On the Complexity of Quantum ACC

Frederic Green[1] Steven Homer[2] Christopher Pollett[3]

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 $\mathrm{AC}^{(0)}$, $\mathrm{ACC}[q]$, and ACC, denoted $\mathrm{QAC}^{(0)}_{wf}$, QACC[2], QACC respectively, define the same class of operators, leaving $q > 2$ as an open question. Our result resolves this question, implying that $\mathrm{QAC}^{(0)}_{wf} = \mathrm{QACC}[q] = \mathrm{QACC}$ for all q. We also prove the first upper bounds for QACC in terms of related language classes. We define classes of languages EQACC, NQACC (both for arbitrary complex amplitudes) and $\mathrm{BQACC}_{\mathbf{Q}}$ (for rational number amplitudes) and show that they are all contained in $\mathrm{TC}^{(0)}$. To do this, we show that a $\mathrm{TC}^{(0)}$ circuit can keep track of the amplitudes of the state resulting from the application of a QACC operator using a constant width polynomial size tensor sum. In order to accomplish this, we also show that $\mathrm{TC}^{(0)}$ can perform iterated addition and multiplication in certain field extensions.

February 21, 2000

[1]Department of Mathematics and Computer Science, Clark University, Worcester, MA 01610, fgreen@black.clarku.edu

[2]Computer Science Department, Boston University, Boston, MA 02215, homer@cs.bu.edu

[3]Department of Mathematics, University of California, Los Angeles, CA, cpollett@willow.math.ucla.edu

1 Introduction

Advances in quantum computation during 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 have lead 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 $\mathrm{AC}^{(0)}$ circuits) can add and subtract binary numbers. The class ACC extends $\mathrm{AC}^{(0)}$ by allowing modular counting gates. The class $\mathrm{TC}^{(0)}$, 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 $\mathrm{QAC}^{(0)}$, $\mathrm{QACC}[q]$, and QNC and compared their power to that of their classical counterparts. This paper is a contribution to this line of research.

For example, a quantum analog of $\mathrm{AC}^{(0)}$, defined by Moore and denoted $\mathrm{QAC}^{(0)}_{wf}$, 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 $\mathrm{ACC}[q]$ (i.e., ACC circuit families only allowing Mod_q gates) is $\mathrm{QACC}[q]$, defined similarly to $\mathrm{QAC}^{(0)}_{wf}$, 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 $\mathrm{QAC}^{(0)}_{wf} = \mathrm{QACC}[2] = \mathrm{QACC}$. This is in sharp contrast to the classical result of Smolensky [13] that says $\mathrm{ACC}^{(0)}[q] \neq \mathrm{ACC}^{(0)}[p]$ for any pair of distinct primes q, p, which implies that for any prime p, $\mathrm{AC}^{(0)} \subset \mathrm{ACC}^{(0)}[p] \subset \mathrm{ACC}$. 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 $\mathrm{QACC} \neq \mathrm{QACC}[q]$ for odd q. In this paper, we provide the missing ingredients to show that in fact $\mathrm{QACC} = \mathrm{QACC}[q]$ 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

[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.

the Quantum Fourier Transform (using complex q^{th} roots of unity), and encodings of base q digits using qubits.

We also prove the first upper bounds for language classes related to QACC. Roughly speaking, we show that QACC is no more powerful than $TC^{(0)}$ and that this holds for arbitrary complex amplitudes in the QACC circuits. To make this statement 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. 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 NQACC is in $TC^{(0)}$. Although the proof uses some of the techniques developed by Yamakami and Yao [14] to show that NQP_C = co-$C_=P$, 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 $TC^{(0)}$ circuit can keep track of the amplitudes of each possible resulting state as each layer is applied. After all layers have been applied, the $TC^{(0)}$ 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-sum," a new way to represent a collection of states. We use these sums to check (in $TC^{(0)}$) whether the amplitude of any particular vector is non-zero. Another problem that arises is that it is necessary to add and multiply a polynomial number of complex amplitudes. In this case, this in turn reduces to adding and multiplying polynomially many elements of a certain transcendental extension of the rational numbers. We show that $TC^{(0)}$ 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-sums 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 show EQACC, NQACC, and $BQACC_{\mathbf{Q}}$ are all contained in $TC^{(0)}$.

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 $TC^{(0)}$ are described.

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 follows:* $\wedge_0(U) = U$ *and for* $m > 0$, $\wedge_m(U)$ *is*

$$\wedge_m(U)(|x_1, \ldots, x_m, y\rangle) = \begin{cases} u_{y0}|\vec{x}, 0\rangle + u_{y1}|\vec{x}, 1\rangle & \text{if } \wedge_{k=1}^{m} x_k = 1 \\ |\vec{x}, y\rangle & \text{otherwise} \end{cases}$$

Let $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. *A* Tofolli gate *is an* $\wedge_m(X)$ *gate for some* $m \geq 0$.

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

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

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.

Barenco, et al. [1] show any $U \in U(2^n)$ can be built out of layers of one-qubit and $\wedge_1(X)$ gates. $M^{\otimes n}$ is the n-fold tensor 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* $F_n \in U(2^{n+p(n)})$, p *a polynomial, and each* F_n *is writable as a product of* $O(\log^k n)$ *layers, where a* layer *is a tensor product of one-qubit gates and Toffoli gates acting on disjoint sets of qubits. 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.*

$QACC$ *denotes* $QACC^{(0)}$ *and* $QACC[q]$ *denotes* $QACC^{(0)}[q]$.

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* $\mathrm{QAC}^{(0)}$ 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* $\mathrm{QAC}^{(0)}$ 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 $\mathrm{QAC}^{(0)}$ equvialence, the auxiliary bits must be returned to their original values in a computation.

It follows for any $\{F_n\} \in \mathrm{QAC}^{(0)}$ that F_n is writable as a product of finite number of layers. Moore [8] shows $\mathrm{QAC}^{(0)}_{wf} = \mathrm{QACC}[2] = \mathrm{QACC}$. 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 $n = |x|$. Let $\vec{z} = z_1, \ldots, z_{p(n)}$.*

1. *E·$\mathcal{C}$ is the class of languages L such that for some $\{F_n\} \in \mathcal{C}$ and $\vec{z}$ a fixed binary vector, $m := |\langle x, \vec{z}|F_n|x, 0^{p(n)}\rangle|^2$ 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 $\vec{z}$ a fixed binary vector, $x \in L$ iff $|\langle x, \vec{z}|F_n|x, 0^{p(n)}\rangle|^2 > 0$.*

3. *B·$\mathcal{C}$ is the class of languages L such that for some $\{F_n\} \in \mathcal{C}$ and $\vec{z}$ a fixed binary vector, $x \in L$ if $|\langle x, \vec{z}|F_n|x, 0^{p(n)}\rangle|^2 > 3/4$ and $x \notin L$ if $|\langle x, \vec{z}|F_n|x, 0^{p(n)}\rangle|^2 < 1/4$.*

It follows E·$\mathcal{C} \subseteq$ N·$\mathcal{C}$ and E·$\mathcal{C} \subseteq$ B·$\mathcal{C}$. We are interested in E·QACC, B·QACC$_{\mathbf{Q}}$, and N·QACC which we abbreviate as EQACC, BQACC$_{\mathbf{Q}}$, and NQACC. Let $|\Psi\rangle := F_n|x, 0^{p(n)}\rangle$. Notice that $|\langle x, \vec{z}|F_n|x, 0^{p(n)}\rangle|^2 = \langle\Psi|P_{|x,\vec{z}\rangle}|\Psi\rangle$, where $P_{|x,\vec{z}\rangle}$ is the projection matrix onto $|x, \vec{z_n}\rangle$. We could allow in our definitions measurements of up to polynomially many such projection observables and not affect our results below concerning EQACC, BQACC$_{\mathbf{Q}}$, and NQACC. 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 mentions some closure properties of TC$^{(0)}$ computable functions useful in proving NQACC$\subseteq$ 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

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

Using the above functions we describe a way to do sequence coding in $\mathrm{TC}^{(0)}$. Let $\beta_{|t|}(x,w) := \lfloor (w \dot{-} \lfloor w/2^{(x+1)|t|} \rfloor \cdot 2^{(x+1)|t|})/2^{x|t|} \rfloor$. Let $B = 2^{|\max(x,y)|}$. So 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 $(w)_1 := \beta_{\lfloor \frac{1}{2}|w| \rfloor - 1}(0, \beta_{\lfloor \frac{1}{2}|w| \rfloor}(0,w))$ and $(w)_2 := \beta_{\lfloor \frac{1}{2}|w| \rfloor - 1}(0, \beta_{\lfloor \frac{1}{2}|w| \rfloor}(1,w))$. We can encode a poly-lengthed, $\mathrm{TC}^{(0)}$ 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. $\mathrm{TC}^{(0)}$ can use the $\mathrm{TC}^{(0)}$ circuits for natural numbers to compute both the polynomial sum and polynomial product of a sequence of $\mathrm{TC}^{(0)}$ 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:

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

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 denote F_2 by F, being 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 Tranform 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 the result of Barenco et al. [1] mentioned just before Definition 2.2, 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 $\mathrm{QAC}^{(0)}$, 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 $\mathrm{QAC}^{(0)}$-equivalent. Moore's definition 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 input 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 $x_i^{(k)}$ $(0 \le k \le \lceil \log q \rceil - 1)$ 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.

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, $0 \le r \le q-1$, 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. 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 $\widehat{\text{MOD}}_{q,r}$ 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 $\widehat{\text{MOD}}_{q,r}$, this collection of OR gates outputs exactly the bits in the value of $\sum_{i=1}^{n} x_i \bmod q$. Call this sum 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 described above, thus clearing the auxiliary inputs. 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 $\text{QAC}^{(0)}$-equivalent. By Moore's result, MOD_q is $\text{QAC}^{(0)}$-reducible to F. Hence F_q is $\text{QAC}^{(0)}$-reducible to F.

Conversely, arrange each block of $\lceil \log q \rceil$ input bits to an F_q^{-1} 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. For the i^{th} input-bit block, set the first $\lceil \log q \rceil - 1$ bits to 0, and the last bit to the input bit x_i. Now the i^{th} output of the F_q^{-1} circuit will be $(x_i - b) \bmod q$. This yields 1 or -1 (mod q) if $x_i \neq b$, and 0 if $x_i = b$. By (reversibly) squaring this output digit, and reducing the result mod q, we obtain $x_i \oplus b$ (along with some auxiliary bits). The input blocks of bits can now be restored to their original setting by applying F_q. The resulting circuit then simulates F, looking at the appropriate output wires. The squaring and reduction mod q operations are constant depth because they involve a fixed number of 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 $\text{QACC} = \text{QACC}[2] \subseteq \text{QACC}[q]$. □

4 Upper Bounds

In this section, we prove that NQACC $\subseteq$ TC$^{(0)}$ and that BQACC$_{\mathbf{Q}} \subseteq$ TC$^{(0)}$.

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 $|\langle x, \vec{z}|U_1 \cdots U_t|x, 0^{p(n)}\rangle|^2 > 0$ with a 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) $\langle \lceil \lambda_0 \rceil, \ldots, \lceil \lambda_{d-1} \rceil \rangle$ where $\lceil \lambda_j \rceil$ 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

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

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. Fix a common denominator u for the elements in $E \cup \{\beta_m \cdot \beta_l\}$. We take a common denominator 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 the elements of 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 $\langle\langle a_{\vec{k_j}}, k_{1j}, \ldots, k_{mj}\rangle\rangle$ means 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 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) For sums, first form the list $L1 = \langle f(0,x), \ldots, f(p(|x|),x)\rangle$ and then 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, closure under sums and max_i to find the length of the longest $f(i,x)$. To flatten the list 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 take 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 $Z[A]$ is isomorphic to the polynomial ring $Z[y_1, \ldots, y_m]$ under the natural map which takes α_j to y_j. We view our encodings $f(i,x)$ as m-variate polynomials in $Z[y_1, \ldots, y_m]$. We will describe for any p' a circuit that will work 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 $g(i,x)$ which consists of the sequence of polynomially many integer values which result from evaluating the polynomial encoded by $f(i,x)$ at the points $(i_1, \ldots, i_m) \in \mathbf{R}^m$ 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 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 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 Therorem 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 $k(j,x)$ will be 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 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 $\mathrm{TC}^{(0)}$.

(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 $\mathrm{TC}^{(0)}$ 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 straightfoward 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 one (1) we can compute the encoding e_j of $(\sum_{i=1}^{p(|x|)} s_{ij})$ in $\mathrm{TC}^{(0)}$. So the desired answer $\langle\langle r_0, e_0\rangle, \cdots, \langle r_0, e_{d-1}\rangle\rangle$ is in $\mathrm{TC}^{(0)}$.

For products, note that $\prod_{i=1}^{p(|x|)} f(i,x)$ is

$$\sum_{j_1 < d, \ldots, j_{p(|x|)} < d} \lambda_{1j_1} \cdots \lambda_{p(|x|)j_{p(|x|)}} \beta_{j_1} \cdots \beta_{j_{p(|x|)}}$$

If $\mathrm{TC}^{(0)}$ can compute the summands, the code of the whole sum is in $\mathrm{TC}^{(0)}$ by the first part of (2) since there are polynomially many summands each of which is computable in $\mathrm{TC}^{(0)}$. $\lambda_j = \prod_{i=1}^{p(|x|)} \lambda_{ij_i}$ is in $\mathrm{TC}^{(0)}$ since the numerator is a polynomial product of elements of F and the power u is raised to in the denominator is a polynomial sum of the exponents in the denominators of the λ_{ij_i}'s. The β_{j_i}'s are complex number so commute with each other. Thus, $\prod_{i=1}^{p(|x|)} \beta_{j_i} = \prod_{t=1}^{m} \beta_j^{t_j}$. To compute $\beta_j^{t_j}$, let ζ_{vw} be such that $\beta_w\beta_j = \sum_v \zeta_{vw}\beta_v$. These ζ_{vw}'s exist since the β_v's are a basis of G. So

$$\beta_j^{t_j} = \sum_{v_{t_j-1}=0}^{d-1} (\cdots (\sum_{v_2=0}^{d-1} (\sum_{v_1=0}^{d-1} \zeta_{jv_1})\zeta_{v_1v_2}) \cdots \zeta_{v_{t_j-2}v_{t_j-1}})\beta_{v_{t_j-1}}.$$

Since d is finite this can be rewritten so that the coefficients one needs to compute to get the encoding of $\beta_j^{t_j}$ are a polynomial sum of a polynomial product, and thus, in $\mathrm{TC}^{(0)}$. The product $\prod_{t=1}^{m} \beta_j^{t_j}$ is then a finite product so will be straightforward to compute in $\mathrm{TC}^{(0)}$, completing the proof of closure under products. □

The vectors that F_n for $\{F_n\} \in \mathrm{QAC}_{wf}^{(0)}$ =QACC act on live in a $2^{n+p(n)}$ dimensional space $\mathcal{E}_{1,n+p(n)}$ space which is a tensor product of the 2-dimensional spaces $\mathcal{E}_1, \ldots \mathcal{E}_{n+p(n)}$. i.e., these spaces are 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 sum* in the space $\mathcal{E}_{j,k}$ is a sum containing terms which are either (1) products of elements in G with basis vectors of $\mathcal{E}_{j,k}$ or (2) the tensor products of tensor sums from $\mathcal{E}_{j,j_1-1}$, $\mathcal{E}_{j_1,j_2-1}$, ..., $\mathcal{E}_{j_n,k}$. As an example,

$$.5|111\rangle + (.25|1\rangle \otimes (.25|10\rangle + (-.30)|11\rangle))$$

is a tensor-sum in $\mathcal{E}_{1,3}$. In this sum, $|111\rangle$ has amplitude .435 and $|110\rangle$ has amplitude .0625. Using our encoding for $\alpha \in G$, we can give an encoding for tensor sums using brackets '[]',

parentheses '()', and braces '{}'. We use codes $\ulcorner 0 \urcorner = 0$, $\ulcorner 1 \urcorner = 1$, $\ulcorner \{ \urcorner = 2$, $\ulcorner \} \urcorner = 3$, $\ulcorner [\urcorner = 2$, $\ulcorner] \urcorner = 3$, $\ulcorner (\urcorner = 4$, and $\ulcorner) \urcorner = 5$. The codes for elements of G are six plus the codes described earlier. It is easy to verify closure under sums and products for these new codes. A vector $\alpha|x\rangle$ is the sequence as $\langle \ulcorner \{ \urcorner, \alpha, x, \ulcorner \} \urcorner \rangle$ where x is 0 or 1. The tensor-product of two tensor sums given by sequences $\langle \sigma_1 \rangle$, $\langle \sigma_2 \rangle$ is the sequence $\langle \ulcorner [\urcorner, \sigma_1, \sigma_2, \ulcorner] \urcorner \rangle$. The sum of two tensor sums $\langle \sigma_1 \rangle$, $\langle \sigma_2 \rangle$ is the sequence $\langle \ulcorner (\urcorner, \sigma_1, \sigma_2, \ulcorner) \urcorner \rangle$. Thus, our tensor sum example above can be written as the sequence:

$$\langle \ulcorner (\urcorner, \ulcorner [\urcorner, \ulcorner [\urcorner, \ulcorner \{ \urcorner, \ulcorner .5 \urcorner, 1, \ulcorner \} \urcorner, \ulcorner \{ \urcorner, \ulcorner 1 \urcorner, 1, \ulcorner \} \urcorner, \ulcorner] \urcorner, \ulcorner \{ \urcorner, \ulcorner 1 \urcorner, 1, \ulcorner \} \urcorner, \ulcorner] \urcorner, \ulcorner [\urcorner, \ulcorner \{ 1 \ulcorner .25 \urcorner, 1 \ulcorner \} \urcorner, \ulcorner (\urcorner, \ulcorner [\urcorner, \ulcorner \{ \urcorner, \ulcorner .25 \urcorner, 1, \ulcorner \} \urcorner, \ulcorner \{ \urcorner, \ulcorner 1 \urcorner, 0, \ulcorner \} \urcorner, \ulcorner] \urcorner, \ulcorner [\urcorner, \ulcorner \{ \urcorner, \ulcorner .30 \urcorner, 1, \ulcorner \} \urcorner, \ulcorner \{ \urcorner, \ulcorner 1 \urcorner, 1, \ulcorner \} \urcorner, \ulcorner] \urcorner, \ulcorner) \urcorner, \ulcorner] \urcorner, \ulcorner) \urcorner \rangle.$$

The *width* of a tensor-sum is the maximum number of unclosed open parenthesis in a left-right scan of an initial subsequence of the sum. For instance, the above sum has width 2. The *height* of a symbol in a tensor-sum sequence without parentheses is the number of open brackets minus closed brackets to the left of the symbol. Height can be $\mathrm{TC}^{(0)}$ computed using $\dot{-}$ and iterated addition. The *height* of a tensor-sum is defined to be the maximum height of a symbol in it. We extend the definition of height of a symbol to the case where we have parentheses by saying the height of a symbol is the difference between the number of left minus right brackets plus the heights of the first tensor sum in each closed non-nested pair of parentheses to its left and on the same branch as the symbol. For fixed k we can use the fact $\mathrm{TC}^{(0)}$ is closed under polynomial sums and *cond* to inductively define a function which computes the height of the ith element of a tensor-sum of width less than k. A $\mathcal{E}_{j,k}$*-term* in a tensor sum is a subsequence which begins at a '[' of height j and ends at the first ']' of height k or first '[' of height $k+1$ to its right.

Lemma 4.2 *Let s be a constant-width tensor sum of vectors in $\mathcal{E}_{1,p(n)}$. Then the amplitude of any basis vector of $\mathcal{E}_{1,p(n)}$ in s is $TC^{(0)}$ computable.*

Proof. The proof is by induction on w. When w is 0, s is just the tensor product of polynomial of $\alpha|0\rangle$'s and $\gamma|1\rangle$'s. So only one basis vector is being represented and its amplitude can be calculated using the fact $\mathrm{TC}^{(0)}$ is closed under products. Assume we have a function $f_{w-1}(s', v)$ which computes the amplitude of the basis vector v in any tensor sum s' of vectors in $\mathcal{E}_{1,p(n)}$ of width up to $w-1$. Let $v_{i,j}$ denote the projection of v onto the subspace $\mathcal{E}_{i,j}$. Suppose s is a tensor sum of width w. We can compute the location of first open parenthesis and its matching closed parenthesis in $\mathrm{TC}^{(0)}$ using the μ-function. We can thus compute the subsequence between these two position. Each of the two summands s_1, s_2 is a sum of width less than w in some $\mathcal{E}_{i,j}$. By padding as neccessary we can use our f_{w-1} function to compute the amplitudes α_1 and α_2 in s_1 and s_2 of $v_{i,j}$. We might need to pad since f_{w-1} takes inputs which are sums in the whole space $\mathcal{E}_{1,p(n)}$. The tensor sum s has the form $s_0 \otimes (s_1 + s_2) \otimes s_3$ where either or both of s_0 and s_3 many not be present. Since both s_0 and s_3 have width at most $w-1$ by using padding again and f_{w-1} we can compute the amplitudes α_0 and α_3 in s_0 and s_3 of $v_{1,i-1}$ and $v_{j+1,p(n)}$ respectively. Then the amplitude of v in all of s is just $\alpha_0 \cdot (\alpha_1 + \alpha_2) \cdot \alpha_3$. Thus, using the fact that $\mathrm{TC}^{(0)}$ is closed under addition

and multiplication, we can use f_{w-1} to make a function $f_w(s,v)$ that computes the amplitude of the basis vector v in any width w tensor-sum s of vectors in $\mathcal{E}_{1,p(n)}$. □

Theorem 4.1 *The output of $U_t \cdots U_1|\vec{x}, 0^{p(n)}\rangle$ can be written as a width $2^{2^{2t}}$ tensor sum of polynomial size whose encoding is computable by a $TC^{(0)}$ function g of x.*

Proof. The proof is by induction on t. In the base case the input is a width 1 tensor sum, since $|\vec{x}, 0^{p(n)}\rangle = (\otimes_{i=1}^{n}|x_i\rangle) \otimes (|0\rangle^{\otimes p(n)})$ is a basis element of $\mathcal{E}_{1,2^{n+p(n)}}$. It is polynomial sized and is easily seen to be $\mathrm{TC}^{(0)}$ computable from x. Assume for $j < t$ that $U_j \cdots U_1|\vec{x}, 0^{p(n)}\rangle$ can be written as $\mathrm{TC}^{(0)}$ computable width $2^{2^{2j}}$ tensor sum of polynomial size. The layer U_t by definition 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 $\mathrm{QAC}^{(0)}_{wf}$ =QACC). To multiply U_t against our current sum we multiply 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 this would amount to replacing vectors $\alpha|0\rangle$ in our tensor sum by $(u_{00}\alpha|0\rangle + u_{01}\alpha|1\rangle)$ and vectors $\gamma|1\rangle$ in our tensor sum by $(u_{10}\gamma|0\rangle + u_{01}\gamma|1\rangle)$. This can at most increase the width of the sum by one and by Lemma 4.1 this replacement is $\mathrm{TC}^{(0)}$ computable. If M_j is a Toffoli gate, we replace each term S in $\mathcal{E}_{j',k'}$ in our tensor sum with

$$S + (\delta - \gamma)|1\rangle^{\otimes(k'-j'-1)} \otimes |0\rangle + (\gamma - \delta)|1\rangle^{\otimes(k'-j'-1)} \otimes |1\rangle$$

Here γ is the sum of the amplitudes of each possible way to get the vector $|1\rangle^{\otimes(k'-j'-1)} \otimes |0\rangle$ in S and δ is the sum of the amplitudes of each possible way to get the vector $|1\rangle^{\otimes(k'-j'-1)} \otimes |1\rangle$ in S. Computing γ and δ is in $\mathrm{TC}^{(0)}$ by Lemma 4.2. This operation increases the width of the new tensor sum by at most 2 for each $\mathcal{E}_{j',k'}$-term. There are at most $2^{2^{2(t-1)}}$ such terms. So the new width is at most old width + new terms which is less than $2^{2^{2(t-1)}} + 2 \cdot 2^{2^{2(t-1)}} < 2^{2^{2t}}$. Lastly, if M_j is a fan-out gate, we view any term S in $\mathcal{E}_{j',k'}$ in our tensor sum as a term S' in $\mathcal{E}_{j',k'-1}$ tensored with a sum of $\alpha|0\rangle$'s and $\gamma|1\rangle's$. S is replaced in the output by a sum of new terms one for each $\alpha|0\rangle$ and $\gamma|1\rangle$. In the case of an $\alpha|0\rangle$, the new term is just $S' \otimes \alpha|0\rangle$. In the case of an $\gamma|1\rangle$, the new term is $S'' \otimes \gamma|1\rangle$ where S'' is obtained from S' by replacing $|1\rangle$'s with $|0\rangle$'s and vice versa. The output of performing this operation is essentially copying and bit-swapping so can be done in $\mathrm{TC}^{(0)}$. There are at most $2^{2^{2(t-1)}}$ $\mathcal{E}_{j',k'}$-terms S and the number of things S' is tensored with and the width of S can also both be bounded by $2^{2^{2(t-1)}}$. So the new width is bounded by $2^{3 \cdot 2^{2(t-1)}}$ which is less than $2^{2^{2t}}$.

Since we have handled each possible gate type and these actions only increase the resulting tensor-sum polynomially, we thus have established the induction step and the theorem. □

Corollary 4.3 *$EQACC \subseteq NQACC \subseteq TC^{(0)}$ and also $BQACC_{\mathbf{Q}} \subseteq TC^{(0)}$.*

Proof. Theorem 4.1 shows we can compute in $\mathrm{TC}^{(0)}$ a constant-width tensor sum s which encodes the amplitudes of the vectors which result from multiplying a $QACC$ operator against a starting configuration. To compute whether $|\langle x_1 \cdots x_n z_1 \cdots z_{p(n)} | F_n | \vec{x}, 0^{p(n)} \rangle|^2 > 0$, amounts to computing the amplitude of the vector $|\vec{x}, \vec{z}\rangle$ in s. We can do this by Lemma 4.2. In $\mathrm{TC}^{(0)}$ we can then easily check if this is non-zero. In the $\mathrm{BQACC}_{\mathbf{Q}}$ case everything is a rational so $\mathrm{TC}^{(0)}$ can explicitly compute the magnitude of the amplitude and check if it is greater than $3/4$. □

5 Discussion and Open Problems

The position of the languages defined by QACC families, between ACC and $\mathrm{TC}^{(0)}$, make QACC attractive as a model for highly parallelizable quantum computation. A number of questions are suggested by our work.

- 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 $\mathrm{TC}^{(0)}$ (or $\mathrm{BQACC}_{\mathbf{Q}}$ and $\mathrm{TC}^{(0)}$) 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).

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.