_1, \ldots, \beta_k) = 0$ is the zero polynomial.*

We now briefly mention some closure properties of $\text{TC}^{(0)}$ computable functions that are useful in proving $\text{NQACC}^{\log}_{gates} \subseteq \text{TC}^{(0)}$. For proofs of the statements in the next lemma see [11, 12, 3].

Lemma 2.5 *(1) TC$^{(0)}$ functions are closed under composition. (2) The following are TC$^{(0)}$ computable: $x + y$, $x \dot{-} y := x - y$ if $x - y > 0$ and 0 otherwise, $|x| := \lceil \log_2(x+1) \rceil$, $x \cdot y$, $\lfloor x/y \rfloor$, $2^{\min(i, p(|x|))}$, and $cond(x, y, z) := y$ if $x > 0$ and z otherwise. (3) If $f(i, x)$ is a TC$^{(0)}$ computable then $\sum_{k=0}^{p(|x|)} f(k, x)$, $\prod_{k=0}^{p(|x|)} f(k, x)$, $\forall i \le p(|x|)(f(i, x) = 0)$, $\exists i \le p(|x|)(f(i, x) = 0)$, and $\mu i \le p(|x|)(f(i, x) = 0) :=$ the least i such that $f(i, x) = 0$ or $p(x) + 1$ otherwise, are TC$^{(0)}$ computable.*

We drop the $\min$ from the $2^{\min(i, p(|x|))}$ when it is obvious a suitably large $p(|x|)$ can be found. We define $max(x, y) := cond(1 \dot{-} (y \dot{-} x)), x, y)$ and define

$$max_{i \le p(|x|)}(f(i)) \;:=\; (\mu i \le p(|x|)\ )(\forall j \le p(|x|)(f(j) \dot{-} f(i) = 0)$$

Using the above functions we describe a way to do sequence coding in $\text{TC}^{(0)}$. Let $\beta_{|t|}(x, w) := \lfloor (w \dot{-} \lfloor w/2^{(x+1)|t|} \rfloor \cdot 2^{(x+1)|t|})/2^{x|t|} \rfloor$. The function $\beta_{|t|}$ is useful for block coding. Roughly, $\beta_{|t|}$ first gets rid of the bits after the $(x+1)|t|$th bit then chops off the low order $x|t|$ bits. Let $B = 2^{|\max(x,y)|}$, so that B is longer than either x or y. Hence, we code pairs as $\langle x, y\rangle := (B + y) \cdot 2B + B + x$, and projections as $(w)_1 := \beta_{\lfloor \frac{1}{2}|w| \rfloor \dot{-} 1}(0, \beta_{\lfloor \frac{1}{2}|w| \rfloor}(0, w))$ and $(w)_2 := \beta_{\lfloor \frac{1}{2}|w| \rfloor \dot{-} 1}(0, \beta_{\lfloor \frac{1}{2}|w| \rfloor}(1, w))$. We can encode a poly-length, $\text{TC}^{(0)}$ computable sequence of numbers $\langle f(1), \ldots, f(k)\rangle$ as the pair $\langle \sum_i^k (f(i) 2^{i \cdot m}), m\rangle$ where $m := |f(\max_i(f(i)))| + 1$. We then define the function which projects out the ith member of a sequence as $\beta(i, w) := \beta_{(w)_2}(i, w)$.

We can code integers using the positive natural numbers by letting the negative integers be the odd natural numbers and the positive integers be the even natural numbers. $\text{TC}^{(0)}$ can use the $\text{TC}^{(0)}$ circuits for natural numbers to compute both the polynomial sum and polynomial product of a sequence of $\text{TC}^{(0)}$ definable integers. It can also compute the rounded quotient of two such integers. For instance, to do a polynomial sum of integers, compute the natural number which is the sum of the positive numbers in the sum using $cond$ and our natural number iterated addition circuit. Then compute the natural number which is the sum of the negative numbers in the sum. Use the subtraction circuit to subtract the smaller from the larger number and multiply by two. One is then added if the number should be negative. For products, we compute the product of the natural numbers which results by dividing each integer code by two and rounding down. We multiply the result by two. We then sum the number of terms in our product which were negative integers. If this number is odd we add one to the product we just calculated. Finally, division can be computed using the Taylor expansion of $1/x$.

3. QACC[q]

In this section, we show QACC[q]=QACC for any $q \geq 2$.

Let $q \in \mathbf{N}$, $q \geq 2$ be fixed throughout this discussion. Consider quantum states labelled by digits in $D = \{0, ..., q-1\}$. By analogy with "qubit," we refer to a state of the form,

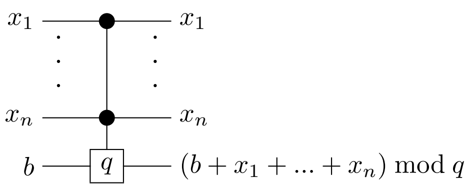
$$\sum_{k=0}^{q-1} c_k |k\rangle$$

with $\sum_k |c_k|^2 = 1$ as a "qudigit." Direct products of the basis states will be labelled by lists of eigenvalues, e.g., $|x\rangle|y\rangle$ is denoted as $|x, y\rangle$.

We define three important operations on qudigits. The n-ary *modular addition* operator M_q acts as follows:

$$M_q|x_1, ..., x_n, b\rangle = |x_1, ...x_n, (b + x_1 + ... + x_n) \bmod q\rangle$$

The gate is represented graphically as in the following figure:



Since M_q merely permutes the states, it is clear that it is unitary. Similarly, the n-ary unitary *base q fanout* operator F_q acts as,

$$F_q|x_1, ...x_n, b\rangle = |(x_1 + b) \bmod q, ...(x_n + b) \bmod q, b\rangle.$$

We write F for F_2, since it is the "standard" fan-out gate introduced by Moore (see Definition 2.1). Note that $M_q^{-1} = M_q^{q-1}$ and $F_q^{-1} = F_q^{q-1}$.

Finally, the Quantum Fourier Transform H_q (which generalizes the Hadamard transform H on qubits) acts on a single qudigit as,

$$H_q|a\rangle = \frac{1}{\sqrt{q}} \sum_{b=0}^{q-1} \zeta^{ab} |b\rangle,$$

where $\zeta = e^{\frac{2\pi i}{q}}$ is a primitive complex q^{th} root of unity. It is easy to see that H_q is unitary, via the fact that $\sum_{\ell=0}^{q-1} \zeta^{a\ell} = 0$ iff $a \not\equiv 0 \bmod q$.

The first observation is that, analogous to parity and fanout for Boolean inputs, the operators M_q and F_q are "conjugates" in the following sense.

Proposition 3.1 $M_q = (H_q^{\otimes(n+1)})^{-1} F_q^{-1} H_q^{\otimes(n+1)}$.

Proof. We apply the operators $H_q^{\otimes(n+1)}$, F_q^{-1}, and $(H_q^{\otimes(n+1)})^{-1}$ in that order to the state $|x_1, ..., x_n, b\rangle$, and check that the result has the same effect as M_q.

The operator $H_q^{\otimes(n+1)}$ simply applies H_q to each of the $n+1$ qudigits of $|x_1, ..., x_n, b\rangle$, which yields,

$$\frac{1}{q^{\frac{(n+1)}{2}}} \sum_{\mathbf{y} \in D^n} \sum_{a=0}^{q-1} \zeta^{\mathbf{x}\cdot\mathbf{y}+ab} |y_1, ..., y_n, a\rangle,$$

where $\mathbf{y}$ is a compact notation for $y_1, ..., y_n$, and $\mathbf{x} \cdot \mathbf{y}$ denotes $\sum_{i=1}^{n} x_i y_i$. Then applying F_q^{-1} to the above state yields,

$$\frac{1}{q^{\frac{(n+1)}{2}}} \sum_{\mathbf{y} \in D^n} \sum_{a=0}^{q-1} \zeta^{\ \mathbf{x}\cdot\mathbf{y}+ab} \\ |(\ y_1 - a) \bmod q, ..., (y_n - a) \bmod q, a\rangle.$$

By a change of variable, the above can be re-written as,

$$\frac{1}{q^{\frac{(n+1)}{2}}} \sum_{\mathbf{y} \in D^n} \sum_{a=0}^{q-1} \zeta^{\sum_{i=1}^{n} x_i(y_i+a)+ab} |y_1, ..., y_n, a\rangle$$

Finally, applying $(H_q^{\otimes(n+1)})^{-1}$ to the above undoes the Fourier transform and puts the coefficient of a in the exponent into the last slot of the state. The result is,

$$(H_q^{\otimes(n+1)})^{-1} F_q^{-1} H_q^{\otimes(n+1)} |x_1, ..., x_n, b\rangle = \\ |x_1, ..., x_n, (b + x_1 + ... + x_n) \bmod q\rangle,$$

which is exactly what M_q would yield.

□

We now describe how the operators M_q, F_q and H_q can be modified to operate on registers consisting of qu*bits* rather than qu*digits*. Firstly, we encode each digit using $\lceil \log q \rceil$ bits. Thus, for example, when $q = 3$, the basis states $|0\rangle, |1\rangle$ and $|2\rangle$ are represented by the two-qubit registers $|00\rangle, |01\rangle$ and $|10\rangle$, respectively. Note that there remains one state (in the example, $|11\rangle$) which does not correspond to any of the qudigits. In general, there will be $2^{\lceil \log q \rceil} - q$ such "non-qudigit" states. M_q, F_q and H_q can now be defined to act on qubit registers, as follows. Consider a state $|x\rangle$ where x is a number represented as m bits (i.e., an m-qubit register). If $m < \lceil \log q \rceil$, then H_q leaves $|x\rangle$ unaffected. If $0 \leq x \leq q-1$ (where here we are identifying x with the number it represents), then H_q acts exactly as one expects, namely, $H_q|x\rangle = (1/\sqrt{q}) \sum_{y=0}^{q-1} \zeta^{xy} |y\rangle$. If $x \geq q$, again H_q leaves $|x\rangle$ unchanged. Since the resulting transformation is a direct sum of unit matrices and matrices of the form of H_q as it was originally set down, the result is a unitary transformation. M_q and F_q can be defined to operate similarly on m-qubit registers for any m: Break up the m bits into blocks of $\lceil \log q \rceil$ bits. If m is not divisible by $\lceil \log q \rceil$, then M_q and F_q do not affect the "remainder" block that contains fewer than $\lceil \log q \rceil$ bits. Likewise, in a quantum register $|x_1, ..., x_n\rangle$ where each of the x_i's (with the possible exception of x_n) are $\lceil \log q \rceil$-bit numbers, M_q and F_q operate on the blocks of bits $x_1, ..., x_n$ exactly as expected, except that there is no affect on the "non-qudigit" blocks (in which $x_i \geq q$), or on the (possibly) one remainder block for which $|x_n| < \lceil \log q \rceil$. Since M_q and F_q operate exactly as they did originally on blocks representing qudigits, and like unity for non-qudigit or remainder blocks, it is clear that they remain unitary.

Henceforth, M_q, F_q, and H_q should be understood to act on qubit registers as described above. Nevertheless, it will usually be convenient to think of them as acting on qu*digit* registers consisting of $\lceil \log q \rceil$ qubits in each.

Lemma 3.2 *F_q and M_q are QAC$^{(0)}$-equivalent.*

Proof. By Barenco et al. [1], any fixed dimension unitary matrix can be computed in fixed depth using one-qubit gates and controlled nots. Hence H_q can be computed in QAC$^{(0)}$, as can $H_q^{\otimes(n+1)}$. The result now follows immediately from Proposition 3.1. □

The classical Boolean Mod_q-function on n bits is defined so that $\mathrm{Mod}_q(x_1, ..., x_n) = 1$ iff $\sum_{i=1}^n x_i \equiv 0 \pmod q$. (The more common definition sets it to 1 if $\sum_{i=1}^n x_i$ is *not* divisible by q, but this convention is less convenient in this setting, and is not important technically either.) We also define $\mathrm{Mod}_{q,r}(x_1, ..., x_n)$ to output 1 iff $\sum_{i=1}^n x_i \equiv r \pmod q$. Note that $\mathrm{Mod}_q = \mathrm{Mod}_{q,0}$. Reversible, quantum versions of these functions can also be defined. The operator $\mathrm{MOD}_{q,r}$ on $n+1$ qubits has the following effect:

$$|x_1, ..., x_n, b\rangle \mapsto |x_1, ..., x_n, b \oplus \mathrm{Mod}_{q,r}(x_1, ..., x_n)\rangle.$$

We write $\mathrm{MOD}_{q,0}$ as MOD_q. Since negation is built into the output (via the exclusive OR), it is easy to simulate negations using $\mathrm{MOD}_{q,r}$ gates. For example, by setting $b = 1$, we can compute $\neg\mathrm{Mod}_{q,r}$. More generally, using one auxiliary bit, it is possible to simulate "$\neg\mathrm{MOD}_{q,r}$," defined so that,

$$|x_1, ..., x_n, b\rangle \mapsto |x_1, ..., x_n, b \oplus (\neg\mathrm{Mod}_{q,r}(x_1, ..., x_n))\rangle,$$

using just $\mathrm{MOD}_{q,r}$ and a controlled-NOT gate. Thus $\mathrm{MOD}_{q,r}$ and $\neg\mathrm{MOD}_{q,r}$ are QAC$^{(0)}$-equivalent. Moore's version of MOD_q is our $\neg\mathrm{MOD}_q$. Observe that $\mathrm{MOD}_{q,r}^{-1} = \mathrm{MOD}_{q,r}$.

Lemma 3.3 *MOD_q and M_q are QAC$^{(0)}$-equivalent.*

Proof. First note that MOD_q and $\mathrm{MOD}_{q,r}$ are equivalent, since a $\mathrm{MOD}_{q,r}$ gate can be simulated by a MOD_q gate with $q - r$ extra inputs set to the constant 1. Hence we can freely use $\mathrm{MOD}_{q,r}$ gates in place of MOD_q gates.

It is easy to see that, given an M_q gate, we can simulate a MOD_q gate. Applying M_q to $n+1$ digits (represented as bits, but each digit only taking on the values 0 or 1) transforms,

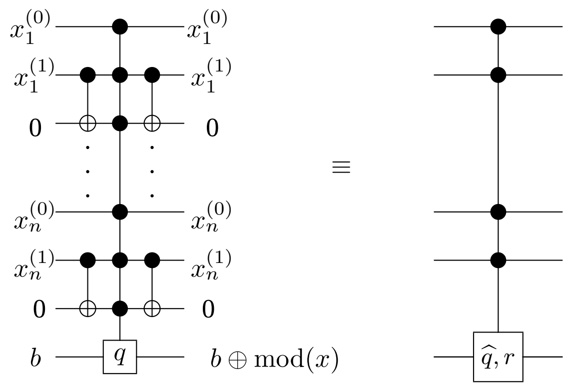
$$|x_1, ..., x_n, 0\rangle \mapsto |x_1, ..., x_n, (\sum_i x_i) \bmod q\rangle.$$

Now send the bits of the last block ($\sum_i x_i \bmod q$) to a Toffoli gate with all inputs negated and control bit b. The resulting output is exactly $b \oplus \mathrm{Mod}_q(x_1, ..., x_n)$. The bits in the last block can be erased by re-negating them and reversing the M_q gate. This leaves only $x_1, ..., x_n$, $O(n)$ auxiliary bits, and the output $b \oplus \mathrm{Mod}_q(x_1, ..., x_n)$.

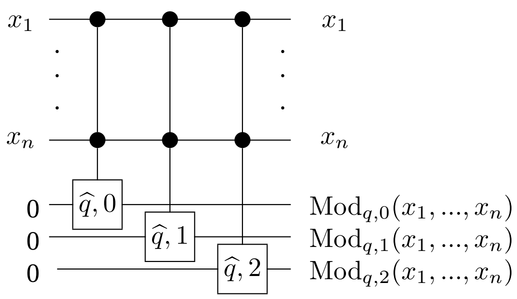
The converse (simulating M_q given MOD_q) requires some more work. The first step is to show that MOD_q can also determine if a sum of *digits* is divisible by q. Let $x_1, ..., x_n \in D$ be a set of digits represented as $\lceil \log q \rceil$ bits each. For each i, let $x_i^{(k)}$ ($0 \leq k \leq \lceil \log q \rceil - 1$) denote the bits of x_i. Since the numerical value of x_i is $\sum_{k=0}^{\lceil \log q \rceil - 1} x_i^{(k)} 2^k$, it follows that

$$\sum_{i=1}^n x_i = \sum_{k=0}^{\lceil \log q \rceil - 1} \sum_{i=1}^n x_i^{(k)} 2^k.$$

The idea is to express this last sum in terms of a set of Boolean inputs that are fed into a MOD_q gate. To account for the factors 2^k, each $x_i^{(k)}$ is fanned out 2^k times before plugging it into the MOD_q gate. Since $k < \lceil \log q \rceil$, this requires only constant depth and $O(n)$ auxiliary bits (which of course are set back to 0 in the end by reversing the fanout). Thus, just using MOD_q and constant fanout, we can determine if $\sum_{i=1}^n x_i \equiv 0 \pmod q$. More generally, we can determine if $\sum_{i=1}^n x_i \equiv r \pmod q$ using just a $\mathrm{MOD}_{q,r}$ gate and constant fanout. Let $\widehat{\mathrm{MOD}}_{q,r}(x_1, ..., x_n)$ denote the resulting circuit, that determines if a sum of digits is congruent to r mod q. The construction of $\widehat{\mathrm{MOD}}_{q,r}(x_1, ..., x_n)$ is illustrated in the figure below for the case of $q = 3$. In the figure, $\mathrm{mod}(x)$ denotes $\mathrm{Mod}_{3,r}(x_1, ..., x_n)$. The notation on the right will be used as a shorthand for this circuit:

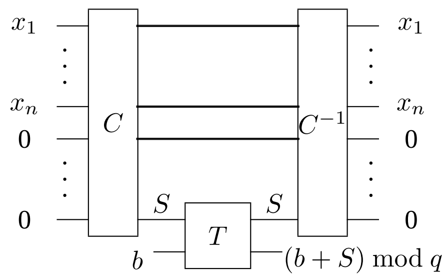


We can get the bits in the value of the sum $\sum_{i=1}^{n} x_i \bmod q$ using $\widehat{\mathrm{MOD}}_{q,r}$ circuits. This is done, essentially, by implementing the relation $x \bmod q = \sum_{r=0}^{q-1} r \cdot \mathrm{Mod}_{q,r}(x)$. For each r, $0 \leq r \leq q-1$, we compute $\mathrm{Mod}_{q,r}(x_1, ..., x_n)$ (where now the x_i's are digits). This can be done by applying the $\widehat{\mathrm{MOD}}_{q,r}$ circuits in series (for each r) to the same inputs, introducing an auxiliary 0-bit for each application, as illustrated here.



Let r_k denote the k^{th} bit of r. For each r and for each k, we take the AND of the output of the $\widehat{\mathrm{MOD}}_{q,r}$ with r_k (again by applying the AND's in series, which is still constant depth, but introduces q extra auxiliary inputs). Let $a_{k,r}$ denote the output of one of these AND's. For each k, we OR together all the $a_{k,r}$'s, that is, compute $\vee_{r=0}^{q-1} a_{k,r}$, again introducing a constant number of auxiliary bits. Since only one of the r's will give a non-zero output from $\widehat{\mathrm{MOD}}_{q,r}$, this collection of OR gates outputs exactly the bits in the value of $\sum_{i=1}^{n} x_i \bmod q$. Call the resulting circuit C, and the sum it outputs S.

Finally, to simulate M_q, we need to include the input digit $b \in D$. To do this, we apply a unitary transformation T to $|S, b\rangle$ that transforms it to $|S, (b+S) \bmod q\rangle$. By Barenco, et al. [1] (as in the proof of Lemma 3.2), T can be computed in fixed depth using one-qubit gates and controlled NOT gates. Now using S and all the other auxiliary inputs, we reverse the computation of the circuit C, thus clearing the auxiliary inputs. This is illustrated in this figure:



The result is an output consisting of $x_1, ..., x_n$, $O(n)$ auxiliary bits, and $(b + \sum_{i=1}^{n} x_i) \bmod q$, which is the output of an M_q gate. □

It is clear that we can fan out digits, and therefore bits, using an F_q gate (setting $x_i = 0$ for $1 \leq i \leq n$ fans out n copies of b). It is slightly less obvious (but still straightforward) that, given an F_q gate, we can fully simulate an F gate.

Lemma 3.4 *For any $q > 2$, F and F_q are QAC$^{(0)}$-equivalent.*

Proof. By the preceeding lemmas, F_q and MOD_q are QAC$^{(0)}$-equivalent. By Moore's result, MOD_q is QAC$^{(0)}$-reducible to F. Hence F_q is QAC$^{(0)}$-reducible to F.

Conversely, arrange each block of $\lceil \log q \rceil$ input bits to an F_q gate as follows. For the control-bit block (which contains the bit we want to fan out), set all but the last bit to zero, and call the last bit b. Set all bits in the i^{th} input-bit block to 0. Now the i^{th} output of the F_q circuit is b, represented as $\lceil \log q \rceil$ bits with only one possibly nonzero bit. Send this last output bit b and the input bit x_i to a controlled-NOT gate. The outputs of that gate are b and $b \oplus x_i$. Now apply F_q^{-1} to the bits that were the outputs of the F_q gate (which are all left unchanged by the controlled-not's). This returns all the b's to 0 except for the control bit which is always unchanged. The outputs of the controlled-not's give the desired $b \oplus x_i$. Thus the resulting circuit simulates F, with $O(n)$ auxiliary bits. □

Theorem 3.5 *For any $q \in \mathbf{N}$, $q \neq 1$, QACC = QACC[q].*

Proof. By the preceeding lemmas, fanout of bits is equivalent to the MOD_q function. By Moore's result, we can do MOD_q if we can do fanout in constant depth. By our result, we can do fanout, and hence MOD_2, if we can do MOD_q. Hence QACC = QACC[2] $\subseteq$ QACC[q]. □

4. Upper Bounds

In this section, we prove the following upper bounds results $\mathrm{NQACC}^{\log}_{gates} \subseteq \mathrm{TC}^{(0)}$, $\mathrm{BQACC}^{\log}_{\mathbf{Q},gates} \subseteq \mathrm{TC}^{(0)}$, $\mathrm{NQACC}^{\log}_{pl} \subseteq$ P/poly, and $\mathrm{BQACC}^{\log}_{\mathbf{Q},pl} \subseteq$ P/poly.

Suppose $\{F_n\}$ and $\{z_n\}$ determine a language L in NQACC. Let F_n be the product of the layers $U_1, \ldots, U_t$ and E be the distinct entries of the matrices used in the U_j's. By our definition of QACC, the size of E is fixed with respect to n. We need a canonical way to write sums and products of elements in E to be able to check $|\langle \vec{z}|U_1 \cdots U_t|x, 0^{p(n)}\rangle|^2 > 0$ with a $\mathrm{TC}^{(0)}$ function. To do this let $A = \{\alpha_i\}_{1 \leq i \leq m}$ be a maximal algebraically independent subset of E. Let $F = \mathbf{Q}(A)$ and let $B = \{\beta_i\}_{0 \leq i < d}$ be a basis for the field G generated by the elements in $(E - A) \cup \{1\}$ over F. Since the size of the bases of F and G are less than the cardinality of E the size of these bases is also fixed with respect to n.

As any sum or product of elements in E is in G, it suffices to come up with a canonical form for elements in G. Our representation is based on Yamakami and Yao [14]. Let $\alpha \in G$. Since B is a basis, $\alpha = \sum_{j=0}^{d-1} \lambda_j \beta_j$ for some $\lambda_j \in F$. We encode an α as a d-tuple (we iterate the pairing function from the preliminaries to make d-tuples) $\langle \ulcorner\lambda_0\urcorner, \ldots, \ulcorner\lambda_{d-1}\urcorner\rangle$ where $\ulcorner\lambda_j\urcorner$ encodes λ_j. As the elements of A are algebraically independent, each $\lambda_j = s_j/u_j$ where s_j and u_j are of the form

$$\sum_{\vec{k}_j, |\vec{k}_j| \leq e} a_{\vec{k}_j} (\prod_{i=1}^{m} \alpha_i^{k_{ij}}).$$

Here $\vec{k}_j = (k_{1j}, \ldots, k_{mj}) \in \mathbf{Z}^m$, $|\vec{k}_j|$ is $\sum_i k_{ij}$, $a_{\vec{k}_j} \in \mathbf{Z}$, and $e \in \mathbf{N}$. In particular, any product $\beta_m \cdot \beta_l = \sum_{j=0}^{d-1} \lambda_j \beta_j$ with $\lambda_j = s_j/u_j$ and s_j and u_j in this form. We take a common denominator u for elements of $E \cup \{\beta_m \cdot \beta_l\}$ and not just E since the λ_j's associated with the $\beta_m \cdot \beta_l$ might have additional factors in their denominators not in E. Also fix an e large enough to bound the $|\vec{k}_j|$'s which might appear in any element of E or a product $\beta_m \cdot \beta_l$. This e will be constant with respect to n. In multiplying t layers of QACC circuit against an input, the entries in the result will be polynomial sums and products of elements in $E \cup \{\beta_m \cdot \beta_l\}$, so we can bound $|\vec{k}_j|$ for $\vec{k}_j$'s which appear in the λ_j's of such an entry by $e \cdot p(n)$. To complete our representation of $\alpha \in G$ we encode λ_j as the sequence $\langle r, \langle\langle a_{\vec{k}_j}, k_{1j}, \ldots, k_{mj}\rangle\rangle\rangle$ where r is the power to which u is raised and $\langle\langle a_{\vec{k}_j}, k_{1j}, \ldots, k_{mj}\rangle\rangle$ is the sequence of $\langle a_{\vec{k}_j}, k_{1j}, \ldots, k_{mj}\rangle$'s that appear in s_j. By our discussion, the encoding of an α that appears as an entry in the output after applying a QACC operator to the input is of polynomial length and so can be manipulated in $\mathrm{TC}^{(0)}$.

We have need of the following lemma:

Lemma 4.1 *Let p be a polynomial. (1) Let $f(i,x) \in$ TC$^{(0)}$ output encodings of $a_{i,x} \in \mathbf{Z}[A]$. Then $\mathbf{Z}[A]$ encodings of $\sum_{i=1}^{p(|x|)} a_{i,x}$ and $\prod_{i=1}^{p(|x|)} a_{i,x}$ are TC$^{(0)}$ computable. (2) Let $f(i,x) \in$ TC$^{(0)}$ output encodings of $a_{i,x} \in G$. Then G encodings of $\sum_{i=1}^{p(|x|)} a_{i,x}$ and $\prod_{i=1}^{p(|x|)} a_{i,x}$ are TC$^{(0)}$ computable.*

Proof. We will abuse notation in this proof and identify the encoding $f(i,x)$ with its value $a_{i,x}$. So $\sum_i f(i,x)$ and $\prod_i f(i,x)$ will mean the encoding of $\sum_i a_{i,x}$ and $\prod_i a_{i,x}$ respectively.

(1) To do sums, the first thing we do is form the list $L1 = \langle f(0,x), \ldots, f(p(|x|), x)\rangle$. Then we create a flattened list $L2$ from this with elements which are the $\langle a_{\vec{k}_j}, k_{1j}, \ldots, k_{mj}\rangle$'s from the $f(i,x)$'s. $L1$ is in $\mathrm{TC}^{(0)}$ using our definition of sequence from the preliminaries, and closure under sums and max_i to find the length of the longest $f(i,x)$. To flatten $L1$ we use max_i to find the length d of the longest $f(i,x)$ for $i \leq p(|x|)$. Then using max twice we can find the length of the longest $\langle a_{\vec{k}_j}, k_{1j}, \ldots, k_{mj}\rangle$. This will be the second coordinate in the pair used to define sequence $L2$. We then do a sum of size $d \cdot p(|x|)$ over the subentries of $L1$ to get the first coordinate of the pair used to define $L2$. Given $L2$, we make a list $L3$ of the distinct $\vec{k}_j$'s that appear as $\langle a_{\vec{k}_j}, k_{1j}, \ldots, k_{mj}\rangle$ in some $f(i,x)$ for some $i \leq p(|x|)$. This list can be made from $L2$ using sums, $cond$ and μ. We sum over the $t \leq length(L2)$ and check if there is some $t' < t$ such that the t'th element of $L2$ has same $\vec{k}_j$ as t and if not add the tth elements $\vec{k}_j$ times 2 raised to the appropriate power. We know what power by computing the sum of the number of smaller t' that passed this test. Using $cond$ and closure under sums we can compute in $\mathrm{TC}^{(0)}$ a function which takes a list like $L2$ and a $\vec{k}_j$ and returns the sum of all the $a_{\vec{k}_j}$'s in this list. So using this function and the lists $L2$ and $L3$ we can compute the desired encoding.

For products, since the α_i's of A are algebraically independent, $\mathbf{Z}[A]$ is isomorphic to the polynomial ring $\mathbf{Z}[y_1, \ldots, y_m]$ under the natural map which takes α_j to y_j. We view our encodings $f(i,x)$ as m-variate polynomials in $\mathbf{Z}[y_1, \ldots, y_m]$. We describe for any p' a circuit that works for any $\mathrm{TC}^{(0)}$ computable $f(i,x)$ such that $\prod_i f(i,x)$ is of degree less than p' viewed as an m-variate polynomial. In $TC^{(0)}$ we define $g(i,x)$ to consist of the sequence of polynomially many integer values which result from evaluating the polynomial encoded by $f(i,x)$ at the points $(i_1, \ldots, i_m) \in \mathbf{N}^m$ where $0 \leq i_s$ and $\sum_s i_s \leq p'$. To compute $f(i,x)$ at a point involves computing a polynomial sum of a polynomial product of integers, and so will be in $TC^{(0)}$. Using closure under polynomial integer products we compute $k(j,x) := \prod_i \beta(j, g(i,x))$ where β is the sequence projection function from the preliminaries. Our choice of points is what is called by Chung and Yao [2] the *p'-th order principal lattice* of the m-simplex given by the origin and the points p' from the origin in each coordinate axis. By Theorems 1 and 4 of that paper (proved earlier by a harder argument in Nicolaides [9]) the multivariate Lagrange Interpolant of degree p' through the points $k(j,x)$ is unique. This interpolant is of the form $P(y_1,\ldots,y_m) = \sum_j p_j(y_1,\ldots,y_m)k(j,x)$ where the p_j's are polynomials which do not depend on the function f. An explicit formula for these p_j's is given in Corollary 2 of Chung and Yao [2] as a polynomial product of linear factors. Since these polynomials are all of degree less than p', they have only polynomial in p' many coefficients and in PTIME these coefficients can be computed by iteratively multiplying the linear factors together. We can then hard code these p_j's (since they don't depend on f) into our circuit and with these p_j's, $k(j,x)$, and closure under sums we can compute the polynomial of the desired product in TC$^{(0)}$.

(2) We do sums first. Assume $f(i,x) := \sum_{j=0}^{d-1}\lambda_{ij}\beta_j$. One immediate problem is that the λ_{ij} and $\lambda_{i'j}$ might use different u^r's for their denominators. Since TC$^{(0)}$ is closed under poly-sized maximum, it can find the maximum value r_0 to which u is raised. Then it can define a function $g(i,x) = \sum_{j=0}^{d-1}\gamma_{ij}\beta_j$ which encodes the same element of G as $f(i,x)$ but where the denominators of the γ_{ij}'s are now u^{r_0}. If λ_j was s_j/u^r we need to compute the encoding $s_j \cdot u^{r_0-r}/u^{r_0}$. This is straightforward from (1). Now

$$\sum_{i=1}^{p(|x|)} f(i,x) = \sum_{i=1}^{p(|x|)} g(i,x) = \sum_{j=0}^{d-1}[(\sum_{i=1}^{p(|x|)} s_{ij})/u^{r_0}]\beta_j,$$

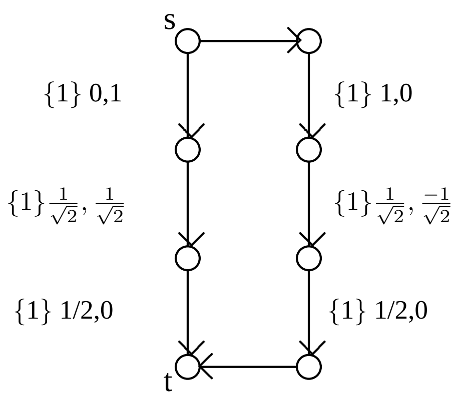
where s_{ij}'s are the numerators of the γ_{ij}'s in $g(i,x)$. From part (1) we can compute the encoding e_j of $(\sum_{i=1}^{p(|x|)} s_{ij})$ in TC$^{(0)}$. So the desired answer $\langle\langle r_0, e_0\rangle, \cdots, \langle r_0, e_{d-1}\rangle\rangle$ is in TC$^{(0)}$.

For products $\prod_{i=1}^{p(|x|)} f(i,x)$, we play the same trick as the in the $\mathbf{Z}[A]$ product case. We view our encodings of elements of G as d-variate polynomials in $F(y_0,\ldots,y_{d-1})$ under the map β_k goes to y_k. (Note that this map is not necessarily an isomorphism.) We then create a function $g(i,x)$ which consists of the sequence of values obtained by evaluating $f(i,x)$ at polynomially many points in a lattice as in the first part of this lemma. Evaluating $f(i,x)$ at a point can easily be done using the first part of this lemma. We then use part (1) of this lemma to compute the products $k(j,x) = \beta(j, g(i,x))$. We then get the interpolant $P(y_0,\ldots,y_{d-1}) = \sum_j p_j(y_0,\ldots,y_m)k(j,x)$. We non-uniformly obtain the encoding of $p_j(\beta_0,\ldots,\beta_{d-1})$ expressed as an element of G. i.e., in the form $\sum_{w=0}^{d-1}\lambda_{jw}\beta_w$. Thus, the product $\prod_{i=1}^{p(|x|)} f(i,x)$ is

$$\sum_{w=0}^{d-1}(\sum_j \lambda_{jw}k(j,w))\beta_w$$

The encoding of the products is the d-tuple given by $\langle\sum_j \lambda_{j0}k(j,0),\ldots,\sum_j \lambda_{jd-1}k(j,d-1)\rangle$ Each of its components is a polynomial sum of a product of two things in F and can be computed using the first part of the lemma. □

For $\{F_n\} \in$ QAC$^{(0)}_{wf}$ = QACC, the vectors that F_n act on are elements of a $2^{n+p(n)}$ dimensional space $\mathcal{E}_{1,n+p(n)}$ space which is a tensor product of the 2-dimensional spaces $\mathcal{E}_1,\ldots\mathcal{E}_{n+p(n)}$, which in turn are each spanned by $|0\rangle, |1\rangle$. We write $\mathcal{E}_{j,k}$ for the subspace $\otimes_{i=j}^k\mathcal{E}_i$ of $\mathcal{E}_{1,n+p(n)}$. We now define a succinct way to represent a set of vectors in $\mathcal{E}_{1,n+p(n)}$ which is useful in our argument below. A *tensor graph* is a directed acyclic graph with one source node of indegree zero, one terminal node of outdegree zero, and two kinds of edges: horizontal edges, which are unlabeled, and vertical edges, which are labeled with a pair of amplitudes and a product of *colors* and *anticolors*. (The color product may be the number 1.) We require that all paths from the source to the terminal traverse the same number of vertical edges and that no vertex can have vertical edge indegree greater than one or outdegree greater than one. For a color c we write $\tilde{c}$ for its corresponding anticolor. The *height* of a node in a tensor graph is the number of vertical edges traversed to get to it on any path from the source; the *height* of an edge is the height of its end node. The *width* of a tensor graph is maximum number of nodes of the same height. As an example of a tensor graph where our color product is the number 1, consider the following figure:



The rough idea of tensor graphs is that paths through the graph correspond to collections of vector in $\mathcal{E}_{1,n}$. For this particular figure the left path from the source node (s) to the terminal node (t) corresponds to the vectors given by

$$|1\rangle \otimes (\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle) \otimes \frac{1}{2}|0\rangle$$

and the right hand path corresponds to

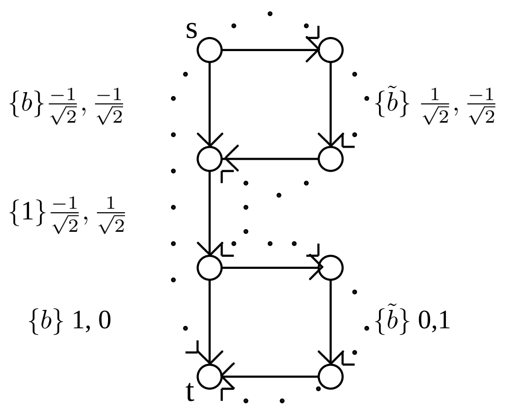
$$|0\rangle \otimes (\frac{1}{\sqrt{2}}|0\rangle + \frac{-1}{\sqrt{2}}|1\rangle) \otimes \frac{1}{2}|0\rangle.$$

A $\mathcal{E}_{j,k}$*-term* in a tensor graph is a maximal induced tensor subgraph between a node of height $j-1$ and a node of height k. We also require that the horizontal indegree of the node at height $j-1$ be zero and that the horizontal outdegree of the node at height k be zero. For the graph we considered above there are two $\mathcal{E}_{1,2}$-terms and two $\mathcal{E}_{2,3}$-terms but only one $\mathcal{E}_{1,3}$-term corresponding to the whole figure.

Colors are used to handle controlled-not layers. A color c and its anticolor $\tilde{c}$ satisfy the following multiplicative properties: $c{\cdot}c = \tilde{c}{\cdot}\tilde{c} = 1$ and $c{\cdot}\tilde{c} = 0$. Given two distinct colors b and c we have $b{\cdot}c = c{\cdot}b$ and $\tilde{b}{\cdot}c = c{\cdot}\tilde{b}$. If a is a product of colors and anticolors not involving the color b or $\tilde{b}$ and c is another product of colors we have $a(bc) = (ab)c$. We consider formal sums of products of complex numbers times colors. We require complex numbers to commute with colors and require colors and anticolors to distribute, i.e., if a, b, c are colors or anticolors then $a \cdot (b + c) = a \cdot b + a \cdot c$ and $(b + c) \cdot a = b \cdot a + c \cdot a$. Finally, we require addition to work so that the above structure satisfies the axioms of an $\mathbf{C}$-algebra. Given a tensor graph G denote by $\mathcal{A}_G$ the $\mathbf{C}$-algebra above. Since

$$(a \cdot a) \cdot \tilde{a} = \tilde{a} \neq 0 = a \cdot (a \cdot \tilde{a})$$

this algebra is not associative. However, in the sums we will consider the terms will never have more than two positions where a color or its anticolor can occur, so the products we will consider are associative. Using our our earlier encoding for the elements of $\mathbf{C}$ which could appear in a $QACC$ computation, it is straightforward to use sequence coding to get a $\mathrm{TC}^{(0)}$ encodings of the relevant elements of $\mathcal{A}_G$. As an example of how colors affect amplitudes, consider the following picture:



The amplitude of $|1,0,0\rangle$ in the left hand dotted path is $b \cdot \frac{-1}{\sqrt{2}} \cdot 1 \cdot \frac{-1}{\sqrt{2}} \cdot b \cdot 1 = 1/2$ using commutativity and $b^2 = 1$. Its amplitude in the right hand dotted path would be zero because of the last vertical edge. However, vectors such as $|0,0,1\rangle$ would have nonzero amplitude in the right hand dotted path. Nevertheless, the amplitude of any vector $|\vec{x}\rangle$ in any path other than the dotted ones from s to t will be 0 as $b \cdot \tilde{b} = 0$. More formally, we define the amplitude of an $|\vec{x}\rangle$ in a vertical edge as equal to the left amplitude times the color product in the edge if $\vec{x}$ is $|0\rangle$ and equal to the right amplitude times the color product in the edge if $\vec{x}$ is $\vec{1}$. The amplitude of a vector $|x_1, \ldots, x_j\rangle$ in a path in a tensor graph is the product over k from 1 to j of the amplitude of the vectors $|x_k\rangle$ in the vertical edge of height k. The amplitude of a vector $|x_j, \ldots, x_k\rangle$ in an $\mathcal{E}_{j,k}$-term is the sum of its amplitude in its paths. The amplitude of a vector $|x_1, \ldots, x_{p(n)}\rangle$ in a tensor graph G is defined to be the sum of its amplitudes in G's $\mathcal{E}_{1,p(n)}$-terms.

As we will be interested in families of tensor graphs $\{G_n\}$, corresponding to our circuit families we want to look at those families with a certain degree of uniformity. We say a family of tensor graphs $\{G_n\}$ is *color consistent* if: (1) the number of colors for edges of the same height is bounded by a constant k with respect to n, (2) the number of heights in which a given color/anticolor can appear is exactly two (colors and their anticolors must appear on the same heights), (3) each color product at the same height is of the form $\prod_{i=0}^{k} l_i$ where l_i must be either a color c_i or $\tilde{c_i}$ (it follows there are 2^k possible color products for edges at a given height). We say that a color/anticolor is *active* at a given height if the height is at or after the first height at which the color/anticolor occurs and is below the height of its second occurrence. The family is further said to be *log-color depth* if the number of active colors/anticolors of a given height is log-bounded.

Theorem 4.1 *Let $\{F_n\}$ be a family of QACC operators and let $\{\langle\vec{z}_n|\}$ a family of observables. (1) There is a color-consistent family of tensor graphs of width $2^{2^{2^t}}$ and polynomial size representing the output amplitudes of $U_1 \cdots U_t|\vec{z}_n\rangle$ where U_i are the layers of F_n. (2) If $\{F_n\}$ is in $QACC^{\log}_{pl}$ then the family of tensor graphs will be of log-color depth. (3) If $\{F_n\}$ is in $QACC^{\log}_{gates}$ then the number of paths from the source to the terminal node is polynomially bounded.*

Proof. The proof is by induction on t. In the base case, $t = 0$, we do not multiply any layers, and we can easily represent this as a tensor graph of width 1. Assume for $j < t$ that $U_j \cdots U_1|\vec{x}, 0^{p(n)}\rangle$ can be written as color consistent tensor graph of width $2^{2^{2^t}}$ and polynomial size. There are two cases to consider: In the first case the layer is a tensor product of matrices $M_1 \otimes \cdots \otimes M_\nu$ where the M_k's are Toffoli gates, one qubit gates, or fan-out gates (since $\mathrm{QAC}_{wf}^{(0)}$ =QACC); in the second case the layer is a controlled-not layer.

For the first case we "multiply" U_t against our current graph by "multiplying" each M_j in parallel against the terms in our sum corresponding to M_j's domain, say $\mathcal{E}_{j',k'}$. If $M_j = \begin{pmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{pmatrix}$ with domain $\mathcal{E}_{j'}$ is a one-qubit gate, then we multiply the two amplitudes in each vertical edge of height j' in our tensor graph by M_j. This does not effect the width, size, or number of paths through the graph. If M_j is a Toffoli gate, then for each term S in $\mathcal{E}_{j',k'}$ in our tensor graph we add one new term to the resulting graph. This term is added by adding a horizontal edge going out from the source node of S followed by the new $\mathcal{E}_{j',k'}$-term followed by a horizontal edge into the terminal node of S. The new term is obtained from the old one by setting to 0 the left hand amplitudes of all edges in S of height between j' and $k'-1$ and then if α, γ is the amplitude of an edge of height k' in the new term we change it to $\gamma - \alpha, \alpha - \gamma$. This new term adjusts the amplitude for the case of a $|1\rangle^{\otimes(k'-j'-1)}$ vector in $\mathcal{E}_{j',k'-1}$ tensored with either a $|0\rangle$ or $|1\rangle$. This operation increases the width of the new tensor graph by the width of the $\mathcal{E}_{j',k'}$-term for each $\mathcal{E}_{j',k'}$-term in the graph. Since the original graph has width $2^{2^{2(t-1)}}$ there are at most this many starting and ending vertices for such terms. So there at most $(2^{2^{2(t-1)}})^2$ such terms. Each of these terms has width at most $2^{2^{2(t-1)}}$. Thus, the new width is at most

$$2^{2^{2(t-1)}} + (2^{2^{2(t-1)}})^2 \cdot 2^{2^{2(t-1)}} < 2^{2^{2t}}.$$

Notice this action adds one new path through the $\mathcal{E}_{j',k'}$ part of the graph for every existing one.

Now suppose M_j is a fan-out gate, let S be a $\mathcal{E}_{j',k'}$-term in our tensor graph and let e be any vertical edge in S in $\mathcal{E}_{k'}$. Suppose e has amplitude α for $|0\rangle$ and amplitude γ for $|1\rangle$. In the new graph we change the amplitude of e to $\alpha, 0$. We then add a horizontal edge out of the source node of S followed by a new $\mathcal{E}_{j',k'}$-term followed by a horizontal edge into the terminal node of S. The new term is obtained from S by changing the amplitude for edges in $\mathcal{E}_{k'}$ with amplitudes α, γ in S to $0, \gamma$. The amplitudes of the non-$\mathcal{E}_{k'}$ edges in this term are the reverse of the corresponding edge in S, i.e., if the edge in S had amplitude δ, ζ then the new term edge would have amplitude ζ, δ. The same argument as in the Toffoli case shows the new width is bounded by $2^{2^{2t}}$ and that this action adds one new path through the $\mathcal{E}_{j',k'}$ part of the graph for every existing one.

For the case of a controlled-not layer, suppose we have a controlled-not going from line i onto line j. Let $c, \bar{c}$ be a new color, anti-color pair not yet appearing in the graph. Let e_i be a vertical edge of height i in the graph and let C_i, α_i, γ_i be respectively its color product and two amplitudes. Similarly, let e_j be a vertical edge of height j in the graph and C_j, α_j, γ_j be its color product and two amplitudes. In the new graph we multiply c times the color product of e_i and e_j and change the amplitude of e_i to $\alpha_i, 0$. We then add a horizontal edge going out from the starting node of e_i, followed by a vertical edge with values $C_i \cdot \tilde{c}, 0, \gamma_i$ followed by a horizontal edge into the terminal node of e_i. In turn, we add a horizontal edge going out of the starting node of e_j, followed by a vertical edge with values $C_j \cdot \tilde{c}, \gamma_i, \alpha_j$ followed by a horizontal edge into the terminal node of e_j. We handle all other controlled gates in this layer in a similar fashion (recall they must go to disjoint lines). We add at most a new vertex of a given height for every existing vertex of a given height. So the total width is at most doubled by this operation and $2 \cdot 2^{2^{2(t-1)}} < 2^{2^{2t}}$. In the $\mathrm{QACC}_{pl}^{\log}$case, simulating a layer which is a Kronecker product of spaced controlled-not gates and identity matrices, notice we would at most add one to the color depth at any place. So if a controlled-not layer is a composition of $O(\log)$ many such layers it will increase the color depth by $O(\log)$. In the $\mathrm{QACC}_{gates}^{\log}$ case, notice that simulating a single controlled-not we add one new path for each existing path through the graph at each of the two heights affected. This gives three new paths on the whole subspace for each old one.

Since we have handled the two possible layer cases and the changes we needed to make only increase the resulting tensor graph polynomially, we thus have established the induction step and (1) and (2) of the theorem. For (3), observe for each multi-line gate we handle in adding a layer we at most quadruple the number of paths through the subspace where that gate applies. Since there are at most logarithmically many such gates, the number of paths through the graph increases polynomially. □

Theorem 4.2 *Let $\{G_n\}$ be a family of constant width color-consistent tensor graphs of vectors in $\mathcal{E}_{1,p(n)}$. Assume the coefficients of amplitudes in the $\{G_n\}$ can be encoded in $TC^{(0)}$ using our encoding scheme described earlier and that $\{G_n\}$ has log-color depth. Then the amplitude of any basis vector of $\mathcal{E}_{1,p(n)}$ in G_n is P/poly computable. If the number of paths through the graph from the source to the terminal node is polynomially bounded then the amplitude of any basis vector is $TC^{(0)}$ computable.*

Proof. Let G_n be a particular graph in the family and let $|\vec{x_n}\rangle$ be the vector whose amplitude we want to compute. Assume that all graphs in our family have fewer than k colors in any color product and have a width bounded by w. We will proceed from the source to the terminal node one height at a time to compute the amplitude. Since the width is w the number of $\mathcal{E}_1$-terms is at most w and each of these must have width at most w. Let $\alpha_{1,1}, \ldots, \alpha_{1,w}$ (some of which may be zero) denote the amplitudes in $\mathcal{A}_{G_n}$ of $|x_{n,1}\rangle$ in each of these terms. The $\alpha_{1,i}$ are each sums of at most w amplitudes times the color products of at most k colors and anticolors, so the encoding of these w amplitudes is $\mathrm{TC}^{(0)}$ computable. Because of the restriction on the width of G_n there are at most w many $\mathcal{E}_{1,j}$-terms, w^2 many $\mathcal{E}_{j,j+1}$-terms, and w many $\mathcal{E}_{1,j+1}$-terms. Fixing some ordering on the nodes of height j and $j+1$ let $\gamma_{j,i,k}$ be the amplitude of $|x_{n,j+1}\rangle$ in the $\mathcal{E}_{j,j+1}$-term with source the ith node of height j and with terminal node the kth node of height $j+1$. The amplitude is zero if there is no such $\mathcal{E}_{j,j+1}$-term. Then the amplitudes $\alpha_{j+1,1}, \ldots, \alpha_{j+1,w}$ of the $\mathcal{E}_{1,j+1}$-terms can be computed from the amplitudes $\alpha_{j,1}, \ldots, \alpha_{j,w}$ of the $\mathcal{E}_{1,j}$-terms using the formula

$$\alpha_{j+1,k} = \sum_{i=1}^{w} \alpha_{j,i} \cdot \gamma_{j,i,k}.$$

Thus $\alpha_{j+1,k}$ can be computed from the $\alpha_{j,i}$ using a polynomial sized circuit to do these adds and multiplies. Similarly, each $\alpha_{j,k}$ can be computed by polynomial sized circuits from the $\alpha_{j-1,k}$'s and so on. Since we have log-color depth the number of terms consisting of elements in our field times color products in a $\alpha_{j,k}$ will be polynomial. So the size of the $\alpha_{j,k}$'s $j \leq p(n)$, $k \leq w$ will be polynomial in the input $\vec{x}_n$. So the size of the circuits for each $\alpha_{j,k}$ where $j \leq p(n)$ and $k \leq w$ will be polynomial size. There is only one $\mathcal{E}_{1,p(n)}$-term in G_n and its amplitude is that of $|\vec{x}_n\rangle$, so this shows it has polynomial sized circuits.

For the $\mathrm{TC}^{(0)}$ result, if the number of paths is polynomially bounded, then the amplitude can be written as the polynomial sum of the amplitudes in each path. The amplitude in a path can in turn be calculated as a polynomial product of the amplitudes times the colors on the vertical edges in the path. Our condition on every color appearing at exactly two heights guarantees the color product along the whole path will be 1 or 0, and will be zero iff we get a color and its anticolor on the path. This is straightforward to check in $\mathrm{TC}^{(0)}$, so this sum of products can thus be computed in $\mathrm{TC}^{(0)}$ using Lemma 4.1. □

Corollary 4.3

(1) $\mathrm{EQACC}^{\log}_{pl} \subseteq \mathrm{NQACC}^{\log}_{pl} \subseteq$ P/Poly, and $\mathrm{BQACC}^{\log}_{\mathbf{Q},pl} \subseteq$ P/poly.

(2) $\mathrm{EQACC}^{\log}_{gates} \subseteq \mathrm{NQACC}^{\log}_{gates} \subseteq \mathrm{TC}^{(0)}$, and $\mathrm{BQACC}^{\log}_{\mathbf{Q},gates} \subseteq \mathrm{TC}^{(0)}$.

Proof. Given a a family $\{F_n\}$ of $\mathrm{QACC}^{\log}_{pl}$operators and a family $\{\langle \vec{z}_n|\}$ of states we can use Theorem 4.1 to get a family $\{G_n\}$ of log color depth, color-consistent tensor graphs representing the amplitudes of $F_n^{-1}|\vec{z}_n\rangle$. Note $\{F_n^{-1}\}$ is also a family of $\mathrm{QACC}^{\log}_{pl}$operators since Toffoli and fanout gates are their own inverses, the inverse of any one qubit gate is also a one qubit gate (albeit usually a different one), and finally a controlled-not layer is its own inverse. Theorem 4.2 shows there is a P/poly circuit computing the amplitude of any vector $|\vec{x}_n\rangle$ in this graph. This amounts to calculating

$$\langle \vec{x}_n|F_n^{-1}|\vec{z}_n\rangle = \langle \vec{z}_n|F_n|\vec{x}_n\rangle$$

If this is nonzero, then $|\langle \vec{z}_n|F_n|\vec{x}_n\rangle|^2 > 0$, and we know $\vec{x}$ is in the language. In the $\mathrm{BQACC}_{\mathbf{Q}}$ case everything is a rational so P/poly can explicitly compute the magnitude of the amplitude and check if it is greater than $3/4$. The $\mathrm{TC}^{(0)}$ result follows similarly from the $\mathrm{TC}^{(0)}$ part of Theorem 4.1. □

5. Discussion and Open Problems

A number of questions are suggested by our work.

- Is all of NQACC in $\mathrm{TC}^{(0)}$ or even P/Poly? We conjecture that NQACC is in $\mathrm{TC}^{(0)}$. As mentioned in the introduction, we have developed techniques that remove some of the important obstacles to proving this.
- Are there any natural problems in NQACC that are not known to be in ACC?
- What exactly is the complexity of the languages in EQACC, NQACC and $\mathrm{BQACC}_{\mathbf{Q}}$? We entertain two extreme possibilities. Recall that the class ACC can be computed by quasipolynomial size depth 3 threshold circuits [15]. It would be quite remarkable if EQACC could also be simulated in that manner. However, it is far from clear if any of the techniques used in the simulations of ACC (the Valiant-Vazirani lemma, composition of low-degree polynomials, modulus amplification via the Toda polynomials, etc.), which seem to be inherently irreversible, can be applied in the quantum setting. At the other extreme, it would be equally remarkable if NQACC and $\mathrm{NQTC}^{(0)}$ (or $\mathrm{BQACC}_{\mathbf{Q}}$ and $\mathrm{NQTC}^{(0)}$) coincide. Unfortunately, an optimal characterization of QACC language classes anywhere between those two extremes would probably require new (and probably difficult) proof techniques.
- How hard are the fixed levels of QACC? While lower bounds for QACC itself seem impossible at present, it might be fruitful to study the limitations of small depth QACC circuits (depth 2, for example).

Acknowledgments: We thank Cris Moore for pointing out an error in an earlier version of Theorem 4.1, and Bill Gasarch for helpful comments and suggestions.

References

[1] A. Barenco, C. Bennett, R. Cleve, D.P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J.A. Smolin, and H. Weifurter. Elementary gates for quantum computation. Phys. Rev. A **52**, pages 3457–3467, 1995.

[2] K. C. Chung and T. H. Yao. On Lattices Admitting Unique Lagrange Interpolations . Siam Journal of Numerical Analysis **14**, pages 735–743, 1977.

[3] P. Clote. On polynomial Size Frege Proofs of Certain Combinatorial Principles. In P. Clote and J. Krajicek, editors, *Arithmetic, Proof Theory, and Computational Complexity*, pages 164–184. Oxford, 1993.

[4] L. Fortnow and J. Rogers. Complexity Limitations on Quantum Computation. *Proceedings of 13th IEEE Conference on Computational Complexity*, pages 202–209, 1998.

[5] S. Fenner, F. Green, S. Homer, and R. Pruim. Quantum NP is hard for PH. *Proceedings of 6th Italian Conference on theoretical Computer Science*, World Scientific, Singapore, pages 241–252, 1998.

[6] Alexis Maciel and Denis Therien. Threshold Circuits of Small Majority-Depth. Information and Computation **146**. 55–83, 1998.

[7] Cristopher Moore and Martin Nilsson. Parallel Quantum Computation and Quantum Codes In Los Alamos Preprint archives (1998), quant-ph/9808027.

[8] Cristopher Moore. Quantum Circuits: Fanout, Parity, and Counting. In Los Alamos Preprint archives (1999), quant-ph/9903046.

[9] R. A. Nicolaides. On a class of finite elements generated by Lagrange interpolation. Siam Journal of Numerical Analysis **9**, pages 177–199, 1972.

[10] P. W. Shor. Polynomial-time algorithms for prime number factorization and discrete logarithms on a quantum computer. *SIAM J. Comp.*, 26:1484–1509, 1997.

[11] K.-Y. Siu and V Rowchowdhury. On optimal depth threshold circuits for multiplication and related problems. *SIAM J. Discrete Math.* **7**. 284–292, 1994.

[12] K.-Y. Siu and J Bruck. On the power of threshold circuits with small weights *SIAM J. Discrete Math.* **4**. 423–435, 1991.

[13] R. Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. *Proceedings of the 19th Annual ACM Symposium on Theory of Computing.* 77-82, 1987.

[14] T. Yamakami and A.C. Yao. $NQP_{\mathbf{C}} = co\text{-}C_{=}P$. To appear in *Information Processing Letters*.

[15] A. C.-C. Yao. On ACC and threshold circuits. In *Proceedings of the 31st Symposium on Foundations of Computer Science*, (1990), 619-627.

[16] A. C.-C. Yao. Quantum circuit complexity. In *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science*, pages 352–361, 1993.