Algebraic and Logical Emulations of Quantum Circuits

Kenneth Regan[1], Amlan Chakrabarti[2], and Chaowen Guan[1]

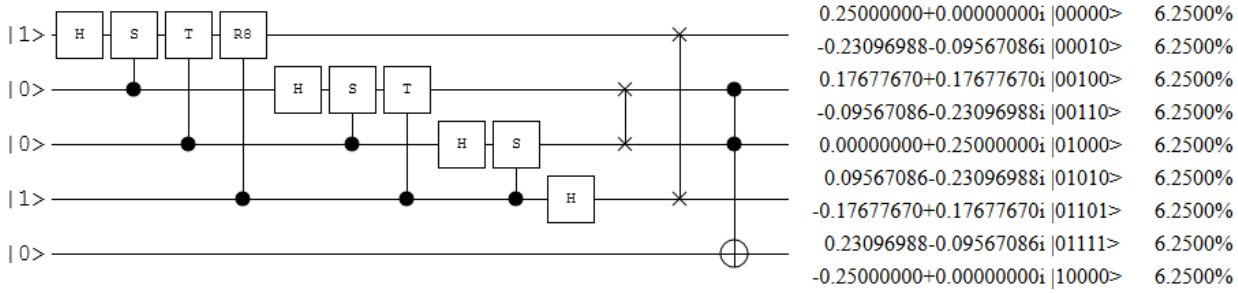
[1] University at Buffalo (SUNY)
[2] University of Calcutta

Abstract. Quantum circuits exhibit several features of large-scale distributed systems. They have a concise design formalism but behavior that is challenging to represent let alone predict. Issues of scalability—both in the yet-to-be-engineered quantum hardware and in classical simulators—are paramount. They require sparse representations for efficient modeling. Whereas *simulators* represent both the system's current state and its operations directly, *emulators* manipulate the images of system states under a mapping to a different formalism. We describe three such formalisms for quantum circuits. The first two extend the polynomial construction of Dawson et al. [1] to (i) work for any set of quantum gates obeying a certain "balance" condition and (ii) produce a single polynomial over any sufficiently structured field or ring. The third appears novel and employs only simple Boolean formulas, optionally limited to a form we call "parity-of-AND" equations. Especially the third can combine with off-the-shelf state-of-the-art third-party software, namely *model counters* and #SAT *solvers*, that we show capable of vast improvements in the emulation time in natural instances. We have programmed all three constructions to proof-of-concept level and report some preliminary tests and applications. These include algebraic analysis of special quantum circuits and the possibility of a new classical attack on the factoring problem. Preliminary comparisons are made with the `libquantum` simulator[2–4].

1 A Brief But Full QC Introduction

A *quantum circuit* is a compact representation of a computational system. It consists of some number m of *qubits* represented by *lines* resembling a musical staff, and some number s of *gates* arrayed like musical notes and chords. Here is an example created using the popular visual simulator [5]:

Fig. 1. A five-qubit quantum circuit that computes a Fourier transform on the first four qubits.



The circuit C operates on $m = 5$ qubits. The input is the binary string $x = 10010$. The first $n = 4$ qubits see most of the action and hold the nominal input $x_0 = 1001$ of length $n = 4$, while the fifth qubit is an *ancilla* initialized to 0 whose purpose here is to hold the nominal output bit. The circuit has thirteen *gates*. Six of them have a single *control* represented by a black dot; they activate if and only if the control receives a 1 signal. The last gate has two controls and a *target* represented by the parity symbol $\oplus$ rather than a labeled box. Called a *Toffoli gate*, it will set the output bit if and only if *both* controls receive a 1 signal. The two gates before it merely *swap* the qubits 2 and 3 and 1 and 4, respectively. They have no effect on

the output and are included here only to say that the first twelve gates combine to compute the *quantum Fourier transform* QFT_4. This is just the ordinary discrete Fourier transform F_{16} on $2^4 = 16$ coordinates.

The actual output $C(x)$ of the circuit is a *quantum state* $\mathcal{Z}$ that belongs to the complex vector space $\mathbf{C}^{32}$. Nine of its entries in the *standard basis* are shown in Figure 1; seven more were cropped from the screenshot. Sixteen of the components are absent, meaning $\mathcal{Z}$ has 0 in the corresponding coordinates. Despite the diversity of the nine complex entries $\mathcal{Z}_L$ shown, each has magnitude $|\mathcal{Z}_L|^2 = 0.0625$. In general, $|\mathcal{Z}_L|^2$ represents the probability that a *measurement*—of all qubits—will yield the binary string $z \in \{0,1\}^5$ corresponding to the coordinate L under the standard ordered enumeration of $\{0,1\}^5$. Here we are interested in those z whose final entry z_5 is a 1. Two of them are shown; two others (11101 and 11111) are possible and also have probability $\frac{1}{16}$ each, making a total of $\frac{1}{4}$ probability for getting $z_5 = 1$. Owing to the "cylindrical" nature of the set B of strings ending in 1, a measurement of just the fifth qubit yields 1 with probability $\frac{1}{4}$.

Where does the probability come from? The physical answer is that it is an indelible aspect of nature as expressed by quantum mechanics. For our purposes the computational answer is that it comes from the four gates labeled H, for *Hadamard gate*. Each supplies one bit of nondeterminism, giving four bits in all, which govern the sixteen possible outcomes of this particular example. It is a mistake to think that the probabilities must be equally spread out and must be multiples of $1/2^h$ where h is the number of Hadamard gates. Appending just one more Hadamard gate at the right end of the third qubit line creates nonzero probabilities as low as $0.0183058\ldots$ and as high as $0.106694\ldots$, each appearing for four outcomes of 24 nonzero possibilities. This happens because the component values follow wave equations that can *amplify* some values while reducing or zeroing the amplitude of others via *interference*. Indeed, the goal of quantum computing is to marshal most of the amplitude onto a small set of desired outcomes, so that measurements—that is to say, quantum *sampling*—will reveal one of them with high probability.

All of this indicates the burgeoning complexity of quantum systems. Our original circuit has 5 qubits, 4 nondeterministic gates, and 9 other gates, yet there are $2^5 = 32$ components of the vectors representing states, 32 basic inputs and outputs, and $2^4 = 16$ branchings to consider. Adding the fifth Hadamard gate creates a new fork in every *path* through the system, giving 32 branchings. The whole circuit C defines a 32×32 matrix U_C in which the I-th row encodes the quantum state Φ_I resulting from computation on the standard basis vector $x = e_I$. The matrix is *unitary*, meaning that U_C multiplied by its conjugate transpose U_C^* gives the 32×32 identity matrix. Indeed, U_C is the product of thirteen simpler matrices U_ℓ representing the respective gates ($\ell = 1, \ldots, s$ with $s = 13$). Here each gate engages only a subset of the qubits of arity $r < m$, so that U_ℓ decomposes into its $2^r \times 2^r$ unitary *gate matrix* and the identity action (represented by the 2×2 identity matrix I) on the other $m - r$ lines. Here are some single-qubit gate matrices:

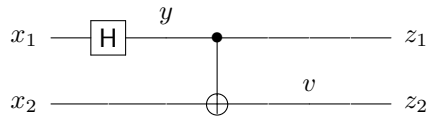
$$\mathsf{H} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad \mathsf{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \mathsf{Y} = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad \mathsf{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad \mathsf{S} = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}, \quad \mathsf{T} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}, \quad \mathsf{R}_8 = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/8} \end{bmatrix}.$$

The *phase gate* S is also called R_2 and the *twist gate* T is also called R_4. Multiplying a quantum state by a unit scalar changes none of the magnitudes so the unconditional global *phase shift* by the scalar has no effect on outcomes. The conditional phase shift on a 1 signal embodied by the lower-right entry of all these matrices does matter, and unconditionally effects a rotation of the qubit line through its own 2×2 space.

Adding *controls* is one way to extend effects to other qubits. The gate X, which is also called the NOT gate for the negation it effects on the classical bit corresponding to the qubit line, yields the controlled forms CX (aka. CNOT) and CCX (aka. Tof for the Toffoli gate) at left and right:

$$\mathsf{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \mathsf{CS} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & i \end{bmatrix}, \quad \mathsf{Tof} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

Note that the NAND of u, v equals $\mathsf{Tof}(u, v, 1)$ so the Toffoli gate alone can efficiently code any classical computation. The controlled versions of the Z, T, and $\mathsf{R_8}$ gates, as also employed in our example circuit, are defined similarly. Here is a simple circuit using one Hadamard gate and the standard symbol for CNOT:



Here x_1, x_2 and z_1, z_2 are variables that stand for the input and output viewed as binary strings, while y and v are internal variables following the scheme laid out in section 3 below. On input $a = 00$, which is represented by the vector $(1, 0, 0, 0)$ in $\mathbf{C}_4$, the final quantum state $\mathcal{Z}$ is $\frac{1}{\sqrt{2}}(1, 0, 0, 1)$. The outcomes $b = 00$ and $b = 11$ are possible with probability $\frac{1}{2}$ each, but the outcomes 01 and 10 are impossible. So the two qubits are constrained to give the same binary value. When a quantum state $\mathcal{Z}$ cannot be written as a tensor product of two other vectors—as implied here by the lack of probabilistic independence of the output bits b_1 and b_2—it is *entangled*.

The matrix of the Swap gate is the special 4×4 permutation matrix that fixes coordinates 1 and 4 corresponding to the strings 00 and 11 but sends 2 to 3 and 3 to 2 so that 01 switches with 10. We have now completed the description of the circuit C in Figure 1. Two quirks of the notation must be noted:

- If the qubits engaged by U_j are not contiguous, then we may not get U_j as a literal tensor product of its gate matrix and identity matrices, but we can write $U_j = P^{-1}U'_jP$ where U'_j is such a product and P induces a permutation of the qubit lines in the manner of the swap matrix above.
- We read the circuits left-to-right but matrices are composed and applied to column vectors right-to-left.

Above we have ignored the swapping in the former point by writing the matrices with *control lines* first. The second point looks ignorable because all our matrices happen to be symmetric, but when the gate Y is included this is no longer so. The single-qubit gates $\mathsf{X}, \mathsf{Y}, \mathsf{Z}$ (plus I for completeness) are called *Pauli gates*, while adding H, S, and CZ creates a basis for the set of *Clifford gates*, which also include CNOT but not Tof, CS, T, or $\mathsf{R_8}$. Adding any one of the latter four gates creates a gate set that is *universal* in that any quantum circuit C can have its states $\Phi = C(a)$ approximated to arbitrary precision with comparable efficiency by a circuit C' using only gates from the set. Indeed, the set consisting of just H and Tof is universal in a weaker sense that encodes the real and complex parts of Φ separately and approximates the probabilities of the basic outcomes. Two minimal universal sets that keep $r \leq 2$ are $\{\mathsf{H}, \mathsf{CS}\}$ and $\{\mathsf{H}, \mathsf{T}, \mathsf{CNOT}\}$.

To complete our description of quantum computation, we note that to compute a mapping f from inputs $x \in \{0, 1\}^n$ to outcomes $z_0 \in \{0, 1\}^q$ it is conventional to use $m = n + m_0 + q$ qubits with the latter $m_0 + q$ initialized to 0 and produce outcomes of the form $x \cdot 0^{m_0} \cdot y$. This can be effected by using only the initial n qubits plus m_0 more qubits for "scratch space" for the main computation, then using q CNOT gates to copy out desired results into the last q qubits—in like manner to how the fifth qubit was employed in our example. Finally one can restore the first $n + m_0$ qubits to their initial state by repeating the conjugate transposes of the gates of the main computation in reverse order. The extra cost of the "copy" and "uncompute" steps after the main computation stays within a linear factor in terms of the visible hardware—that is, the count $m + s$ of qubits and gates (presuming bounded arity r of any one gate). The convention works even if f is not one-to-one.

We can implement traditional definitions of computing functions $f : \{0, 1\}^n \to \{0, 1\}^q$ and deciding membership in languages L with high probability of success or correctness. Say f is (*quantum*) *efficiently computable* if there is a polynomial $p(n)$ such that for all $\epsilon > 0$ and all n we can build a circuit $C_{n,\epsilon}$ of size $m + s \leq p(n + \log(1/\epsilon))$ such that for all inputs x, the measurement gives $x0^{m_0}f(x)$ with probability at least $1 - \epsilon$. Applying this criterion to the 0-1 valued characteristic functions of L defines the quantum complexity class BQP standing for "Bounded-error Quantum Polynomial time." Functions and even multi-valued mappings f are also said to belong to BQP with the understanding that computing $f(x) \mapsto z$ is equivalent to deciding $L_f = \{(x, y) : \text{some legal value } z \text{ has } y \text{ as an initial substring}\}$. But the primary utility may come from the quantum ability to *sample* the joint distribution of (x, z) where z is a legal value of f and see what the induced distribution on x may highlight.

2 A System View: Simulation and Emulation

What is the "true" complexity of the system we have described? We can first ask about its effective size:

(a) Is it the simple count $m + s$ of qubits and gates, perhaps multiplied by some function of r?
(b) Or is it the dimension 2^m of the ambient space, and/or the size 2^h of the branching paths?
(c) Or is the "real size" something in-between?

Two ideas of "in-between" are whether the layout size $m \times s$ of the grid should be counted and whether entanglements render its effective cross-section as more than the number m of qubits. The former still leaves the size polynomial in n while it seems hard to argue the latter higher than the number of possible binary entanglements, which is $O(m^2)$. Moreover, there is a dearth of natural mathematical functions between polynomial (or quasi-polynomial: $n^{(\log n)^{O(1)}}$) and exponential ($2^{n^{\Omega(1)}}$).

The polynomial versus exponential divide is burnished by Shor's algorithm [6] for factoring n-bit integers M. Its quantum circuits C_n sit inside a non-quantum program loop that samples integers $a < M$, runs $C(a, M)$, measures all qubits of the resulting state $\mathcal{Z}$, and tries to infer a factor of M from the outcome b. The circuits C_n are relatively simple; they use the gates in Figure 1 plus one call to the quantum Fourier transform on n qubits, and are known to have size no worse than slightly above $O(n^2)$. Classical algorithms, however, are conjectured to require time 2^{n^c} with c not much smaller than $1/3$ (see [7]) on *many* instances M. The security of RSA and many other cryptographic protocols depends on this hardness assumption or a similar one for the discrete logarithm problem, which is also solved by Shor's algorithm. If one subscribes both to the classical hardness of factoring and to our circuit modeling correctly metering quantum mechanics as a true theory of nature, the upshot is:

Nature can compute efficiently in ways that symbolic operations are far from emulating.

This is stronger than concluding that classical physical systems cannot approximate the quantum sampling $C(a, M) \mapsto b$ as above. It says that no human system of notation for describing quantum circuits can execute their calculations with comparable efficiency—nor any reasonable efficiency. Somewhere we must face an exponential blowup. To riff on the enigmatic final words in Latin of Umberto Eco's novel *The Name of the Rose*, all the QC notations we can hold are "bare names" that miss the pristine mechanism through which the quantum "rose" abides unsullied by notation.

Of course, this is a *challenge* for the description of computational systems. Formal analysis tools for computing systems are legion but this system—after we needed only a few pages to specify its components fully and indicate how signals are processed—spits fire saying it can't be done. There are two levels of demands we can make on the description, one stronger and the other weaker:

(a) At any time t, it should describe with high fidelity the state of the system at time t.
(b) On any input, it should reproduce with high fidelity the *final* state of the system.

We identify these capabilities with the distinction made by Häner at al. [8] between (a) a *simulator* and (b) an *emulator* of quantum circuits. As described in section 2 of their papers, simulators to date have directly manipulated the gate matrices under sparse representations and using hardware systems dedicated to such matrix operations. We read the essence as being that the simulators can on request deliver a "census" of active system elements and their microstates as they stand after multiplying j of the matrices, for any j from 0 to s. As such, they need to grapple with the exponential blowup at the first stages where it is in force. *Emulators* are freed from having to comply with such requests, and in particular are entitled to ignore the synchrony imposed by j and the $m \times s$ grid view of the circuit. Their prime goal is to postpone the blowup as long as possible and perhaps—in enough important cases—to avoid it.

In this paper we present three formalisms for describing quantum circuits that begin with the capability of simulation but promote non-quantum, non-physical manipulations that may yield faster emulation. The first two represent circuits C by single polynomials P_C and Q_C over various finite fields or rings, the third by

simple quantifier-free Boolean formulas ϕ_C. In their brute-force rendition they can perform simulation, and we show that the Boolean one is competent on simple hardware. The former two may connect to existing software packages that perform algebraic manipulations such as equation solving and reduction using Gröbner bases, such as *Singular* [9, 10]. The latter immediately connects to existing *model counters* and #SAT *solvers* such as *sharpSAT* [11] and *Cachet* [12–14]. Both P_C and ϕ_C have size $O(m + s)$. The blowup is entirely handed off to the algebraic system or solution counter.

The first express conversion to (sets of) polynomials was by Dawson et al. [1] and programmed by Gerdt and Severyanov [15]. It applied only to the universal set $\{ \mathsf{H}, \mathsf{CNOT}, \mathsf{Tof} \}$ of gates with ± 1 entries, except for remarks in [1] about "mixed mode (mod-2/mod-8) arithmetic." Bacon, van Dam, and Russell [16] tailored a construction to (singly and doubly) controlled phase-changing gates modulo various values of K, like those for $K = 4, 8, 16$ in the last section's example, plus the Fourier transform F_K. All of these constructions constitute proofs that BQP polynomial-time reduces to the complexity class $\#\mathsf{P}$ of functions and its decision analogue PP [17]. Fortnow and Rogers [18] improved this to place BQP inside a "low" subclass of PP called AWPP and characterized by one evaluation of the difference $f(x)-g(x)$ of two functions $f, g \in \#\mathsf{P}$ that obey a further restriction. We claim to give the first "global" presentation of polynomial constructions P_C, including a trick to produce polynomials Q_C of bounded degree at the cost of using more variables. Our translation into Boolean formulas ϕ_C appears to be entirely new as we discuss later. But to be sure, the constructions are elementary enough that they could have been done soon after these papers or even the seminal gate papers [19, 20]. Perhaps motives are stronger now amid greater development of algebraic system-analysis tools, the increasing success of heuristic SAT and #SAT solvers, and ramped-up efforts at high-performance emulation of quantum systems represented recently by [8, 21, 22].

Our formalisms employ the *path* concept promoted by Richard Feynman, whose papers [23, 24] are often jointly credited with Deutsch [25, 26] as originating quantum computing. Each path contributes a unit amount to a term in the matrix product $U_s U_{s-1} \cdots U_2 U_1$ applied to an input vector x when everything is multiplied out. It begins in some row I, $0 \le I \le 2^m - 1$, of the column vector x and enters column I of U_1. It may exit U_1 at any row J for which $U_1[J, I] \neq 0$ and then enters column J of U_2. Finally, it exits U_s at some row L, which we call the final *location* of the path. The matrices of most elementary gates satisfy the following condition:

Definition 1. *A quantum gate matrix is* balanced *if all nonzero entries have the same magnitude.*

If r_ℓ is the magnitude for each gate ℓ, then the final path value is a unit complex scalar $e^{i\theta}$ divided by $R = r_1 \cdot r_2 \cdots r_s$ which is independent of the path. The θ is the *phase* of the path. We can identify a path's *current phase and location* at any stage j upon exiting U_j. Tracking this explicitly for all j and all paths would constitute what we have called a simulation. The freedom of emulation we desire will come from (i) treating paths individually without the idea of "stage j," (ii) combining and transforming their representations into the small objects P_C and ϕ_C, and hopefully (iii) further algebraic and formal manipulations that may break the association with gates and matrices but gather the final locations and phases into outcomes more efficiently than by census.

The contributions of the present paper are:

- Main theorems showing how to convert quantum circuits C—using all common gate sets—into polynomials P_C, Q_C and Boolean formulas ϕ_C that produce amplitudes $\langle a| C |b\rangle$ for every basic input a and output b.
- Further theorems building polynomials $\mathcal{P}_C, \mathcal{Q}_C$ and formulas Φ_C that compute probabilities, foster partial measurements, and enable an exact emulation of quantum sampling.
- Examples of algebraic and logical calculations, including a new proof of the *Gottesman-Knill theorem* that draws an especially fine line between BQP and P.
- Very preliminary experiments showing that on reasonable cases up to several dozen qubits and nondeterministic gates, #SAT solvers not only soundly beat intelligent brute force but *scale* much better as the problem size increases.

3 The Common Architecture

The two main structural parameters are the number m of qubits and the *minimum phase denominator* K, which is the minimum K such that all matrix entries $re^{i\theta}$ have θ an integer multiple of $2\pi/K$. We assume K is a power of 2, $K = 2^k$. The architecture maintains the phase $J \in \{0, \ldots, K-1\}$ and location in $\{0,1\}^m$ of any one path. The following elements are common to the descriptions by polynomials and by formulas:

- All variables range over 0 and 1 only. (Theorem 4 will allow values in $\mathbf{Z}_4$.)
- The input $a_1 \cdots a_m \in \{0,1\}^m$ designates the one initial location for all paths.
- Variables $X = x_1, \ldots, x_m$ are placeholder variables assigned $a_1, \ldots, a_m$ when an input $a \in \{0,1\}^m$ is given.
- Variables $Z = z_1, \ldots, z_m$ can be fully or partially substituted by *targets* b_i for $i \in I \subseteq \{1, \ldots, m\}$, in order to define desired and undesired final locations.
- The *value(s)* of the polynomial or formula represent only a path's phase, not its location.
- The current location of a path is embodies by *line designators* $u_1, \ldots, u_m$ which are not variables but rather references to already-existing variables. Initially they refer to $x_1, \ldots, x_m$, respectively.
- Upon finishing the circuit, algebraic terms or logical equations are added to force $z_1, \ldots, z_m$ to be equal to the variables designated by $u_1, \ldots, u_m$ upon finishing the circuit. They may also be substituted by some or all target values $b_1, \ldots, b_m$ prefatory to executing measurements.
- *Free variables* $Y = y_1, \ldots, y_h$ represent nondeterministic bits (from Hadamard and some other gates) and are the main variables over which solutions are counted. When a designator u_i refers to a free variable y_j it denotes a fork in the path tied to line i but affecting every location.
- *Forced variables* $V = v_1, \ldots, v_s, \ldots$ are placed on qubit lines i and become the new u_i when placed. Their values are forced—else the path being built is zeroed out or cancels itself out or fails a satisfaction filter.
- Extra variables Υ in terms Υ_ℓ may be used to enforce constraints by making failure fork the path into the whole range of phases—equally weighted—so that its net contribution is zero.
- The polynomial's *value* may—or may not—be maintained in *phase variables* $W_\ell = \{(w_j)_\ell\}$ at any juncture ℓ, with $1 \le j \le k$ in binary. These variables are also forced, and the designations "V" and "W" will be flexible in context.

We will also consider liberalizing $u_1, \ldots, u_m$ to refer to arbitrary subterms, not just variables (or their negations). In the simple Hadamard plus CNOT example in section 1 we could replace the forced variable v by its algebraic value $2x_2y - x_2 - y$ or its Boolean value $x_2 \oplus y$ instead of creating a polynomial term or Boolean clause to constrain v to have that value. Our general theorem, however, uses only fresh single variables y as the internal *annotations* on qubit lines. Absorbing this architecture may be enough to treat section 5 as self-justifying and skip over the full proof of the general theorem in the next section.

4 General Simulation

To state the main theorem, we call a ring $\mathcal{R}$ *adequate* for a quantum circuit C of min-phase denominator K if there is a 1-to-1 mapping h from the K-th roots of unity into $\mathcal{R}$ such that, letting $\omega = e^{2\pi i/K}$, we have $h(\omega^{I+J}) = h(\omega^I)h(\omega^J)$ for all $I, J \le K$. Since the range of h cannot include zero, $\mathcal{R}$ must have at least $K+1$ elements. In this multiplicative case, zero will annihilate terms that would denote impossible paths. Our theorem will also include a 1-to-1 function h' such that $h'(\omega^{I+J}) = h'(\omega^I) + h'(\omega^J)$ in a different ring $\mathcal{R}'$ while translating C into a polynomial Q_C over $\mathcal{R}'$. For simplicity we will fix $R' = \mathbf{Z}_K$, the integers modulo K, and fix $h'(\omega^I) = I$ as the mapping. Now zero is in the range: $0 = h'(1)$. We will use the additivity to make impossible paths yield a net-zero effect by having them cause a corresponding constraint expression γ to have value 1 rather than 0. In the Boolean case we will have k variables $W_\ell = (w_0, \ldots, w_{k-1})$ that represent the values $0, \ldots, K-1$ in binary notation, while an impossible path will correspond to a truth assignment that is immediately unsatisfying—that will satisfy none of the formulas $\phi_J = \phi_C[W_\ell = J]$ expressing that

the final value is J. The "indicators" $u_1, \ldots, u_m$ are not actual variables but rather references to variables in $X \cup Y \cup V$.

Finally, given a polynomial p in h variables, let $\#binsols(p)$ denote the number of zeros of the polynomial in $\{0,1\}^h$. If some variables in p are substituted by constants, we note that in square brackets and may subscript the non-substituted variables for emphasis. We similarly use $\#sat(\phi)$ for the number of satisfying assignments to a formula ϕ. The counting runs over $\{0,1\}^q$ where q is the number of non-substituted variables. If $q = 0$, then $\{0,1\}^q$ still has one member, so the count of solutions will be 1 or 0 according to whether the resulting constant p is zero or ϕ is reduced to the constant $\top$ (true) function. Although "V" will be unused in the proof for P_C and Q_C, we keep it in the theorem statement to cover usage in the next section.

Theorem 1. *There is an efficient uniform procedure that transforms any balanced m-qubit quantum circuit C of min-phase $K = 2^k$ with s gates of maximum arity r into constants R, R', polynomials P_C over an adequate ring and Q_C over $\mathbf{Z}_K$, and a Boolean formula ϕ_C—all with variables from V, W, X, Y, Υ, Z as in section 3—such that for all $a, b \in \{0,1\}^m$:*

$$\langle a| C |b\rangle = \frac{1}{R} \sum_{J=0}^{K-1} \omega^J \#binsols_{V,Y}(P_C[X = a, Z = b] - h(\omega^J)) \tag{1}$$

$$= \frac{1}{R'} \sum_{J=0}^{K-1} \omega^J \#binsols_{V,Y,\Upsilon}(Q_C[X = a, Z = b] - J) \tag{2}$$

$$= \frac{1}{R} \sum_{J=0}^{K-1} \omega^J \#sat_{V,Y}(\phi_C[X = a, Z = b, W = J]). \tag{3}$$

The objects P_C, Q_C, ϕ_C all have formulas of size $O(2^{2r}msk)$ and can be written down in time $\tilde{O}(m + 2^{2r}sk)$, where the $\tilde{O}$ means to ignore logarithmic factors coming from the variable labels.

Proof. We first describe the construction of P_C, then indicate the adjustments needed to produce Q_C and ϕ_C. On input a, let $U_s U_{s-1} \cdots U_2 U_1 a$ be the matrix computation. We will first treat each U_ℓ as a general $2^m \times 2^m$ matrix engaging all m qubits, then show how things simplify for arity r. We maintain a running polynomial P_ℓ in stages $\ell = 0, 1, 2, \ldots, s$. This will give $P_C = P_s E_C$ where

$$E_C = \prod_{i=1}^{m} (1 + 2u_i z_i - u_i - z_i),$$

using whatever variables (or alternatively subterms) are designated by $u_1, \ldots, u_m$ at the end. On 0-1 arguments, E_C gives 1 if $u_i = z_i$ for all i and 0 otherwise. We can also put $P_{C,\ell} = P_\ell E_C$ for any ℓ using the variables the u_i refer to on completing stage ℓ.

Initially, $P_0 = 1$, $R = 1$, and the indicators $u_1, \ldots, u_m$ refer to the variables $x_1, \ldots, x_m$, which themselves will be substituted by the binary arguments $a_1, \ldots, a_m$. If $s = 0$, i.e., if C has no gates, then we will have $u_i = x_i = a_i$ and $z_i = b_i$ for all i, yielding 1 if $b = a$ and 0 otherwise. Since we have $\langle a| I |b\rangle = 1$ when $b = a$ and 0 otherwise, (1)–(3) hold by the convention on counting over empty domains before the theorem statement.

As we add gates in stages $\ell = 1$ to s, we will maintain the invariant that paths from a to some current location L giving current phase J are in 1-to-1 correspondence with solutions to $P_C[X = a, Z = L] - h(\omega^J) = 0$. Suppose this is true for stage $\ell - 1$ (as initially when $\ell = 1$) and consider the matrix U_ℓ. Each path enters in some column L and exits in some row I. Using the m-bit binary code for each L, define the *indicator term*

$$t_L = \prod_{j:L_j=0} (1 - u_j) \prod_{j:L_j=1} u_j.$$

Then $t_L = 1$ if the current entering location given by $(u_1, \ldots, u_m)$ equals L and $t_L = 0$ otherwise. Let g stand for the current number of Y-variables allocated so far ($g = m(\ell - 1)$ before we simplify), allocate new variables $y_{g+1}, \ldots, y_{g+m}$, and for each possible I define

$$t_I = \prod_{j:I_j=0} (1 - y_{g+j}) \prod_{j:I_j=1} y_{g+j}.$$

For each I, L we either have $U_\ell[I, L] = 0$ or we have $U_\ell[I, L] = r\omega^d$ for some d, where r is independent of I, L by the balance condition. Multiply R by r and define P_ℓ to be $P_{\ell-1}$ multiplied by the term

$$p_\ell = \sum_{I,L:U_\ell[I,L]=r\omega^d \neq 0} t_I t_L h(\omega^d).$$

Finally, re-assign $u_1, \ldots, u_m$ to refer to the newly-created variables $y_{g+1}, \ldots, y_{g+m}$. This completes stage ℓ.

We claim that this preserves the invariant. The main point is that each 0-1 assignment to the variables previously designated by $u_1, \ldots, u_m$ and thew new variables $y_{g+1}, \ldots, y_{g+m}$ makes exactly one product $t_I t_L$ nonzero. So it corresponds to the path segment that enters at L and exits at I. If $U_\ell[I, L] = 0$ then any extension of that assignment will make P_C have value 0, so it cannot be a solution to any equation $P_C - h(\omega^J) = 0$. Else, the segment advances the phase of any path it extends by d. Hence for paths that are in location I and have current phase J after U_ℓ we know:

- There is some L and d such that the path came into U_ℓ by column L and $p_\ell = t_I t_L h(\omega^d)$. On entry it had phase ω^{J-d} with the exponent wrapped mod K.
- By the inductive invariant, there is a unique assignment to the variables appearing in $P_{C,\ell-1}[X = a, Z = L]$ that gives value $h(\omega^{J-d})$. Those are all variables in $P_{C,\ell}$ except $y_{g+1}, \ldots, y_{g+m}$.
- Setting $y_{g+1}, \ldots, y_{g+m} = I$ generates a solution to $P_{C,\ell}[X = a, Z = I] - h(\omega^J) = 0$.

Now we need to show that every solution uniquely defines a path:

- Given a solution to $P_{C,\ell}[X = a, Z = I] - h(\omega^J) = 0$, that solution must give some value L to the variables that were designated by $u_1, \ldots, u_m$ entering stage ℓ.
- We have $p_\ell \neq 0$ since the assignment is a solution.
- No assignment can give the sum in p_ℓ more than one nonzero summand, so p_ℓ evaluates to $h(\omega^d)$ where $U_\ell[I, L] = r\omega^d$, taking d and r from above.
- Hence the assignment induces a solution to $P_{C,\ell-1}[X = a, Z = L] - h(\omega^{J-d}) = 0$.
- By the induction invariant, the assignment induces a unique path from a to location L entering U_ℓ and with phase ω^{J-d}. (There may be other paths that converge in that location with the same phase, but they are induced by other assignments.)
- Forming $P_{C,\ell}$ from P_ℓ involved equating $y_{g+1}, \ldots, y_{g+m}$ to the respective variables $z_1, \ldots, z_m$, which were substituted by I in the equation, so the solution correctly induces the unique extension of the path into column L of U_ℓ and out row I.

This establishes the needed invariant after stage ℓ and completes the induction. So (1) holds. Before optimizing the polynomials P_ℓ obtained, we adapt this construction and proof to Q_C, ϕ_C and the analogous inductively defined Q_ℓ, ϕ_ℓ, which are this time based on $Q_0 = 0$ and $\phi_0 = \top$.

In forming and evaluating Q_C we are taking logarithms base ω so that products become sums. We can use the same indicator subterms t_L, t_I as above, except that the matrix gives an additive d not a multiplicative $h(\omega^d)$. Note that if $U[I, L] = 1$ we will get an additive 0. The issue is what to do with the cases $U[I, L] = 0$—what to use in place of the logarithm of 0? The answer is to allocate k variables $\upsilon_0, ..., \upsilon_{k-1}$, put $\Upsilon = \upsilon_0 + 2\upsilon_1 + \cdots 2^{k-1}\upsilon_{k-1}$, and define

$$q_\ell = \sum_{d,I,L:U_\ell[I,L]=r_\ell\omega^d \neq 0} t_I t_L \cdot d \quad + \sum_{I,L:U_\ell[I,L]=0} t_I t_L \cdot \Upsilon.$$

And now instead of multiplying R by $1/r_\ell$ we multiply it by something also involving k. The reason is that a path entering U_ℓ in column L and exiting in row I now induces 2^k assignments rather than just one, counting all those to $v_0, ..., v_{k-1}$. If $U_\ell[I, L] \neq 0$ the latter assignments are irrelevant because Υ is multiplied by 0 but the variables $v_0, ..., v_{k-1}$ are still present in the formula. (Unless, that is, U_ℓ is a matrix like $\mathsf{QFT_m}$ which has no zero entries—in which case we do nothing more.) If $U_\ell[I, L] = 0$ then the corresponding assignment c to the other variables multiplies Υ by 1. Now we can associate to c the K assignments to $v_0, ..., v_{k-1}$, producing K assignments c_J in all, $0 \leq J < K$. Each of these assignments gives a different final phase value J'. When we sum over those assignments, they augment each of the solution counts multiplying $\omega^{J'}$ in (2), giving a net-zero contribution to the whole sum equated to $\langle a| C |b\rangle$.

When using this trick multiple times one needs to use distinct suites Υ_ℓ of variables. The reason is that if the number n_c of violated constraints, each contributing +1, becomes a positive multiple of K, the contribution from $\sum_{J'=0}^{K-1} \omega^{n_c J'}$ in (2) is no longer zero but K. The number N_c of constraints will be at most $s + m$ and so the bump in formula size will be $O(sk + mk)$. We also need to multiply R by $\sqrt{K^{N_c}}$, but otherwise we can ignore all assignments inducing illegal paths while repeating the above correctness analysis.

We need to use the same trick to handle the final equations with the output variables z_j. Define

$$E'_C = \sum_{i=1}^{m} (u_i + z_i - 2u_i z_i)\Upsilon,$$

and finally,

$$Q_C = E'_C + \sum_{\ell=1}^{s} q_\ell.$$

Note that whereas the degree of P_C is linear in s, the degree of Q_C depends only on the maximum arity r—not even on k.

To define a Boolean formula ϕ_C we do not need this "Υ trick" but instead introduce suites $W_\ell = (w_{0,\ell}, \ldots, w_{k-1,\ell})$ to track the phase at each stage ℓ. In place of the "indicator terms" t_I, t_L we use subformulas defined as follows: For any column value $L \in \{0, 1\}^m$, u_L denotes the unique conjunction of signed literals $\pm u_i$ (over $i = 1$ to m) whose value is 1 on L and 0 for all $L' \neq L$. For instance, if $L = 01101$ then $u_L = (\bar{u}_1 \wedge u_2 \wedge u_3 \wedge \bar{u}_4 \wedge u_5)$. We denote row conjuncts y_I similarly using the newly allocated variables $y_{g+1}, \ldots, y_{g+m}$ defined as before. Entering stage ℓ of the circuit, we consider all possible phases $J_{\ell-1}$ coded by the variables $W_{\ell-1} = (w_{0,\ell-1}, \ldots, w_{k-1,\ell-1})$. For all pairs I, L we add clauses as follows:

- If $U_\ell[I, L] = 0$ then we add $\neg(u_L \wedge y_I)$, which becomes a clause of $2m$ disjoined literals.
- If $U_\ell[I, L] = r_\ell \omega^d$ then we add for $j = 0$ to $k - 1$ the clauses

$$(u_L \wedge y_I) \rightarrow (w_{j,\ell} = w_{j,\ell-1} \oplus F_d(W_{\ell-1})),$$

  where F_d is the fixed finite function true on all c such that $c + d$ causes a flip in bit j.

Note that F_d can be a function of the variables $w_{0,\ell-1}, \ldots, w_{j,\ell-1}$ alone. We can alternately consider that over $j = 0$ to $k - 1$ alone we have added the single clauses

$$w_{j,\ell} = w_{j,\ell-1} \oplus F'(u_1, \ldots, u_m, y_1, \ldots, y_m, w_0, \ldots, w_j),$$

where F' takes into account all the phases d that arise in the matrix entries $U_\ell[I, J]$ as specified by the values L for $u_1, \ldots, u_m$ and I for $y_{g+1}, \ldots, y_{g+m}$. Economizing F' will occupy much attention later, but for this proof we reason about F_d for all the u_L and v_I.

Finally we note that $y_{g+1}, \ldots, y_{g+m}$ become "$u_1, \ldots, u_m$" for the next stage if there is one, else we conjoin the clauses $\wedge_{i=1}^{m}(y_{g+i} = z_i)$ (or just substitute $z_1, \ldots, z_m$ directly). The last act is to add the clauses $\wedge_j \bar{w}_{j,0}$ and declare "W" in the theorem statement to refer to the terminal $w_{j,s}$ phase variables. Then V in the theorem statement ranges over $w_{j,\ell}$ for $1 \leq \ell \leq s - 1$ and Y ranges over variables $y_{g+i,\ell}$ introduced as "y_{g+i}" in the corresponding stages ℓ. (We will say more below.) This finishes the construction of ϕ_C.

To see that it is correct, first consider any path P from a to b whose phase changes by J. First we substitute $\boldsymbol{x} = a$ and $\boldsymbol{z} = b$ and $W_s = J$. In the base case $s = 0$ with empty circuit, P can only be a path from a to $b = a$ with $J = 0$. Then we have $W_s = W_0$ and substituting J gives $\top$ if $b = a$ and $J = 0$, $\bot$ otherwise. For $s \geq 1$, to P there is a unique assignment of row and column values

$$a = L_1, \quad I_1 = L_2, \quad \ldots, \quad I_{s-1} = L_s, \quad I_s = b$$

to literals designated "u_i" and "y_{g+i}" at each stage ℓ. For all $(I, L) \neq (I_\ell, L_\ell)$, all clauses $(u_I \wedge v_L) \to (\ldots)$ are vacuously satisfied. This leaves the clause

$$(u_{L_\ell} \wedge y_{I_\ell}) \to (w_{j,\ell} = w_{j,\ell-1} \oplus F_d(W_{\ell-1})),$$

where d is the phase of the nonzero entry $U_\ell[I, L]$. By induction, the values of $W_{\ell-1}$ in the assignment either have the phase $J_{\ell-1}$ of the path entering that stage or the assignment is already determined to be unsatisfying. These determine the value $F_d(W_{\ell-1})$ and hence collectively over j these clauses determine that W_ℓ must have the correct value $J_{\ell-1}+d$ modulo K, else they are not satisfied. Since the values of the variables in W_ℓ are forced, we have a unique continuation of a satisfying assignment. In the last stage, the current phase value must become J. Hence we have mapped P to one satisfying assignment of $\phi_C[X = a, Z = b, W = J]$ (with W_0 already zeroed).

Going the other way, suppose c is any satisfying assignment to ϕ_C (again with $W_0 = 0$). We argue that c maps uniquely to a path P_c. We get $a = L_1$ from the values assigned to X, then the values $L_2, \ldots, L_s$ of the other column entries, and finally the exit row I_s which gives a b. The values of phases along the path are likewise determined by the assignment and must be correct. Hence the assignment yields a unique path. The path must be legal: at any stage the left-hand side of one clause of the form $(u_J \wedge y_I) \to (\cdots)$ holds so its consequent must be made true. Thus the correspondence of counting paths and counting satisfying assignments is parsimonious for each phase value L, so the equation (3) follows.

It remains to reduce the objects P_C, Q_C, and ϕ_C down to the stated size, without changing the numbers of solutions or of satisfying assignments. For P_C, consider any qubit line i that is not involved in gate ℓ, so that U_ℓ acts as the identity on i. The product terms in p_ℓ involving line i divide into four groups with $u_i y_{g+i}$, $(1-u_i)(1-y_{g+1})$, $z u_i(1-y_{g+1})$, and $(1-u_i)y_{g+i}$, respectively. Because U_ℓ acts as the identity on line i, the latter two groups occur only for entries $U_\ell[I, L]$ that are 0, so they vanish. Since having $u_i = 0$ while $y_{g+1} = 1$ or vice-versa zeroes out the former two groups as well, any 0-1 solution to $P_C[X = a, Z = b] - h(\omega^J) = 0$ must have $y_{g+i} = u_i$. Hence without changing the number of binary solutions, we may for each such i substitute $y_{g+i} = u_i$, delete the terms for the vanishing groups, and make the factors on the surviving groups just u_i and $(1 - y_{g+i})$, respectively. Doing so cuts the size of p_ℓ down by a factor of 2^{m-r}. But since p_ℓ is a sum of 2^{2r} terms, each a product of $2r$-many factors, this is not yet good enough.

Again focusing on qubit line i, the remaining terms have the forms $y_{g+i}H_1$ and $(1-u_i)H_2$. Because U_ℓ acts as the identity on qubit i, every entry $U[I, L]$ where $I_i = L_i = 1$ equals the entry $U[I', L']$ where $I'_i = L'_i = 0$ with the other bits the same as in I and L. Hence terms in H_1 pair off with equal terms in H_2. We claim that we can replace the remaining terms in p_ℓ by just H_1 $(= H_2)$. Doing this does not add any new solutions, because if a solution makes $u_i = 1$ then the original p_ℓ got the same contribution from H_1 as it gets now (with H_2 being zeroed), and similarly for $u_i = 0$. Nor does doing this remove any solutions—nor does it remove all dependence on u_i because line i may be involved in $U_{\ell+1}$. It does allow us to avoid introducing the new variable y_{g+i}, so entering stage $\ell + 1$, u_i refers to the same variable as previously (and g counts only the new variables added). Applying this second process cuts the number of terms in p_ℓ down by another factor of (at least) 2^{m-r}, and also cuts the degrees of terms down from m to r. The polynomial P obtained by doing this for all stages thus has size $O(m + sk2^{2r})$ as claimed.

Similar remarks apply to Q_C with possible occurrences of Υ_ℓ absorbed into the overall size factor k for coefficients in $0 \ldots K - 1$. For ϕ_C, the size reduction boils down to saying that whenever I and I' vis-à-vis L and L' agree on the r qubit lines touched by the gate, their clauses can be identified, leaving at most 2^{2r} distinct clauses of size $O(k)$ added at stage ℓ. The rest of the size estimation is similar. Also in each case, the objects can be computed in time linear in their size in one "pass" that introduces the terms or clauses for each gate one-by-one. □

We will re-prove this theorem for all the gates covered in section 1. This will show more cases in which it is unnecessary to put a new variable y_{g+i} even for lines i involved in the gate. It will also increase the role of "V" via further cases in which a new variable is forced to have a certain argument value in any solution or satisfying assignment. Then it will be named v_{f+i} instead where f counts the forced line variables. The dedicated constructions will reveal structural properties of these gates and some particular points of elegance.

5 Simulation of Common Gates

Most of the gates have min-phase $K = 2$, 4, or 8. Indeed, $K = 2$ suffices for a universal set able to approximate probabilities and $K = 4$ for a set (namely H and CS) able to approximate all complex amplitudes. Adequacy of the target ring for P_C entails $-1 \neq 1$, so $1 + 1 \neq 0$, so 2 exists, though $2 = -1$ is possible. Accordingly we write -1 in place of $h(-1)$ but write $h(i)$ in place of i and so on. For Q_C, however, we prefer to write $K/2$ rather than specify 1 when $K = 2$ or 2 when $K = 4$, etc. In defining ϕ_C we avoid the double-subscripting in the phase variables $W_\ell = \{ w_{j,\ell} \}$ by letting p_ℓ distinguish the top of the circle from the bottom, q_ℓ the first and third quadrants, and r_ℓ the odd eighths from the even eighths. For instance, when $K = 8$ and J is the phase after stage ℓ, $p_\ell = 0$ means $J \in \{ 0, 1, 2, 3 \}$, $q_\ell = 0$ means $J \in \{ 0, 1, 4, 5 \}$, and $r_\ell = 0$ means $J \in \{ 0, 2, 4, 6 \}$.

Initializing $P = 1$, $Q = 0$, and $\phi = \top$, we describe running changes as gates are appended one-by-one in stages $\ell = 1, 2, 3, \ldots$ When we use ℓ as a subscript on a variable or term $T_\ell = t_0 + 2t_1 + \cdots 2^{k-1}t_{k-1}$ multiplying a constraint for Q, it means "the next available index in that category"—thought we could make it literally agree with the stage number by inserting dummies.

- Hadamard gate $\mathsf{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ on line i: Allocate a new top-phase variable p_ℓ, allocate a new free variable y_g on line i, and change the objects as follows:

$$\begin{aligned} P &*= (1 - 2u_i y_g) \\ Q &+= (u_i y_g K/2) \\ \phi &\wedge= (p_\ell = p_{\ell-1} \oplus (u_i \wedge y_g)). \end{aligned}$$

  Finally, multiply R by $\sqrt{2}$ and set $u_i = y_g$.

The remaining elementary gates are either deterministic or are expressible using Hadamard and deterministic gates. We now regard g as keeping a running count of all free variables up to h total. A deterministic change of location in qubit i (as opposed to phase) is reflected by a variable we denote by v_f rather than y_g being placed "on" line i so that u_i denotes it after the stage. This comes with a *constraint* of the form $v_f = E$ where E is 0-1 valued. Multiplying P by $(1 + 2Ev_f - v_f - E)$ kills paths that violate the constraint and preserves ones that obey it (with no phase change). Adding $T_\ell(v_f + E - 2Ev_f)$ has a similar effect for Q. Two alternatives are worth noting:

- We can add $(v_f = E)$ as a separate equation. Indeed, we can also keep the final $(z_i = u_i)$ equations separate. Then instead of one large P_C we have a system of small equations and a smaller P'_C, with only $P'_C - h(\omega^J)$ changing for different phases J.
- We can avoid introducing v_f and make u_i subsequently refer to E rather than a variable. It is OK for u_i to refer to arbitrary 0-1 valued subterms. The complexity of subsequent terms, including later expressions E', can be compounded, however.

For ϕ we can conjoin $(v_f = E)$ as a Boolean term, or alternatively introduce nothing and revise u_i to refer to E. Again this is OK above for H and below for all the other gates but can cause terms "on" the qubit lines to mushroom.

It is of course possible that the phase may change as well. Then we multiply P_C by other functions of v_f and E that either kill the path or implement the phase change. Gates with diagonal matrices can change

the phase but do not change the location and so do not allocate a v_f or change u_i in any case. Now we see the details for particular gates:

- Pauli gate $\mathsf{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ on qubit line i: Allocate a new line variable v_f on line i and implement the equation $v_f = 1 - u_i$ by:

$$\begin{aligned} P \mathrel{*}= & (u_i + v_f - 2u_i v_f) \\ Q \mathrel{+}= & \Upsilon_\ell (1 - u_i - v_f + 2u_i v_f) \\ \phi \mathrel{\wedge}= & (v_f = \neg u_i). \end{aligned}$$

  There is no change to other variables or to R. Finally, update $u_i = v_f$. The alternative is to substitute whatever subterm u_i referred to into $1 - u_i$ and let the "new u_i" refer to that.
- $\mathsf{CNOT} = \begin{bmatrix} \mathsf{I} & 0 \\ 0 & \mathsf{X} \end{bmatrix}$ with *control* on line i and *target* on line j: Allocate new v_f on line j and implement the equation $v_f = u_i \oplus u_j$, numerically $v_f = u_i + u_j - 2u_i u_j$, by:

$$\begin{aligned} P \mathrel{*}= & (1 - u_i - u_j - v_f + 2u_i u_j + 2u_i v_f + 2u_j v_f - 4u_i u_j v_f) \\ Q \mathrel{+}= & \Upsilon_\ell \cdot (2u_i u_j v_f - 2u_i u_j - 2u_i v_f - 2u_j v_f + u_i + u_j + v_f) \\ \phi \mathrel{\wedge}= & (v_f = u_j \oplus u_i). \end{aligned}$$

  Finally, update u_j to refer to v_f. Alternatives: revise u_j to refer to $u_i + u_j - 2u_i u_j$ with no change to P or to Q, and in the case of ϕ, call $u_i \oplus u_j$ the new u_j.
- Toffoli gate $\mathsf{Tof} = \begin{bmatrix} \mathsf{I} & 0 \\ 0 & \mathsf{CNOT} \end{bmatrix}$ with controls on i, j and target on k: Allocate new v_f on line k and implement the equation $v_f = u_k \oplus (u_i \wedge u_j)$, numerically $v_f = u_i u_j + u_k - 2u_i u_j u_k$, by:

$$\begin{aligned} P \mathrel{*}= & (1 - u_i u_j - u_k - v_f + 2u_i u_j u_k + 2u_i u_j v_f + 2u_k v_f - 4u_i u_j u_k v_f) \\ Q \mathrel{+}= & \Upsilon_\ell \cdot (2u_i u_j u_k v_f - 2u_i u_j u_k - 2u_i u_j v_f - 2u_k v_f + u_i u_j + u_k + v_f) \\ \phi \mathrel{\wedge}= & (v_f = u_k \oplus (u_i \wedge u_j)). \end{aligned}$$

  Finally, update u_k to refer to v_f. Alternatives: revise u_k to refer to $u_i u_j + u_k - 2u_i u_j u_k$ with no change to P or to Q, and in the case of ϕ, call $u_i u_j \oplus u_k$ the new u_k.

The Swap gate on qubits i, j can be implemented simply by interchanging u_i and u_j woth no growth in formulas. The *Fredkin gate* on i, j, k is the i-controlled swap of j, k and needs new forced variables v_j, v_k on the latter two lines with terms expressing $v_j = (u_i \vee u_j) \wedge (\bar{u}_i \vee u_k)$ and similarly for v_k.

Although Hadamard and Toffoli are radically different gates—with radically different changes to P and to Q—the updates to ϕ have the same form. In general we call a Boolean formula of the form

$$p' = p \oplus (\wedge_{i=1}^{j} u_i) \tag{4}$$

a *parity of AND equation*, or *pae* for short, of *order* j. It is elegant that H and Tof are both entirely coded by one *pae* of order 2, with the most overt difference being that the one for H introduces a free variable y_g whereas the one for Tof does not. The *pae* form is natural for carry propagation and adding controls as shown in what follows.

- Pauli gate $\mathsf{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ on qubit line i: Allocate new top-phase variable p_ℓ and do the following with no other change:

$$\begin{aligned} P \mathrel{*}= & (1 - 2u_i) \\ Q \mathrel{+}= & (K/2) u_i \\ \phi \mathrel{\wedge}= & (p_\ell = p_{\ell-1} \oplus u_i). \end{aligned}$$

– Pauli gate $\mathsf{Y} = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$ on qubit line i: Allocate v_f on line i, and for ϕ, allocate new top and quarter-phase variables p_ℓ and q_ℓ. Since this matrix is not symmetric, it is important to remember that the columns are $(1 - u_i)$ and u_i whereas the rows are $(1 - v_f)$ and v_f. Accordingly:

$$\begin{aligned} P &*= h(i)(1 - 2u_i)(u_i + v_f - 2u_i v_f) \\ Q &+= (K/4) - (K/2)u_i + \Upsilon_\ell(1 + 2u_i v_f - u_i - v_f) \\ \phi &\wedge= (v_f = \neg u_i) \wedge (q_\ell = \neg q_{\ell-1}) \wedge (p_\ell = p_{\ell-1} \oplus u_i \oplus q_{\ell-1}). \end{aligned}$$

The last conjunct for ϕ is not a *pae*, but since $\mathsf{Y} = i\mathsf{XZ}$, we can instead compose the actions for Z and X and i. The scalar multiplication by i is optional, but it seems helpful to say we are tracking phases of paths exactly especially when we use *conjugation* in the next section. To update ϕ for multiplication by i, make $q_{\ell+1} = \neg q_\ell$ and $p_{\ell+1} = p_\ell \oplus q_\ell$. For the conjugate multiplication by $-i$ the latter is $p_{\ell+1} = p_\ell \oplus \bar{q}_\ell$ instead. The matrix Y is self-adjoint, i.e., $\mathsf{Y}^* = \mathsf{Y}$, so we do not need to give conjugate forms. But for the gates that follow, we will indicate the conjugate forms P^*, Q^*, ϕ^*, which are likewise initialized to 1, 0, and $\top$, respectively.

– Phase gate $\mathsf{S} = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$: Allocate new p_ℓ, q_ℓ and do:

$$\begin{aligned} P &*= (1 - u_i + h(i)u_i) & P^* &*= (1 - u_i + h(-i)u_i) \\ Q &+= (K/4)u_i & Q^* &+= (3K/4)u_i \\ \phi &\wedge= (q_\ell = q_{\ell-1} \oplus u_i) & \phi^* &\wedge= (q_\ell = q_{\ell-1} \oplus u_i) \\ &\quad \wedge (p_\ell = p_{\ell-1} \oplus (u_i \wedge q_{\ell-1})) & &\quad \wedge (p_\ell = p_{\ell-1} \oplus (u_i \wedge \bar{q}_{\ell-1})). \end{aligned}$$

– $\mathsf{T} = \begin{bmatrix} 1 & 0 \\ 0 & \omega^{K/8} \end{bmatrix}$: Allocate all new p_ℓ, q_ℓ, r_ℓ with equations:

$$\begin{aligned} P &*= (1 - u_i + h(\omega^{K/8})u_i) & P^* &*= (1 - u_i + h(\omega^{7K/8})u_i) \\ Q &+= (K/8)u_i & Q^* &+= (7K/8)u_i \\ \phi &\wedge= (r_\ell = r_{\ell-1} \oplus u_i) & \phi^* &\wedge= (r_\ell = r_{\ell-1} \oplus u_i) \\ &\quad \wedge (q_\ell = q_{\ell-1} \oplus (r_{\ell-1} \wedge u_i)) & &\quad \wedge (q_\ell = q_{\ell-1} \oplus (\bar{r}_{\ell-1} \wedge u_i)) \\ &\quad \wedge (p_\ell = p_{\ell-1} \oplus (q_{\ell-1} \wedge r_{\ell-1} \wedge u_i)) & &\quad \wedge (p_\ell = p_{\ell-1} \oplus (\bar{q}_{\ell-1} \wedge \bar{r}_{\ell-1} \wedge u_i)). \end{aligned}$$

The controlled gate CS forms a universal set with H that also approximates complex amplitudes. So does T when added to H and either CNOT or CZ. Now we show the progression of controlled phase gates, which leave R unchanged and allocate no variables except phase variables for ϕ:

– $\mathsf{CZ} = \begin{bmatrix} \mathsf{I} & 0 \\ 0 & \mathsf{Z} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$ with source i and target j:

$$\begin{aligned} P &*= (1 - 2u_i u_j) \\ Q &+= (K/2)u_i u_j \\ \phi &\wedge= (p_\ell = p_{\ell-1} \oplus (u_i \wedge u_j)). \end{aligned}$$

Note that all three forms are symmetrical in u_i and u_j, expressing the fact that with CZ it does not matter which line is considered the control. As with the gates before S, the conjugate forms change the same way so they are not shown.

– $\mathsf{CS} = \begin{bmatrix} \mathsf{I} & 0 \\ 0 & \mathsf{S} \end{bmatrix} = \begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&i \end{bmatrix}$ with source i and target j:

$$\begin{array}{ll} P \mathrel{*}= (1 - u_iu_j + h(i)u_iu_j) & P^* \mathrel{*}= (1 - u_iu_j + h(-i)u_iu_j) \\ Q \mathrel{+}= (K/4)u_iu_j & Q^* \mathrel{+}= (3K/4)u_iu_j \\ \phi \mathrel{\wedge}= (q_\ell = q_{\ell-1} \oplus (u_i \wedge u_j)) & \phi^* \mathrel{\wedge}= (q_\ell = q_{\ell-1} \oplus (u_i \wedge u_j)) \\ \quad \wedge(p_\ell = p_{\ell-1} \oplus (q_{\ell-1} \wedge u_i \wedge u_j)) & \quad \wedge(p_\ell = p_{\ell-1} \oplus (\bar{q}_{\ell-1} \wedge u_i \wedge u_j)). \end{array}$$

– $\mathsf{CT} = \begin{bmatrix} \mathsf{I} & 0 \\ 0 & \mathsf{T} \end{bmatrix} = \begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&\omega^{K/8} \end{bmatrix}$ with source i and target j:

$$\begin{array}{ll} P \mathrel{*}= (1 - u_iu_j + h(\omega^{K/8})u_iu_j) & P^* \mathrel{*}= (1 - u_iu_j + h(\omega^{7K/8})u_iu_j) \\ Q \mathrel{+}= (K/8)u_iu_j & Q^* \mathrel{+}= (7K/8)u_iu_j \\ \phi \mathrel{\wedge}= (r_\ell = r_{\ell-1} \oplus (u_i \wedge u_j)) & \phi^* \mathrel{\wedge}= (r_\ell = r_{\ell-1} \oplus (u_i \wedge u_j)) \\ \quad \wedge(q_\ell = q_{\ell-1} \oplus (r_{\ell-1} \wedge u_i \wedge u_j)) & \quad \wedge(q_\ell = q_{\ell-1} \oplus (\bar{r}_{\ell-1} \wedge u_i \wedge u_j)) \\ \quad \wedge(p_\ell = p_{\ell-1} \oplus (r_{\ell-1} \wedge q_{\ell-1} \wedge u_i \wedge u_j)) & \quad \wedge(p_\ell = p_{\ell-1} \oplus (\bar{r}_{\ell-1} \wedge \bar{q}_{\ell-1} \wedge u_i \wedge u_j)). \end{array}$$

The controlled $\mathsf{R_8}$ in the circuit in section 1 has $K/16$ in place of $K/8$ for P and Q and updates another phase variable s_ℓ for ϕ, so that the *pae* for p_ℓ has order 5. By using more variables to denote *carries* and sharing intermediate results we can code arbitrary (controlled) rotations $\mathsf{R_K}$ using $O(K)$ rather than quadratically many occurrences of variables in *pae*'s of orders 3 and 4. The next gates break the *pae* form when coded directly but can be written as compositions of gates or otherwise similarly broken down into small *pae*'s.

– $\mathsf{Y}^{1/2} = \frac{1+i}{2}\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} = \frac{\omega^{K/8}}{\sqrt{2}}\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$: Introduce one nondeterministic variable y_g, plus p_ℓ, q_ℓ, r_ℓ to update ϕ. We keep the scalar multiplication by $\omega^{K/8}$ but must remember to conjugate it:

$$\begin{array}{ll} P \mathrel{*}= h(\omega^{K/8})(2u_iy_g - 2u_i + 1) & P^* \mathrel{*}= h(\omega^{7K/8})(2u_iy_g - 2u_i + 1) \\ Q \mathrel{+}= (K/8) + (K/2)u_i(1 - y_g) & Q^* \mathrel{+}= (7K/8) + (K/2)u_i(1 - y_g) \\ \phi \mathrel{\wedge}= (r_\ell = \neg r_{\ell-1}) & \phi^* \mathrel{\wedge}= (r_\ell = \neg r_{\ell-1}) \\ \quad \wedge(q_\ell = q_{\ell-1} \oplus r_{\ell-1} \oplus (u_i \wedge \bar{y}_g)) & \quad \wedge(q_\ell = q_{\ell-1} \oplus \bar{r}_{\ell-1} \oplus (u_i \wedge \bar{y}_g)) \\ \quad \wedge(p_\ell = p_{\ell-1} \oplus \rho) & \quad \wedge(p_\ell = p_{\ell-1} \oplus \rho^*), \end{array}$$

where $\rho = (r_{\ell-1} = q_{\ell-1}) \wedge (r_{\ell-1} \oplus (u_i \wedge (\bar{y}_g)))$ and $\rho^* = (r_{\ell-1} = q_{\ell-1}) \wedge (\bar{r}_{\ell-1} \oplus (u_i \wedge (\bar{y}_g)))$. Finally, multiply R by $\sqrt{2}$, and note that y_g becomes the new u_i.

– $\mathsf{R_x}(\pi/2) = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & -i \\ -i & 1 \end{bmatrix} = \omega^{-K/8}\cdot\mathsf{V}$ on line i: Allocate a new nondeterministic variable y_g, and for ϕ, new phase variables p_ℓ and q_ℓ. Multiply R by $\sqrt{2}$ and do:

$$\begin{array}{ll} P \mathrel{*}= (1 + (h(-i) - 1)(u_i + y_g - 2u_iy_g)) & P^* \mathrel{*}= (1 + (h(i) - 1)(u_i + y_g - 2u_iy_g)) \\ Q \mathrel{+}= (3K/4)(u_i + y_g) - (K/2)u_iy_g & Q^* \mathrel{+}= (K/4)(u_i + y_g) - (K/2)u_iy_g \\ \phi \mathrel{\wedge}= (q_\ell = q_{\ell-1} \oplus u_i \oplus y_g) & \phi^* \mathrel{\wedge}= (q_\ell = q_{\ell-1} \oplus u_i \oplus y_g) \\ \quad \wedge(p_\ell = p_{\ell-1} \oplus (\bar{q}_{\ell-1} \wedge (u_i \oplus y_g))) & \quad \wedge(p_\ell = p_{\ell-1} \oplus (q_{\ell-1} \wedge (u_i \oplus y_g))). \end{array}$$

The equation $\mathsf{Y^{1/2}} = \mathsf{HZ} \cdot \omega^{K/8}$, where again the scalar multiplication can be ignored, gives an efficient alternative. The conjugate is $\mathsf{ZH} \cdot \omega^{7K/8}$. The gate $\mathsf{V} = \omega^{K/8}\mathsf{R_x}(\pi/2)$ satisfies $\mathsf{V^2} = \mathsf{X}$ and hence is also called $\mathsf{X^{1/2}}$ or $\sqrt{\mathsf{NOT}}$. The identity $\mathsf{V} = \mathsf{HSH}$ plus (optionally) multiplying by the scalar $\omega^{-K/8} = e^{7\pi i/4}$ thus

allows coding $\mathsf{R_x}(\pi/2)$ by *pae*'s, on pain of introducing one more nondeterministic variable. The conjugate matrix $\mathsf{R_x}(-\pi/2)$, which has positive-signed entries i on the off-diagonals, can be handled similarly. If these identities are used, then all *free* variables are assigned when placing Hadamard gates.

The sequence of equations, as gates are placed in left-to-right order (with matrices composed in right-to-left order), obeys the following invariant:

Lemma 1. *For any 0-1 assignment $c = (c_1, c_2, \ldots, c_h)$ to the free variables, input $a = a_1 \cdots a_m$ to the variables x_i, and initialization of the phase variables (to zero phase, say), the product of terms in P_C, sum of terms in Q_C, and conjunction of equational clauses in ϕ_C, can be evaluated in order with all right-hand side values defined in the initialization or in previous steps.* □

This enables an "intelligent backtrack" routine that, when incrementing $c \in \{0,1\}^h$ to the next c' in standard order, need only roll back to the first term or *pae* containing y_g, where c and c' agree in the first $g-1$ bits. Roughly speaking, this saves a factor of one-half the number h of free variables when carrying out the brute-force iteration through c to tabulate the results of each path. For Q_C the variables in Υ count as free, but if there are fewer than K uses of Υ_ℓ they can all be identified, so that the extra branching is only K and $R' = R\sqrt{K}$.

The invariant is particularly notable with our Boolean logic emulation. It means that for every gate, the equations typified by $p = q \oplus (u \wedge v)$ for every gate are interpreting the '=' as assignment, not just equality. Only the equations setting the output variables z_i to specified target values b_i (after the z_i have been filled by the equations $z_i = u_i$) have a constraining effect. We have found references using Boolean formulas to specify and SAT-solvers to test equivalence of portions of quantum circuits [27, 28] but not for conducting emulation. The significance of the invariant for a SAT solver or #SAT counter involves how resolvents of clauses (or rather their negated terms) are propagated. To give an intuitive example, consider a Boolean formula $\phi(x_1, \ldots, x_n, y_1, \ldots, y_n, z_1, \ldots, z_n$ expressing the multiplication relation $x \cdot y = z$ for the numbers x, y, z encoded by the variables in binary. If only z is given then we have the factoring problem in the form known to be tough for SAT solvers. In our case, however, the x will be given by the values of incoming line references u_i. The y may come from nondeterministic variables. Any setting of those variables will produce z by propagation. In any event, we are not in the situation where only z is given. Thus although the same Boolean relation ϕ is being used, the manner in which it is approached makes a big difference for solvers—as we demonstrate partially in section 8. Before that we show how our logic emulation binds this Boolean relation up with the Fourier transform.

6 Quantum Fourier Transform, Measurements, and Sampling

The *quantum Fourier transform* $\mathsf{QFT_n}$ on n qubits is represented (in the standard basis) by the ordinary $N \times N$ discrete Fourier matrix F_N where $N = 2^n$. It has entries

$$F_N[I, J] = \omega^{I \cdot J},$$

where $\omega = e^{-2\pi i/N}$. If the incoming phase is c then the new phase is $c' = c + I \cdot J$ modulo N. Hence the only equations we need to add are

$$W_\ell = W_{\ell-1} + I \cdot J \pmod{N}. \tag{5}$$

We can emulate $\mathsf{QFT_n}$ via the general recursion implied by the example in section 1. This uses $\Theta(n^2)$ gates—in particular, $\Theta(n^2)$ cases of $\mathsf{CR_{2^k}}$ for $k \geq n/2$, each of which uses $\Theta(n)$ variables once we apply the optimization by representing carries discussed in the last section regarding $\mathsf{R_8}$. This gives no better than $O(n^3)$ as a size guarantee. We can further observe in Figure 1 that partial results from carries can be shared among the formulas for the adjacent $\mathsf{CR_{2^k}}$ gates, which promises $O(n^2)$. However, (5) has Boolean circuit complexity $O(n \log n \log\log n)$ by the celebrated result of Schönhage and Strassen [29]. The asymptotically-small circuits include gates for each output bit of W_ℓ using the $3n$ input gates for $W_{\ell-1}$, I, and J. Each Boolean gate can be directly converted into one *pae* of order 3 or 4. Hence the size blowup should be only $\tilde{O}(n)$, i.e.,

quasi-linear. However, the advantage is asymptotic and is only known to take precedence when $n > 10,000$ or so, and the question of how best to emulate QFT_n for relevant ranges of the number of qubits seems still open (compare [30–33]).

Shor's algorithm [6] applies QFT in conjunction with *sampling*. To factor a given n-bit integer M, it starts by choosing $Q = 2^\ell$ where $\ell = 2n+1$, so that $M^2 < Q < 2M^2$, and a random $a < M$ which we may presume is relatively prime to M (else $\gcd(a, M)$ gives a factor at once). The quantum circuit C used by its inner loop starts with Hadamard gates on the first ℓ of 2ℓ qubit lines. It then places a deterministic circuit C_0 that maps any binary-encoded number $x < Q$ to $f_a(x) = a^x \pmod M$. More precisely, C_0 maps $x \cdot 0^\ell$ to the concatenation $x \cdot y$ where $y = f_a(x)$ as an ℓ-bit number. The combination with the Hadamard gates creates the *functional superposition*

$$\Phi = \frac{1}{\sqrt{Q}} \sum_x |x f_a(x)\rangle \,.$$

The quantum circuit then applies QFT_ℓ to the first ℓ lines and measures them to get an output $b < Q$. With substantial probability, b reveals a *period* r such that $f_a(x) = f_a(x + r)$ for all x and r in turn reveals a factor.

Thus far we have organized our software system to give the amplitude and probability of specific outcomes b but not to generate b from the set of possible outcomes. In order to emulate algorithms like Shor's, we want to sample according to the distribution

$$\mathcal{D}_{C,a}(b) = \Pr[C(a) = b] = |\langle a| C |b\rangle|^2.$$

Also let $\mathcal{D}_{C,a}(B)$, for any set $B \subset \{0,1\}^m$, be the sum of $|\langle a| C |b\rangle|^2$ over all $b \in B$. We will reduce quantum sampling to the *uniform generation* problem on the set S of 0-1 roots of polynomials P, Q or of satisfying assignments to a formula ϕ. First we review the classical reduction from counting to uniform generation. It starts with one call to $\#sat$ to compute the cardinality of S.

- Using one call to $\#sat$, compute $|S_0| = |\{ x \in S : x_1 = 0 \}|$, so $|\{ x \in S : x_1 = 1 \}| = |S| - |S_0|$.
- Set $x_1 = 0$ with probability $|S_0|/|S|$ and $x_1 = 1$ otherwise.
- Substitute the value of x_1 into ϕ and recurse on x_2 and so on.

The k-th iteration deals with sets of outcomes sharing a common prefix $x_1 x_2 \ldots x_k$. Given any ordered subset $I = \{ i_1, i_2, \ldots, i_k \}$ of the line indices $1, \ldots, m$ and string $\beta \in \{0,1\}^{|I|}$, define the *cylinder* $B_{I,\beta}$ to be the set of strings $b \in \{0,1\}^m$ such that $b_{i_1} b_{i_2} \cdots b_{i_k} = \beta$. If I is omitted then it is the subset $1, \ldots, |w|$.

A naive attempt to emulate the above process beginning with B_0 would substitute $z_1 = 0$ but leave the variables $z_2, \ldots, z_m$ open in the first formula ϕ_0. Applying the counting in (1)–(3) to the phase-shifted formulas derived from ϕ_0 would fail, however, because it would attempt to cancel counts of solutions with different final locations $b_2, \ldots, b_m$ whose waves do not interfere. The fix is to maintain the probabilities directly rather than the amplitudes in a way that preserves the cylindrical structure. We write $[Z_I = \beta]$ for the act of substituting the variables z_i for $i \in I$ by the respective bits in β.

Theorem 2. *For any C, a, m, K as in Theorem 1 we can find constants R, R' and build a polynomial $\mathcal{P}_C$ over any adequate ring, a polynomial $\mathcal{Q}_C$ over $\mathbf{Z}_K$, and a Boolean formula Φ with the same variables as ϕ_C plus an extra variable w such that for any cylinder $B = (I, \beta)$:*

$$\mathcal{D}_{C,a}(B) = \frac{1}{R^2} \left[\#binsols(\mathcal{P}_C[X = a, Z_I = \beta] - 1) - \#binsols(\mathcal{P}_C[X = a, Z_I = \beta] + 1) \right] \tag{6}$$

$$= \frac{1}{R'^2} \left[\#binsols(\mathcal{Q}_C[X = a, Z_I = \beta]) - \#binsols(\mathcal{Q}_C[X = a, Z_I = \beta] - K/2) \right] \tag{7}$$

$$= \frac{1}{R^2} \left[\#sat(\Phi[X = a, Z_I = \beta, w = 0] - \#sat(\Phi[X = a, Z_I = \beta, w = 1] \right]. \tag{8}$$

The same asymptotic size limits as for P_C, Q_C, ϕ_C in Theorem 1 apply to $\mathcal{P}, \mathcal{Q}, \oplus$.

The most striking difference from Theorem 1 is that there is no summation over phases J. Regardless of K, the probability is a difference of two #P functions.

Proof. We use the conjugates P_C^*, Q_C^*, ϕ_C^* with the proviso that they share the input and output variables X, Z but have their own copies V', W', Y' (and Υ') of line, phase, and nondeterministic variables. We also make ϕ_C' to be a similar non-conjugated copy of ϕ_C. We prove (7) first. For any b, abbreviating $Q_C[X = a, Z = b]$ to $Q_{a,b}$, we have

$$\Pr[C(a) = b] = \langle a| C |b\rangle \langle a| C |b\rangle^* = \frac{1}{R'^2} \sum_{J,L=0}^{K-1} \omega^{L+J} \#binsols(Q_{a,b} - J) \#binsols(Q_{a,b}^* - L).$$

Now the left-had side is a real number. Hence all terms involving ω^{L+J} where $L + J$ is not a multiple of $K/2$ *cancel*—as can also be worked out from the conjugacy of values of Q_C vis-à-vis Q_C^*. This leaves only a positive term with $L = K - J$ and a negative term with $L = \frac{K}{2} - J$, both modulo K. So $R'^2 \Pr[C(a) = b]$

$$\begin{aligned}
&= \sum_{J=0}^{K-1} \#binsols(Q_{a,b} - J) \#binsols(Q_{a,b}^* + J) - \sum_{J=0}^{K-1} \#binsols(Q_{a,b} - J) \#binsols(Q_{a,b}^* + J - \frac{K}{2}) \\
&= \#binsols(Q_{a,b} + Q_{a,b}^*) - \#binsols(Q_{a,b} + Q_{a,b}^* - \frac{K}{2}).
\end{aligned}$$

So put $\mathcal{Q}_C = Q_C + Q_C^*$. Since the right-hand side has no phase dependence anymore,

$$\begin{aligned}
R'^2 \Pr[C(a) \in B] &= \sum_{b \in B} R'^2 \Pr[C(a) = b] \\
&= \sum_{b \in B} \#binsols(\mathcal{Q}_C[X = a, Z = b] - \sum_{b \in B} \#binsols(\mathcal{Q}_C[X = a, Z = b] - \frac{K}{2}) \\
&= \#binsols(\mathcal{Q}_C[X = a, Z_I = \beta]) - \#binsols(\mathcal{Q}_C[X = a, Z_I = \beta] - \frac{K}{2}).
\end{aligned}$$

In like manner we obtain $\mathcal{P}_C = P_C \cdot P_C^*$ (with shared X, Z), and using $h(1) = 1$, $h(-1) = -1$ per remarks before Theorem 1, we obtain

$$R^2 \Pr[C(a) \in B] = \#binsols(\mathcal{P}[X = a, Z_I = \beta] - 1) - \#binsols(\mathcal{P}[X = a, Z_I = \beta] + 1).$$

It is notable that no absolute value bars are needed here. For Φ_C there is a final twist. Let W_s, W_s^*, W_s' be the final suites of phase variables in $\phi_C, \phi_C^*, \phi_C'$ and similarly abbreviate $\phi_C[X = a, Z = b]$ to $\phi_{a,b}$. We get that $R^2 \Pr[C(a) = b]$

$$\begin{aligned}
&= \sum_{J,L=0}^{K-1} \omega^{J+L} \#sat(\phi_{a,b}[W_s = J]) \#sat(\phi_{a,b}^*[W_s^* = L]) \\
&= \sum_{J=0}^{K-1} \#sat(\phi_{a,b}[W_s = J]) \#sat(\phi_{a,b}^*[W_s^* = K - J]) - \#sat(\phi_{a,b}[W_s = J]) \#sat(\phi_{a,b}^*[W_s^* = \frac{K}{2} - J]) \\
&= \sum_{J=0}^{K-1} \#sat(\phi_{a,b}[W_s = J]) \#sat(\phi_{a,b}'[W_s' = J]) - \#sat(\phi_{a,b}[W_s = J]) \#sat(\phi_{a,b}'[W_s' = \frac{K}{2} + J])
\end{aligned}$$

Now unpacking $W_s = w_0, \ldots, w_{k-1}$ and similarly for W_s', define

$$\Phi_C = \phi_C \wedge \phi_C' \wedge (w_{k-1} = w_{k-1}') \wedge \cdots \wedge (w_1 = w_1') \wedge (w_0 \oplus w_0' = w).$$

Then $\Phi_C[w=0]$ equates $W'_s = W_s$ and so by the disjointness of Y, V, W from Y', V', W',

$$\#sat(\Phi_C[X=a, Z=b, w=0] = \sum_{J=0}^{K-1} \#sat(\phi_{a,b}[W_s = J])\#sat(\phi'_{a,b}[W'_s = J]).$$

Because adding $K/2$ is the same as flipping the top phase,

$$\sum_{J=0}^{K-1} \#sat(\phi_{a,b}[W_s = J])\#sat(\phi'_{a,b}[W'_s = J + \frac{K}{2}]) = \#sat(\Phi_C[X=a, Z=b, w=1]).$$

The rest involving cylinders B is similar to before. □

Note that using ϕ'_C rather than ϕ^*_C enabled us to avoid representing $K - J$ or $\frac{K}{2} - J$, both of which involve not only negating variables but also the extra cost of adding 1.

Theorem 3. *The distribution $\mathcal{D}_C$ projected onto any cylinder $B = (I, \beta)$ can be computed with $m - |I|$ evaluations of $\#sat(\Phi_C[X = a, Z_{I'} = \beta', w = 0]) - \#sat(\Phi_C[X = a, Z_{I'} = \beta', w = 1])$ for successive extensions I' of I and β' of β. Assuming constant arity, this needs only $O(m + sk)$ time* **and space** *apart from the #sat invocations.*

Proof. Theorem 2 enables using the classical polynomial-time oracle procedure given before it. □

Thus sampling can be emulated via #SAT solvers but at the cost of linearly many invocations (rather than one or two), and most important, of doubling the number $h = |Y|$ of nondeterministic variables. The tradeoff is that the double rail entirely saves the space previously used to build a dictionary of counts for each location. With regard to equation (5) and the idea of emulating Shor's algorithm, it can be objected that the logic uses the multiplication relation $K = I \cdot J$ for numbers I, J, K of the same order as the number M we are trying to factor—indeed, on the order of M^2. We have attempted to rebut this already with the remarks at the end of section 5 about the flow of processing.

What is the meaning of, say, $\phi_C[z_1 = b]$ where $b \in \{0, 1\}$ without the "double-rail" use of ϕ'_C? Theorem 2 also justifies the interpretation that $\phi_C[u_1 = b]$ (at any time, not just at the end when u_1 is equated to z_1) represents *the state of the system after measuring the first qubit and getting the outcome b.* After the measurement, the constant R needs to be re-normalized by multiplying it by the square root of the probability of getting the outcome b as computed via Theorem 2. This enables continuing to represent the system as quantum operations are added to C after the measurement is executed. The use of $\phi_C[u_1 = b]$ is also consonant with the *principle of deferred measurement*, which states that if b on line i is used only in a test "if b then do G else do nothing" on other qubits, then the results are the same as removing the measurement, replacing G by the controlled gate CG with source on i, and including i and result b_i in any final measurement.

We finish our systems toolkit with two more observations. First, define $C*$ to be the mirror image of C with the former outputs $b_1, \ldots b_m$ now being designated as inputs and vice-versa for $a_1, \ldots, a_m$, and with each gate G reversed by substituting its adjoint G^*. Note that this is not the same as conjugating each gate but keeping the sequence from a to b the same, which is what is modeled by P^*_C, Q^*_C, and ϕ^*_C.

Proposition 1. *For every quantum circuit C, $P_{C^*} = P_C$, $Q_{C^*} = Q_C$, and $\phi_{C^*} = \phi_C$ (up to re-labeling of variables).*

Proof. Since the adjoint of a "bra" is a "ket," $\langle a| C^* |b\rangle = \langle b| C |a\rangle$, so we may picture the original C running right-to-left or with a and b interchanged. Since the general-case construction in the proof of Theorem 1 is symmetrical for each gate until the $z_i = u_i$ step, and the only substitution is to equate two variables, the resulting P_{C^*} is the same polynomial as P_C, up to interchanging the substituted variables and a with b. The same goes for Q_C and ϕ_C. □

The *tensor product* $C_1 \otimes C_2$ of two quantum circuits C_1 and C_2 simply consists of laying them side-by-side, with no gates between their respective qubit lines.

Proposition 2. *For any quantum circuits C_1 and C_2 of min-phase K, $P_{C_1 \otimes C_2}$ can be taken as $P_{C_1} \cdot P_{C_2}$, $Q_{C_1 \otimes C_2}$ can be taken as $Q_{C_1} + Q_{C_2}$ (mod K), and $\phi_{C_1 \otimes C_2}$ can be taken as $\phi_{C_1} \wedge \phi_{C_2}$.*

Proof. We can regard $C_1 \otimes C_2$ as a single quantum circuits in which the gates are introduced in any left-to-right order. The sequencing does not affect which variables are "u_i" on any qubit line when each gate is placed. □

The *compute-uncompute* structure mentioned in section 1 combines several of these elements. We have our initial circuit C with some set I of r qubit lines intended for output. Then r CNOT gates are placed with controls in I and targets in r fresh ancilla lines initialized to 0. Then C^* is placed on the initial m lines so that $C(x)$ is "uncomputed" and measurements of the $m + r$ lines will always give xy for some output y. It is noteworthy that the composition of C^* after C gives almost the same product, sum, and AND forms, respectively, as Theorem 2 for the computation of probabilities. Further processing on y alone is in parallel with C^* and is modeled by multiplying, adding, or AND-ing in more terms. The only distinction from the completely-parallel situation of Proposition 2 comes from the forced variables v_j and terms connecting v_j to u_i and u_j (where $j = m + i$ with $i \in I$) when the CNOT gates are placed.

7 Examples and Prospective Applications

First, we take stock. We have defined a three-pronged framework that not only handles some universal set of gates and decision problems in BQP, but directly implements a wide range of quantum operations: exact quantum Fourier transform, sampling, on-the-fly measurement, and some circuit combinations. Our formal objects *remain small* and yet maintain full information about the quantum system. They are conducive to further manipulation by computer algebra and formal logic packages. They not only postpone the ostensible exponential blowup but raise the prospect that heuristic solvers—honed with high effort apart from the quantum context—can avoid it altogether. We give some examples that hint at the manner and plausibility of advances obtained this way.

One problem to attack is, *which restricted forms of quantum circuits can be emulated in classical polynomial time?* A fundamental case is the Gottesman-Knill theorem (see [34, 35]):

Theorem 4. *Quantum circuits that use only Hadamard, the Pauli gates, the phase gate* S, CNOT, CZ, *and swaps can be simulated in classical deterministic polynomial time. In particular, languages defined in the manner of section 1 by uniform families of such circuits belong to* P.

Many proofs of this theorem have been given, including one in the related polynomial framework of [16]. Ours uses the even newer high-level result in [36] (also in section 12 of [37]) that computing functions of the form (2) for polynomials of degree 2 belongs to P. It shows an even finer line than one of degree between polynomial-time computability and hardness.

Proof. Build the additive representation Q_C of a circuit C using these gates with $K = 4$. Note that $\mathsf{Z} = \mathsf{S}^2$ and $\mathsf{HZH} = X$, from which it follows that CNOT can be made from H and CZ, and finally $\mathsf{Y} = i\mathsf{XZ}$. So we need only examine the terms as H, S, and CZ are added—none involves a constraint so there is no Υ:

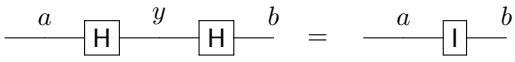
- H: $2u_i y_g$.
- S: u_i, but since u_i is 0-1 valued we can add u_i^2 instead.
- CZ: $2u_i u_j$.

Thus Q_C is a sum of monomials of the form $2xy$ or x^2. Modulo 4, both of these terms are invariant under replacing x by $x+2$. Hence solution counts in $\{0,1\}^t$ where t is the total number of variables and the count in $\mathbf{Z}_4^t$ are related by a fixed factor of 2^t. We can thus use the theorem of [36] directly to count solutions in $\{0,1\}^t$ to $Q_C - J$, $J = 0, 1, 2, 3$. □

Now, however, notice that CS also produces an additive quadratic term, namely $u_i u_j$. Thus any circuit C composed of H and CS (which simulate S and CZ) also has Q_C of degree 2. If the theorem of [36, 37] applied to counting *binary* solutions then BQP = P would follow. The nub is that the term xy is not invariant under $x \mapsto x + 2$. Counting *0-1* solutions for quadratic polynomials with such terms is shown #P-complete in general by [38]. The difference between u_i and $u_i u_j$ reflects the linear-versus-quadratic difference in [16], but here we have isolated the jump in complexity to the latter's coefficient being 1 not 2. This happens entirely within a setting where counting the number of solutions of quadratic polynomials modulo $K \geq 4$ is easy—but counting the number of *binary* solutions is hard.

Since our proof of Theorem 4 can yield amplitude information at any stage of the circuit we speak of simulation not just emulation here. The general algorithm in [36, 37] does not run in $\tilde{O}(t)$ time. It remains to be seen whether its specialization to sums of $2xy$ or x^2 can be honed to rival the nearly linear running time of Jozsa [39]. The Toffoli, CS, and T gates all introduce terms of degree 3 into Q_C. It is interesting to ask whether circuits obtained by adding one of these gates have Q_C with a simply-expressed structure that resists the "dichotomy" phenomenon (problems being either in P or #P-complete, nothing in-between) in these papers.

Note that we avoided the constraint $\Upsilon \cdot (\cdots)$ involved with CNOT. That or propagating the annotation $2u_i u_j - u_i - u_j$ along line j would lead to terms of degree 3 or higher, while still giving polynomials that are functionally equivalent to the ones of degree 2 above. This leads to the broad question of recognizing such equivalence, and whether algebraic and Boolean solvers may even go beyond it by finding simplifications that do not have direct analogues at the level of quantum gates. The simplest identity involving a nondeterministic gate, namely HH = I, already shows some challenges. Substituting $x_1 = a$ and $z_1 = b$ gives the diagram:



The multiplicative polynomial of the left-hand circuit is

$$P_C = (1 - 2ay)(1 - 2yb) \mapsto 1 - 2ay - 2yb + 4ayb,$$

with a background factor of $R = 1/2$ from the two Hadamard gates. Why is this equivalent to the identity-gate polynomial? The latter is

$$P_I = E(a, b) = 1 - a - b + 2ab.$$

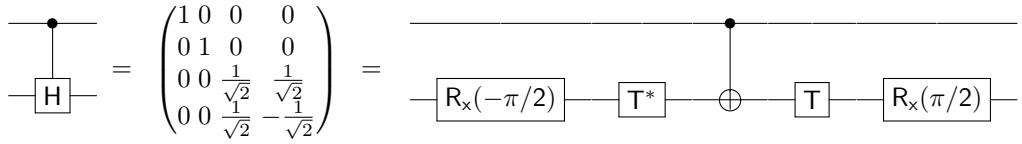
A clue is to look at what happens to P_C under the "illegal" substitution $b = 1 - a$, when it becomes

$$1 - 2ay - 2y + 2ay + 4ay - 4a^2 y \mapsto 1 - 2y.$$

A multiplicative term $(1 - 2y)$ where this is the only occurrence of the interior variable y behaves much like "Υ" in the additive representations. It sets up a 1-1 correspondence between solutions for each $e(\omega)$ and $e(-\omega)$, which cancels everything to zero. Thus all assignments into P_C that make $a \neq b$ contribute a net of zero to the complex amplitude. Substituting $a = b$ makes both polynomials reduce to 1. Interestingly, substituting $y = 1/2$ into P_C yields P_I, but this is not a "legal" substitution.

For the additive representation over $\mathbf{Z}_4$ we get $Q_C = 2ay + 2yb = 2y(a + b)$. Again the legal paths with $b \neq a$ cancel, while those with $b = a$ make $2y(a + b)$ a multiple of 4, which yields 0 like the identity does. But how can we recognize this? The Boolean formula ϕ_C does $p_1 = p_0 \oplus (a \wedge y)$ and $p_2 = p_1 \oplus (y \wedge b)$, which combine to give $p_2 = p_0 \oplus (a \wedge y) \oplus (y \wedge b)$. For $y = 0$ this gives $p_2 = p_0 = 0$, whereas for $y = 1$ this makes $p_2 = p_0 \oplus (a \oplus b)$. Again we can infer that $a \neq b$ gives canceling phases while $a = b$ makes $p_2 = 0$ in this case too, so the net effect is the identity. But still, this already seems challenging to automate, let alone recognizing larger-scale circuit identities discussed previously. The nub is how well #SAT solvers can rearrange clauses so that variables like y get "leveled off" in all cases, in the manner of eliminating rows in the *Tetris* video game.

Now we consider a much harder example of equivalence. A controlled gate CG deviates from the cases in section 5 when G is nondeterministic. For example, the controlled Hadamard gate CH has the circuit notation and matrix shown at left below.



The matrix is not balanced. Letting i be the control line and j the target, we can introduce a new variable y_g opposite u_j. The phase term $(1 - 2u_j y_g)$ for a free-standing Hadamard on line j becomes $(1 - 2u_i u_j y_g)$ but the control gives a constraint: if $\neg u_i$ then $y_g = u_j$. So we could multiply P by $(1 - 2u_i u_j y_g)(u_i + (1 - u_i)(1 + 2u_j y_g - u_j - y_g))$ and do analogously for Q by

$$Q \ += (K/2)u_i u_j y_g + \Upsilon_\ell (1 - u_i)(u_j + y_g - 2u_i y_g).$$

The problem from the lack of balance, however, is a miscount in case $u_i = 0$. Then y_g is forced to equal u_j so, for instance, there is only one path from the input 00 to the output 00. We could patch this by introducing a new variable v, multiplying P also by $(1 - u_i v)$, adding $u_i v \Upsilon_\ell$ to Q, and doing the following for ϕ:

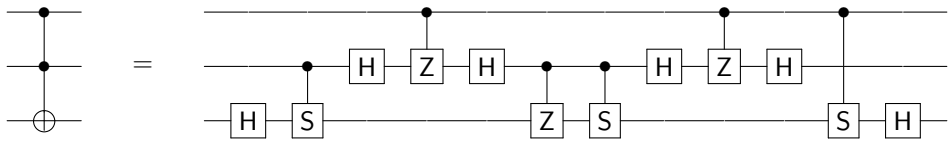
$$\phi \ \wedge= (p_\ell = p_{\ell-1} \oplus (u_i \wedge u_j \wedge y_g)) \wedge (y_g = u_j \oplus (u_i \wedge v))$$

The right-hand *pae* enforces the constraint when $u_i = 0$ and rectifies the count of satisfying assignments. However, v needs to be treated as a new "virtual location" so that the paths for $v = 0$ and $v = 1$ do not amplify. This re-raises issues of sampling and *post-selection* that can be explored further. Alternately, one can use the equivalence of CH to a composition of five balanced gates shown at right. Let y_1 be the variable introduced by the $\mathsf{R_x}(-\pi/2)$ and y_2 the one introduced by its conjugate. For $a = 00$, what happens is that the forced variable v_f introduced by the CNOT gate is equated to y_1. The final quarter and half phases computed in four stages (besides the CNOT which changes no phase) are:

$$\begin{array}{lllll} q_4 = q_3 \oplus v_f \oplus y_2 & = & y_1 \oplus y_1 \oplus y_2 & = & y_2 \\ p_4 = p_3 \oplus ((v_f \oplus y_2) \wedge \bar{q}_3) & = & (y_1 \oplus y_2) \wedge \bar{y}_1 & = & \bar{y}_1 \wedge y_2. \end{array}$$

The outcome $b = 00$ sets $y_2 = 0$ and hence gives both paths for y_1 a final phase of zero in a way that leaves the variable y_1 completely free in both equations. Thus y_1 acts like "v" above. The potential outcome $b = 01$ sets $y_2 = 1$, which fixes the quarter phase as 1 and makes the variable y_1 switch between i and $-i$ in the top phase, which duly cancel to make $b = 01$ impossible. This leads to an operational verification of equivalence when the other values of a are verified as well, but this is far from a simplification.

The challenge is to recognize such simplifications. The identity $\mathsf{HSH} = \mathsf{V}$ and its analogue using CS and CV may have intermediate difficulty. So may the equivalence of the Toffoli gate to two-qubit gates [20]:



In our rendition there are six Hadamard gates to "level away." We have only opened the door to these questions about equivalence and efficacy by a crack. We conclude with only a very preliminary indication that there may be light through the crack from the performance and scaling of existing solvers.

8 A Few Experimental Results

We show the experimental results from the performance of solving the generated Boolean formulas using our brute-force (BF) method and current versions of the *Cachet* [12–14] and *sharpSAT* [11] solvers. All the circuits generated and tested begin with a bank of Hadamard-gates on every line applied to the input vector, and subsequently differ as follows:

– CNOT staircase: apply CNOT gates with control n and target $n+1$ followed by a T-gate on line n for $n = 1, \cdots, m-1$. Concretely, a sequence of applied gates would be $\mathsf{CNOT}(1,2), \mathsf{T}(2), \mathsf{CNOT}(2,3), \mathsf{T}(3), \ldots, \mathsf{T}(m), \mathsf{CNOT}(m,1)$.
– CNOT staircase with appended CZ and CV gates that alternate: After the bank of H gates and the above CNOT staircase, apply CV and CZ alternately to lines $n, n+1$ and $n+2$. Concretely, a sequence of applied gates would be:

$$\mathsf{CNOT}(1,2), \mathsf{T}(2), \mathsf{CNOT}(2,3), \mathsf{T}(3), \ldots, \mathsf{T}(m), \mathsf{CNOT}(m,1)$$
$$\mathsf{CV}(1,2), \mathsf{CZ}(2,3), \mathsf{CV}(3,4), \mathsf{CZ}(4,5), \cdots, \mathsf{CZ}(n-2,n-1), \mathsf{CV}(n-1,n).$$

– Initial segments of circuits proposed in [21] to be hard for classical simulations.

The following results were run from C++ code on one core thread of a Dell PowerEdge R720 server with 3.3Ghz E5-2643 CPU. The experimental specification called for reading a description of the quantum circuit as described above, transcribing it into the standard ("DIMACS") format used by SAT solvers, and running ten trials each of our brute-force routine, *sharpSAT*, and *Cachet*. The latter two were compiled with optimization on the same machine as our solver. Our code uses pointers to unsigned integers to reference Boolean values; there does not seem to be any tangible gain from packing integers to represent multiple Boolean values or using a `bitset` implementation.

Table 1. CNOT staircase (microsecond (us))

m	BF	sharpSAT	Cachet
4	35	7,409	35,870
8	618	8,021	36,245
12	12,239	8,292	35,245
16	122,063	8,100	34,120
20	2,101,034	9,594	39,994
24	62,935,720	9,024	42,994

Table 1 shows the results of solving the generated Boolean formulas for circuits consisting of a "staircase" of m Hadamard and CNOT gates producing entanglements. While the brute-force (BF) running time grows exponentially in m as expected, the running times of *sharpSAT* and *Cachet* change little. This suggests that the solvers are able to figuratively flatten the staircase so that the transformed solutions are easy to count. The next experiment tries to frustrate this by sprinkling controlled gates of non-binary phases amid the lines. The CV gates add extra nondeterminism in the standard basis.

Table 2. CNOT staircase with CZ and CV (microsecond (us))

m	BF	sharpSAT	Cachet
4	410	8,373	42,744
8	91,534	8,486	49,493
12	13.9 s	10,400	64,300
16	14,588 s	18,500	52,000
20	$\gg$ 5 hours	145,700	106,000
24	$\gg$ 5 hours	1,196,100	362,700

In Table 2, the relations between them are similar to that in Table 1 until the line for $m = 16$. The BF running time balloons up even more owing to the extra nondeterministic variables. The *sharpSAT* solver seems to have special difficulty with the circuits of 20 and 24 qubits.

We also tried initial sets of layers from the circuits treated in [22] based on indications from [21] of their being hard to simulate classically. Those circuits had too much nondeterminism for BF but gave results within a few hours for *sharpSAT* and *Cachet* until the circuits reached 6 or 7 layers of 24 to 36 qubits—well short of the 40-layer simulations on massively parallel hardware announced by [22].

The results show that *sharpSAT* and *Cachet* give better scalability on these circuits. They as yet do not, however, even "recognize" the identity $\mathsf{HH} = I$ in the sense of having similarly close running times when extra HH pairs are added to these circuits. Of course, our BF method has its time compounded by a factor of 4 for each pair since it blindly tries all combinations. This points to the goal is tuning the solvers for a repertoire of basic quantum simplifications, in the hope that this will boost the heuristics already employed.

The final preliminary experiment, just at press time, emulated the circuits for Shor's algorithm that are constructed by `libquantum` [2–4]. The `libquantum` package and its `shor` routine are distinguished among quantum simulation software by being part of the SPEC CPU2006 benchmark suite [40]. We modified the v1.1.1 release code so that it prints out each quantum gate in the readable format of our emulator. The circuits are generated specially for each M and choice of random seed a. For $M = 2021$ and $a = 7$ the circuit built by the `shor` routine uses 22 principal qubits, 35 ancillas, and has 98,135 elementary gates. By far the largest block is for the modular exponentiation step which consists entirely of deterministic gates (only NOT, CNOT, and Toffoli). They are somewhat larger than the original circuits for Shor's algorithm detailed in [41]. They are far from optimal; indeed, Markov and Saeedi [31, 42] showed 6-to-8-fold improvements by high-level means and other gate-level improvements have been made [43, 32, 44].

The SPEC CPU2006 benchmark consists of one run of `shor` on M and a, which does just one iteration of the quantum circuit—no restarts in case it doesn't succeed. It uses a numerical gate-by-gate simulation. For M approaching 10,000 our compile of `shor` overflows its hash table of over 500MB. It functioned correctly on $M = 2021 = 43 * 47$, which is just under 2^{11}. Our emulator's brute-force routine reaches its limit for numbers larger than $2^9 = 512$, which entails running through $2^{36} = 64$ billion assignments for each of 9 sampled bits. Optimizing the initial modular exponentiation stage would make very little difference in our brute-force routine because only one in every 2^{18} assignments backtracks beyond the final QFT step which has 18 Hadamard gates of its own. Runs with the #SAT solvers succeeded for $M = 15$ and $M = 21$ but bogged down for $M = 55$, with *sharpSAT* expanding to over 34GB of system resources. This evidently owes to the second copy of ϕ in the proof of Theorem 2 doubling the count of Hadamard gates again. The brute-force compilation needed only the single copy and stayed within 71MB, under 0.1% of system memory, per billion assignments tried.

9 Conclusions and Research Directions

We have defined a natural *emulator* in the sense of [8]. It has no explicit representation of matrices or quantum states or any physical elements and loses no precision while needing only whole integer arithmetic. Preliminary experimental work shows that it is competent even in brute-force simulation and enables distinctly high performance through #SAT solvers in several instances. It has a high memory footprint only in the accumulation of final results. The sampling procedure of section 6 essentially eliminates that footprint but at double the cost in nondeterminism and a squaring of brute-force simulation time. Overall the architecture is markedly different from that of commonly employed systems. It is so fine-grained as to escape limitations of more-structured systems including [45–49], but eschews explicit benefit from such structures. We showed its capability to work with quantum circuits of thousands of gates, though also where it is lagging by a factor of about 20 in the magnitude of numbers M that it can handle compared to `libquantum` and that the #SAT solvers lagged by another factor of 10.

Higher performance may come from software advances in #SAT solvers. These might be tailored to leverage the special "parity-of-AND" form of the equational clauses before conversion to conjunctive normal form. At the very least, our work has supplied a new class of natural instances by which to challenge these solvers. One natural metric will come from the work in progress of emulating Shor's algorithm. We have shown all the ingredients except for concretely optimizing the deterministic layer for modular exponentiation—which may make a greater difference for #SAT solvers than it does for our brute-force routine. Research progress

will need to center on the details of the solvers and their heuristics, to tune them more to recognizing symmetries and identities arising in quantum circuits. Most in particular, it would be interested to test the direction-of-flow arguments made at the end of section 5 on a set of natural instances.

Another question is whether solvers can be tailored to the algebraic forms, the low-degree Q_C polynomials in particular. Comparison of the efficacy of our algebraic and logical representations is hampered by the relative lack of dedicated symbolic polynomial equation solvers compared to SAT solvers. On the theoretical side, our own proof of Theorem 4 shows a new level of fineness in the demarcation between cases of counting problems that are polynomial-time solvable and those that are #P-complete.

A further research direction suggested by the obstacles to exact emulation of Shor's algorithm is to use approximation. In the context of simulators typified by `libquantum` and much research on perturbations of Shor's algorithm, *approximation* has its standard numerical meaning, as exemplified by [50–53]. In logic, we can approximate a complex predicate by a simple one that gives the correct answer on all but a sparse set of instances. This suggests looking for a sparse approximation to the modular exponentiation relation.

We close with an analogy to elaborate the main issue with our architecture. Solvers that represent whole state vectors in some form and emulate circuit levels sequentially figuratively have the memory footprint of a giant. Once the giant gets going, however, it walks with a steady gait. Our model instead employs an army of fleet-footed mice and can send one 'mouse' (i.e., evaluate one Feynman path) at a time with zero footprint—except for housing the results of the mice at the end. The issue is that each intermediate nondeterministic gate doubles the size of the mouse army. Of course the brute-force simulation doesn't reduce it but all heuristics must grapple with the initial condition that multiple mice are still "there" before any cancellations or Tetris-style blocking are applied. Aaronson and Chen [54] have recently shown a simple tradeoff between the "path" and "whole-state" extremes, but it still pays some exponential factor (they state what in our notation would be 2^n versus 2^s, but in our setup it is more like 2^m versus 2^h) up front.

The formulas manipulated by SAT and #SAT solvers, insofar as they expand via resolution and other techniques, are between the mice and the giant. The further success of this approach may depend on how much implicit combination they can achieve. For some sampling steps of quantum algorithms, certain tradeoffs between accuracy of the counting and size can be tolerated. How this can possibly interact with the deep tradeoff of approximation and hardness in sampling, over which the argument over "quantum supremacy" is currently centered, remains to be seen.

Acknowledgments Most of the initial work on this paper was done while the first author was a sabbatical visitor to the Universitié de Montreal, partly supported by the UdeM Département d'informatique et de recherche opérationnelle, and by the University at Buffalo Computer Science Department. We thank especially Professors Pierre McKenzie, Alain Tapp, and Jin-Yi Cai for insightful discussions, and Igor Markov for further pointers to the literature and a press-time tip that `libquantum` could be modified to output the entire quantum circuits of thousands of gates for Shor's algorithm in a format readable by our emulator. We thank the referees and also Michael Nielsen, John Sidles, Wim van Dam, Alex Russell, and Ronald de Wolf for helpful comments.

References

1. Dawson, C., Haselgrove, H., Hines, A., Mortimer, D., Nielsen, M., Osborne, T.: Quantum computing and polynomial equations over the finite field Z_2. Quantum Information and Computation **5** (2004) 102–112
2. Butscher, B., Weimer, H.: Simulation eines Quantencomputers. http://www.libquantum.de/files/libquantum.pdf (2003)
3. Weimer, H., Müller, M., Lesanovsky, I., Zoller, P., Büchler, H.: A Rydberg quantum simulator. Nature Physics **6** (2010) 382–388
4. Weimer, H., Butscher, B.: libquantum 1.1.1: the C library for quantum computing and quantum simulation. http://www.libquantum.de/ (2003–2013 (v. 1.1.1))
5. Wybiral, D., Hwang, J.: Quantum circuit simulator. http://www.davyw.com/quantum/ (2012)
6. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: Proceedings of the 35th Annual IEEE Symposium on the Foundations of Computer Science. (1994) 124–134

7. Boneh, D.: Twenty years of attacks on the RSA cryptosystem. Notices of the American Mathematical Society **46** (1999) 203–213
8. Häner, T., Steiger, D., Smelyanskiy, M., Troyer, M.: High performance emulation of quantum circuits. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Salt Lake City, Utah, Nov. 2016, IEEE press (2016) Article 74 in e-volume.
9. Greuel, G.M., Pfister, G., Schönemann, H.: Singular version 1.2 User Manual . In: Reports On Computer Algebra. Volume 21. Centre for Computer Algebra, University of Kaiserslautern (1998) `http://www.singular.uni-kl.de/`
10. Greuel, G.M., Pfister, G., Schönemann, H.: Singular 3.0. A Computer Algebra System for Polynomial Computations, Centre for Computer Algebra, University of Kaiserslautern (2005) `http://www.singular.uni-kl.de`.
11. Thurley, M.: sharpsat – counting models with advanced component caching and implicit bcp. In: Theory and Applications of Satisfiability Testing - SAT 2006: 9th International Conference, Seattle, WA, USA, August 12-15, 2006. Proceedings. Volume 4121 of Lect. Notes in Comp. Sci., Springer Verlag (2006) 424–429
12. Sang, T., Bacchus, F., Beame, P., Kautz, H., Pitassi, T.: Combining component caching and clause learning for effective model counting. In: Seventh International Conference on Theory and Applications of Satisfiability Testing, Vancouver. (2004)
13. Sang, T., Beame, P., Kautz, H.: Heuristics for fast exact model counting. In: Eighth International Conference on Theory and Applications of Satisfiability Testing, Edinburgh, Scotland. (2005)
14. Sang, T., Beame, P., Kautz, H.: Performing Bayesian inference by weighted model counting. In: Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05), Pittsburgh, PA. (2005)
15. Gerdt, V., Severyanov, V.: A software package to construct polynomial sets over Z_2 for determining the output of quantum computations. Nuclear Instruments and Methods in Physics Research A **59** (2006) 260–264
16. Bacon, D., van Dam, W., Russell, A.: Analyzing algebraic quantum circuits using exponential sums. http://www.cs.ucsb.edu/ vandam/LeastAction.pdf (2008)
17. Adleman, L., DeMarrais, J., Huang, M.: Quantum computability. SIAM J. Comput. **26** (1997) 1524–1540
18. Fortnow, L., Rogers, J.: Complexity limitations on quantum computation. In: Proc. 13th Annual IEEE Conference on Computational Complexity. (1998) 202–206
19. Barenco, A., Deutsch, D., Ekert, A., Jozsa, R.: Conditional quantum dynamics and logic gates. Physical Review Letters **74(20)** (1995) 4083–4086
20. Barenco, A., Bennett, C., Cleve, R., DiVincenzo, D., Margolus, N., Shor, P., Sleator, T., Smolin, J., Weinfurter, H.: Elementary gates for quantum computation. Phys. Rev. A **52(5)** (1995) 3457–3467
21. Boixo, S., Isakov, S.V., Smelyanskiy, V.N., Babbush, R., Ding, N., Jiang, Z., Bremner, M.J., Martinis, J.M., Neven, H.: Characterizing quantum supremacy in near-term devices. https://arxiv.org/pdf/1608.00263.pdf (2016)
22. Häner, T., Steiger, D.: 0.5 petabyte simulation of a 45-qubit quantum circuit. arXiv:1704.01127v1 (2017)
23. Feynmann, R.: Simulating physics with computers. International Journal of Theoretical Physics **21** (1982) 467–488
24. Feynmann, R.: Quantum mechanical computers. Foundation of Physics **16** (1986) 507–531
25. Deutsch, D.: Quantum theory, the Church-Turing principle, and the universal quantum computer. Proceedings of the Royal Society **A400** (1985) 97–117
26. Deutsch, D.: Quantum computational networks. In: Proceedings of the Royal Society of London. Volume 425(1868) of Series A. (1989) 73–90
27. Yamashita, S., Markov, I.: Fast equivalence-checking for quantum circuits. In: Proceedings of the 2010 IEEE/ACM Symposium on Nanoscale Architectures, Anaheim, CA USA. (2010) May 2013 update at https://arxiv.org/pdf/0909.4119.pdf.
28. Eggersglüß, S., Wille, R., Drechsler, R.: Improved SAT-based ATPG: More constraints, better compaction. In: Proceedings of the 2013 International Conference on Computer-Aided Design, San José, CA USA. (2013) 85–90
29. Schönhage, A., Strassen, V.: Schnelle Multiplikation grosser Zahlen. Computing Arch. Elektron. Rechnen **7** (1971) 281–292
30. van Meter, R., Itoh, K.: Fast quantum modular exponentiation. Phys. Rev. A **71** (2005) 052320
31. Markov, I., Saeedi, M.: Constant-optimized quantum circuits for modular multiplication and exponentiation. Quantum Information and Computation **12** (2012) 361–394
32. Pavlidis, A., Gizopoulos, D.: Fast quantum modular exponentiation architecture for shor's factoring algorithm. Quantum Information and Computation **14** (2014) 694–682
33. Cao, Z., Cao, Z., Liu, L.: Remarks on quantum modular exponentiation and some experimental demonstrations of Shor's algorithm. https://arxiv.org/abs/1408.6252 (2014)
34. Gottesman, D.: The Heisenberg representation of quantum computers. http://arxiv.org/abs/quant-ph/9807006 (1998)
35. Aaronson, S., Gottesman, D.: Improved simulation of stabilizer circuits. Phys. Rev. A **70** (2004)

36. Cai, J.Y., Chen, X., Lipton, R., Lu, P.: On tractable exponential sums. In: Proceedings of the 2010 Frontiers in Algorithms Workshop. Volume 6213 of Lect. Notes in Comp. Sci., Springer Verlag (2010) 48–59
37. Cai, J.Y., Chen, X., Lu, P.: Graph homomorphisms with complex values: A dichotomy theorem (2013)
38. Cai, J.Y., Lu, P., Xia, M.: The complexity of complex weighted Boolean #CSP. J. Comp. Sys. Sci. **80** (2014) 217–236
39. Jozsa, R.: Embedding classical into quantum computation. In: Proceedings of MMICS'08. (2008) 43–49 arXiv:quant-ph 0812.4511.
40. Spec.org, Butscher, B., H, W.: 462.libquantum spec cpu2006 benchmark description. https://www.spec.org/cpu2006/Docs/462.libquantum.html (2006)
41. Beckman, D., Chari, A., Devabhaktuni, S., Preskill, J.: Efficient networks for quantum factoring. Phys. Rev. A **54** (1996) 1034–1063
42. Markov, I., Saeedi, M.: Faster quantum number factoring via circuit synthesis. Phys. Rev. A **87** (2013) 012310 1–5
43. Beauregard, S.: Circuit for shor's algorithm using 2n+3 qubits. QUANTUM INFORMATION AND COMPUTATION **3** (2003) 175
44. Häner, T., Roetteler, M., Svore, K.: Factoring using 2n+2 qubits with toffoli based modular multiplication. Quantum Information and Computation **17** (2017)
45. Viamontes, G., Rajagopalan, M., Markov, I., Hayes, J.: Gate-level simulation of quantum circuits. In: Proceedings, ACM/ IEEE Asia and South-Pacific Design Automation Conf. (ASPDAC), Kitakyushu, Japan, January 2003. (2003) 295–301
46. Viamontes, G., Markov, I., Hayes, J.: Improving gate-level simulation of quantum circuits. Quantum Information Processing **2** (2003) 347–380
47. Greve, D.: Qdd: A quantum computer emulation library. http://thegreves.com/david/QDD/qdd.html (1999–2007)
48. Patrzyk, J., Patrzyk, B., Rycerz, K., Bubak, M.: Towards a novel environment for simulation of quantum computing. Computer Science **16** (2015) 103–129
49. Lee, Y., Khalil-Hani, M., Marsono, M.: An FPGA-based quantum computing emulation framework based on serial-parallel architecture. Journal of Reconfigurable Computing **2016** (2016) 18 pages
50. Barenco, A., Ekert, A., Suominen, K.A., Törmä, P.: Approximate quantum Fourier transform and decoherence. Phys. Rev. A **54** (1996) 139–146
51. Zilic, Z., Radecka, K.: Scaling and better approximating quantum fourier transform by higher radices. IEEE Trans. Comp. **56** (2007) 202–207
52. Rötteler, M., Beth, T.: Representation-theoretical properties of the approximate quantum Fourier transform. Applicable Algebra in Engineering, Communication and Computing **19** (2008) 117–193
53. Prokopenya, A.: Approximate quantum fourier transform and quantum algorithm for phase estimation. In: Computer Algebra in Scientific Computing. Volume 9301 of Lect. Notes in Comp. Sci., Springer Verlag (2015) 391–405
54. Aaronson, S., Chen, L.: Complexity-theoretic foundations of quantum supremacy experiments. https://arxiv.org/abs/1612.05903 (2016)