Symbolic Execution of Circuits with Superposition and Entanglement

Jacques Carette
McMaster University
Hamilton, ON, Canada

Gerardo Ortiz
Indiana University
Bloomington, IN, USA

Amr Sabry
Indiana University
Bloomington, IN, USA

ACM Reference Format:
Jacques Carette, Gerardo Ortiz, and Amr Sabry. 2022. Symbolic Execution of Circuits with Superposition and Entanglement. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

There are three common ways to compute with *unknown* values:

1. by treating as symbolic variables,
2. by using distributions,
3. by using unitary operators (i.e. qubits).

As a zeroth-order approximation, one can think of the "unitary operators" of quantum computing as $\mathbb{C}$-valued distributions. The extra "phase" involved allows tracking of simple relations between inputs (i.e. *entanglement*).

What we will do is to look at the classical reversible circuits at the heart of quantum computation, replacing the qubits by symbolic quantities. We look at a particular class of circuits with some *inputs* and some *ancillae* on one side, along with some *outputs* and the same ancillae restored to their initial values.

Furthermore, all the circuits look like ???

What a quantum circuit does with H on the ancilla is to transform a known input to (the equivalent of) the uniform distribution on all possible values. The result is basically *maximally unknown.*

What we do instead is to remove the H, and treat the ancillae as dynamic inputs, i.e. fully unknown. However, these circuits are all *reversible*, so we can also start from a partial measurement (static), treating the rest of the state as dynamic. We then execute the circuit backwards "symbolically".

While this sounds like the perfect setup for partial evaluation (PE) or slicing[1]. Since our circuits are reversible, for simplicity we're going to look at retrodictive execution as forward execution of the reverse program, which puts us in the setup of PE.

Except that in PE, the result is a *residual program* that obeys the Futumura equations. In our case, we have "both sides", i.e. input, output and ancillae, in our hands. So the result is going to be a set of equations constraining the "possible pasts" of the output to be equal to the actual inputs. So rather than obtaining a residual program as output, we obtain a *system of equations.*

It all works because of two key things: we're only dealing with boolean values and with circuits with operations (not, CX, CCX, and CCCX) that have simple symbolic forms. Furthermore, boolean expressions (over AND and XOR) themselves have a normal form. So we can propagate that through.

And now we give actual details.

2 Symbolic Representation of States

3 Partial Evaluation of Circuits

```
Usually program has variables
some known now
some known later
use actual values for variables known now
use symbols for variables known later
run with known values and residualize a program over late varia
```

For quantum circuits over H, CCX restrict to $|0\rangle$ inputs; look at state after the initial group of H gates

```
classical wires treated as known
superpositions |0> + |1> treated as symbols x
run circuit with known values and symbol for superpositions
```

4 Old

Quantum evolution is time-reversible and yet little advantage is gained from this fact in the circuit model of quantum computing. Indeed, most quantum algorithms expressed in the circuit model compute strictly from the present to the future, preparing initial states and proceeding forward with unitary transformations and measurements. In contrast, retrodictive quantum theory [?], retrocausality [?], and the

[1]These are basically the same idea, but PE propagates known information forwards through the program, and slicing propagates it backwards from the exit points

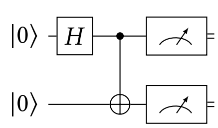
Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

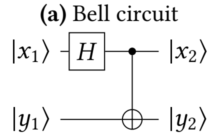
time-symmetry of physical laws [?] all suggest that quantum computation embodies richer –untapped– modes of computation, which could exploit knowledge about the future for a computational advantage.

Here we demonstrate that, in concert with the computational concepts of demand-driven lazy evaluation [?] and symbolic partial evaluation [?], retrodictive reasoning can indeed be used as a computational resource that exhibits richer modes of computation at the boundary of the classical/quantum divide. Specifically, instead of fully specifying the initial conditions of a quantum circuit and computing forward, it is possible to compute, classically, in both the forward and backward directions starting from partially specified initial and final conditions. Furthermore, this mixed mode of computation (i) can solve problems with fewer resources than the conventional forward mode of execution, sometimes even purely classically, (ii) can be expressed in a symbolic representation that immediately exposes global properties of the wavefunction that are needed for quantum algorithms, (iii) can lead to the de-quantization of some quantum algorithms, providing efficient classical algorithms inspired by their quantum counterparts, and (iv) reveals that the entanglement patterns inherent in genuine quantum algorithms with no known classical counterparts are artifacts of the chosen symbolic representation.

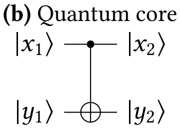
The main ideas underlying our contributions can be illustrated with the aid of the small examples in Fig. 1. In the conventional computational mode (Fig. 1a), the execution starts with the initial state $|00\rangle$. The first gate (Hadamard) evolves the state to $1/\sqrt{2}(|00\rangle + |10\rangle)$ which is transformed by the controlled-not (cx) gate to $1/\sqrt{2}(|00\rangle + |11\rangle)$. The measurements at the end produce 00 or 11 with equal probability. Fig. 1b keeps the quantum core of the circuit, removing the measurements, and naming the inputs and outputs with symbolic variables. Now, instead of setting $x_1 = y_1 = 0$ and computing forward as before, we can, for example, set $x_2 = 1$ and $y_2 = 0$ and calculate backwards as follows: $|10\rangle$ evolves in the backwards direction to $|11\rangle$ and then to $1/\sqrt{2}(|01\rangle - |11\rangle)$. In other words, in order to observe $x_2y_2 = 10$, the variable x_1 should be prepared in the superposition $1/\sqrt{2}(|0\rangle - |1\rangle)$ and y_1 should be prepared in the state $|1\rangle$. More interestingly, we can partially specify the initial and final conditions. For example, we can fix $x_1 = 0$ and $x_2 = 1$ and ask if there are any possible values for y_1 and y_2 that would be consistent with this setting. To answer the question, we calculate, using the techniques of symbolic partial evaluation (Methods), as follows. The initial state is $|0y_1\rangle$ which evolves to $1/\sqrt{2}(|0y_1\rangle + |1y_1\rangle)$ and then to $1/\sqrt{2}(|0y_1\rangle + |1(1 \oplus y_1)\rangle)$ where $\oplus$ is the exclusive-or operation and $1 \oplus y_1$ is the canonical way of negating y_1 in the Algebraic Normal Form (ANF) of boolean expressions (Methods). This final state can now be reconciled with the specified final conditions $1y_2$ revealing that the settings are consistent provided that $y_2 = 1 \oplus y_1$. We can, in fact, go one step further and analyze the circuit without the Hadamard gate as shown in Fig. 1c. The reasoning is that the role of Hadamard is to introduce (modulo phase) uncertainty about whether $x_1 = 0$ or $x_1 = 1$. But, again modulo phase, the same uncertainty can be expressed by just using the variable x_1. Thus, in Fig. 1c, we can set $y_1 = 0$ and $y_2 = 1$ and ask about values of x_1 and x_2 that would be consistent with this setting. We can calculate backwards from $|x_21\rangle$ as follows. The state evolves to $|x_2(1 \oplus x_2)\rangle$ which can be reconciled with the initial conditions yielding the constraints $x_1 = x_2$ and $1 \oplus x_2 = 0$ whose solutions are $x_1 = x_2 = 1$.



(a) Bell circuit

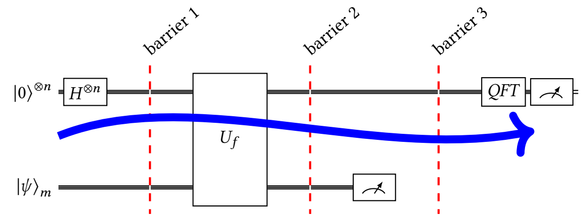


(b) Quantum core

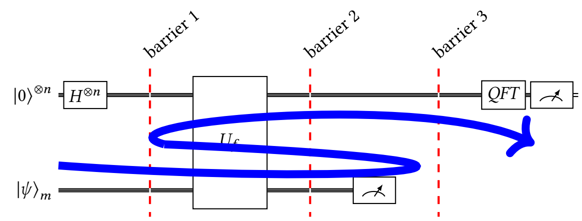


(c) Classical core

Figure 1. A conventional quantum circuit with initial conditions and measurement (a); its quantum core without measurement and with unspecified initial and final conditions (b); and its classical core without explicit quantum superpositions (c).



(a) Conventional Flow



(b) Retrodictive Flow

Figure 2. Template quantum circuit

These insights are robust and can be implemented in software (Methods) to analyze circuits with millions of gates for the quantum algorithms that match the template in Fig. 2 (including Deutsch, Deutsch-Jozsa, Bernstein-Vazirani, Simon, Grover, and Shor's algorithms [? ? ? ? ? ? ?]). The software is completely classical, performing mixed mode executions of the classical core of the circuits, i.e., the U_f block defined as $U_f(|x\rangle\,|y\rangle) = |x\rangle\,|f(x) \oplus y\rangle$. Specifically, in all these algorithms, the top collection of wires (which we will call the computational register) is prepared in a uniform superposition which can be represented using symbolic variables. The measurement of the bottom collection of wires (which we call the ancilla register) after barrier 2 provides partial information about the future which is, together with the initial conditions of the ancilla register, sufficient to symbolically execute the circuit. In each case, instead of the conventional execution flow depicted in Fig. 2(a), we find a possible measurement outcome w at barrier (2) and perform a symbolic retrodictive execution with a state $|xw\rangle$ going backwards to collect the constraints on x that enable us to solve the problem in question.

Algorithms.

The accompanying code includes retrodictive implementations of six major quantum algorithms: Deutsch, Deutsch-Jozsa, Bernstein-Vazirani, Simon, Grover, and Shor. We highlight the salient results for the first five algorithms, and then discuss the most interesting case of Shor's algorithm in detail.

De-Quantization. We abbreviate the set $\{0, 1, \ldots, (n-1)\}$ as $[\mathbf{n}]$. In the Deutsch-Jozsa problem, we are given a function $[\mathbf{2^n}] \to [\mathbf{2}]$ that is promised to be constant or balanced and we need to distinguish the two cases. The quantum circuit Fig. 3 shows the algorithm for the case $n = 1$. Instead of the conventional execution, we perform a retrodictive execution of the U_f block with an ancilla measurement 0, i.e., with the state $|x_{n-1} \cdots x_1 x_0 0\rangle$. The result of the execution is a symbolic formula r that determines the conditions under which $f(x_{n-1}, \cdots, x_0) = 0$. When the function is constant, the results are $0 = 0$ (always) or $1 = 0$ (never). When the function is balanced, we get a formula that mentions the relevant variables. For example, here are the results of three executions for balanced functions $[\mathbf{2^6}] \to [\mathbf{2}]$:

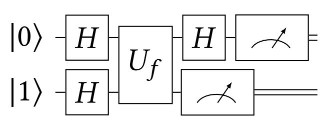


Figure 3. Quantum Circuit for the Deutsch-Jozsa Algorithm ($n = 1$)

- $x_0 = 0$,
- $x_0 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 = 0$, and
- $1 \oplus x_3x_5 \oplus x_2x_4 \oplus x_1x_5 \oplus x_0x_3 \oplus x_0x_2 \oplus x_3x_4x_5 \oplus x_2x_3x_5 \oplus x_1x_3x_5 \oplus x_0x_3x_5 \oplus x_0x_1x_4 \oplus x_0x_1x_2 \oplus x_2x_3x_4x_5 \oplus x_1x_3x_4x_5 \oplus x_1x_2x_4x_5 \oplus x_1x_2x_3x_5 \oplus x_0x_3x_4x_5 \oplus x_0x_2x_4x_5 \oplus x_0x_2x_3x_5 \oplus x_0x_1x_4x_5 \oplus x_0x_1x_3x_5 \oplus x_0x_1x_3x_4 \oplus x_0x_1x_2x_4 \oplus x_0x_1x_2x_4x_5 \oplus x_0x_1x_2x_3x_5 \oplus x_0x_1x_2x_3x_4 = 0$.

In the first case, the function is balanced because it produces 0 exactly when $x_0 = 0$ which happens half of the time in all possible inputs; in the second case the output of the function is the exclusive-or of all the input variables which is another easy instance of a balanced function. The last case is a cryptographically strong balanced function whose output pattern is balanced but, by design, difficult to discern [?].

An important insight is that we actually do not care about the exact formula. Indeed, since we are promised that the function is either constant or balanced, then any formula that refers to at least one variable must indicate a balanced function. In other words, the outcome of the algorithm can be immediately decided if the formula is anything other than 0 or 1. Indeed, our implementation correctly identifies all 12870 balanced functions $[\mathbf{2^4}] \to [\mathbf{2}]$. This is significant as some of these functions produce complicated entangled patterns during quantum evolution and could not be de-quantized using previous approaches [?]. A word of caution though: our results assume a "white-box" complexity model rather than a "black-box" complexity model [?].

The discussion above suggests that the details of the equations may not be particularly relevant for some algorithms. This would be crucial as the satisfiability of general boolean equations is, in general, an *NP*-complete problem [? ? ?]. Fortunately, this observation does hold for other algorithms as

$u = 0$	$1 \oplus x_3 \oplus x_2 \oplus x_1 \oplus x_0 \oplus x_2x_3 \oplus x_1x_3 \oplus x_1x_2 \oplus x_0x_3 \oplus x_0x_2 \oplus x_0x$ $\oplus x_0x_1x_3 \oplus x_0x_1x_2 \oplus x_0x_1x_2x_3$
$u = 1$	$x_0 \oplus x_0x_3 \oplus x_0x_2 \oplus x_0x_1 \oplus x_0x_2x_3 \oplus x_0x_1x_3 \oplus x_0x_1x_2 \oplus x_0x_1x_2x_3$
$u = 2$	$x_1 \oplus x_1x_3 \oplus x_1x_2 \oplus x_0x_1 \oplus x_1x_2x_3 \oplus x_0x_1x_3 \oplus x_0x_1x_2 \oplus x_0x_1x_2x_3$
$u = 3$	$x_0x_1 \oplus x_0x_1x_3 \oplus x_0x_1x_2 \oplus x_0x_1x_2x_3$
$u = 4$	$x_2 \oplus x_2x_3 \oplus x_1x_2 \oplus x_0x_2 \oplus x_1x_2x_3 \oplus x_0x_2x_3 \oplus x_0x_1x_2 \oplus x_0x_1x_2x_3$
$u = 5$	$x_0x_2 \oplus x_0x_2x_3 \oplus x_0x_1x_2 \oplus x_0x_1x_2x_3$
$u = 6$	$x_1x_2 \oplus x_1x_2x_3 \oplus x_0x_1x_2 \oplus x_0x_1x_2x_3$
$u = 7$	$x_0x_1x_2 \oplus x_0x_1x_2x_3$
$u = 8$	$x_3 \oplus x_2x_3 \oplus x_1x_3 \oplus x_0x_3 \oplus x_1x_2x_3 \oplus x_0x_2x_3 \oplus x_0x_1x_3 \oplus x_0x_1x_2x_3$
$u = 9$	$x_0x_3 \oplus x_0x_2x_3 \oplus x_0x_1x_3 \oplus x_0x_1x_2x_3$
$u = 10$	$x_1x_3 \oplus x_1x_2x_3 \oplus x_0x_1x_3 \oplus x_0x_1x_2x_3$
$u = 11$	$x_0x_1x_3 \oplus x_0x_1x_2x_3$
$u = 12$	$x_2x_3 \oplus x_1x_2x_3 \oplus x_0x_2x_3 \oplus x_0x_1x_2x_3$
$u = 13$	$x_0x_2x_3 \oplus x_0x_1x_2x_3$
$u = 14$	$x_1x_2x_3 \oplus x_0x_1x_2x_3$
$u = 15$	$x_0x_1x_2x_3$

Figure 4. Result of retrodictive execution for the Grover oracle ($n = 4$, w in the range $\{0..15\}$). The highlighted red subformula is the binary representation of the hidden input u.

Base	Equations				Solution
$a = 11$	$x_0 = 0$				$x_0 = 0$
$a = 4, 14$	$1 \oplus x_0 = 1$	$x_0 = 0$			$x_0 = 0$
$a = 7, 13$	$1 \oplus x_1 \oplus x_0 x_1 = 1$	$x_0 x_1 = 0$	$x_0 \oplus x_1 \oplus x_0 x_1 = 0$	$x_0 \oplus x_0 x_1 = 0$	$x_0 = x_1 = 0$
$a = 2, 8$	$1 \oplus x_0 \oplus x_1 \oplus x_0 x_1 = 1$	$x_0 x_1 = 0$	$x_1 \oplus x_0 x_1 = 0$	$x_0 \oplus x_0 x_1 = 0$	$x_0 = x_1 = 0$

Figure 5. Equations generated by retrodictive execution of $a^x \mod 15$ for different values of a, starting from observed result 1 and unknown $x_8x_7x_6x_5x_4x_3x_2x_1x_0$. The solution for the unknown variables is given in the last column.

well including the Bernstein-Vazirani algorithm and Grover's algorithm. In both cases, the result can be immediately read from the formula. In the Bernstein-Vazirani case, formulae are guaranteed to be of the form $x_1 \oplus x_3 \oplus x_4 \oplus x_5$; the secret string is then the binary number that has a 1 at the indices of the relevant variables $\{1, 3, 4, 5\}$. In the case for Grover, because there is a unique input u for which $f(u) = 1$, the ANF formula must include a subformula matching the binary representation of u, and in fact that subformula is guaranteed to be the shortest one as shown in Fig. 4.

Shor's Algorithm. The circuit in Fig. 6 uses a hand-optimized implementation of quantum oracle U_f for the modular exponentiation function $f(x) = 4^x \mod 15$ to factor 15 using Shor's algorithm. The white dot in the graphical representation of the first indicates that the control is active when it is 0. In a conventional forward execution, the state before the QFT block is:

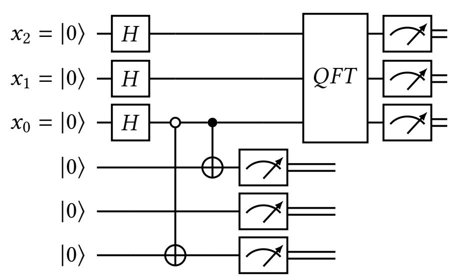


Figure 6. Finding the period of $4^x \mod 15$

$$\frac{1}{2\sqrt{2}}((|0\rangle + |2\rangle + |4\rangle + |6\rangle)\,|1\rangle + (|1\rangle + |3\rangle + |5\rangle + |7\rangle)\,|4\rangle)$$

At this point, the ancilla register is measured to either $|1\rangle$ or $|4\rangle$. In either case, the computational register snaps to a state of the form $\sum_{r=0}^{3} |a + 2r\rangle$ whose QFT has peaks at $|0\rangle$ or $|4\rangle$ making them the most likely outcomes of measurements of the computational register. If we measure $|0\rangle$, we repeat the experiment; otherwise we infer that the period is 2.

In the retrodictive execution, we can start with the state $|x_2x_1x_0001\rangle$ since 1 is guaranteed to be a possible ancilla measurement (corresponding to $f(0)$). The first cx-gate changes the state to $|x_2x_1x_0x_001\rangle$ and the second cx-gate produces $|x_2x_1x_0x_00x_0\rangle$. At that point, we reconcile the retrodictive result of the ancilla register $|x_00x_0\rangle$ with the initial condition $|000\rangle$ to conclude that $x_0 = 0$. In other words, in order to observe the ancilla at 001, the computational register must be initialized to a superposition of the form $|??0\rangle$ where the least significant bit must be 0 and the other two bits are unconstrained. Expanding the possibilities, the first register needs to be in a superposition of the states $|000\rangle$, $|010\rangle$, $|100\rangle$ or $|110\rangle$ and we have just inferred using purely classical but retrodictive reasoning that the period is 2.

This result does not, in fact, require the small optimized circuit of Fig. 6. In our implementation, modular exponentiation circuits are constructed from first principles using adders and multipliers [?]. In the case of $f(x) = 4^x \mod 15$, although the unoptimized constructed circuit has 56,538 generalized Toffoli gates (controlled^n-not gates for all n), the execution results in just two simple equations: $x_0 = 0$ and $1 \oplus x_0 = 1$. Furthermore, as shown in Fig. 5, the shape and size of the equations is largely insensitive to the choice of 4 as the base of the exponent, leading in all cases to the immediate conclusion that the period is either 2 or 4. When the solution is $x_0 = 0$, the period is 2, and when it is $x_0 = x_1 = 0$, the period is 4.

The remarkable effectiveness of retrodictive computation of the Shor instance for factoring 15 is due to a coincidence: a period that is a power of 2 is clearly trivial to represent in the binary number system which, after all is expressly designed for that purpose. That coincidence repeats itself when factoring products of the (known) Fermat primes: 3, 5, 17, 257, and 65537, and leads to small circuits [?]. This is confirmed with our implementation which smoothly deals with unoptimized circuits for factoring such products. Factoring 3*17=51 using the unoptimized circuit of 177,450 generalized Toffoli gates produces just the 4 equations: $1 \oplus x_1 = 1$, $x_0 = 0$, $x_0 \oplus x_0x_1 = 0$, and $x_1 \oplus x0x1 = 0$. Even for 3*65537=196611 whose circuit has 4,328,778 generalized Toffoli gates, the execution produces 16 small equations that refer to just the four variables x_0, x_1, x_2, and x_3 constraining them to be all 0, i.e., asserting that the period is 16.

Since periods that are powers of 2 are rare and special, we turn our attention to factoring

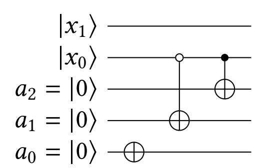


Figure 7. Finding the period of $4^x \mod 21$ using qutrits. The three gates are from left to right are the X, SUM, and $C(X)$ gates for ternary arithmetic [?]. The X gate adds 1 modulo 3; the controlled version $C(X)$ only increments when the control is equal to 2, and the SUM gates maps $|a, b\rangle$ to $|a, a + b\rangle$.

problems with other periods. The simplest such problem is that of factoring 21 with an underlying function $f(x) = 4^x \mod 21$ of period 3. The unoptimized circuit constructed from the first principles has 78,600 generalized Toffoli gates; its execution generates just three equations. But even in this rather trivial situation, the equations span 5 pages of text! (Supplementary Material). A small optimization reducing the number of qubits results in a circuit of 15,624 generalized Toffoli gates whose execution produces still quite large, but more reasonable, equations (Supplementary Material). To understand the reason for these unwieldy equations, we examine a general ANF formula of the form $X_1 \oplus X_2 \oplus X_3 \oplus \ldots = 0$ where each X_i is a conjunction of some boolean variables, i.e., the variables in each X exhibit constructive interference as they must all be true to enable that $X = 1$. Since the entire formula must equal to 0, every $X_i = 1$ must be offset by another $X_j = 1$, thus exhibiting negative interference among X_i and X_j. Generally speaking, arbitrary interference patterns can be encoded in the formulae at the cost of making the size of the formulae exponential in the number of variables. This exponential blowup is actually a necessary condition for any quantum algorithm that can offer an exponential speed-up over classical computation [?].

It would however be incorrect to conclude that factoring 21 is inherently harder than factoring 15. The issue is simply that the binary number system is well-tuned to expressing patterns over powers of 2 but a very poor match for expressing patterns over powers of 3. Indeed, we show that by just using qutrits, the circuit and equations for factoring 21 become trivial while those for factoring 15 become unwieldy. The manually optimized circuit in Fig. 7 consists of just three gates; its retrodictive execution produces two equations: $x_0 = 0$ and $x_0 \neq 2$, setting $x_0 = 0$ and leaving x_1 unconstrained. The matching values in the qutrit system as 00, 10, 20 or in decimal 0, 3, 6 clearly identifying the period to be 3. The idea of adapting the computation to simplify the circuit and equations is inspired by the fact that entanglement is relative to a particular tensor product decomposition (Methods).

Methods

Symbolic Execution of Classical Programs. A well-established technique to simultaneously explore multiple paths that a classical program could take under different inputs is *symbolic execution* [? ? ? ? ?]. In this execution scheme, concrete values are replaced by symbols which are initially unconstrained. As the execution proceeds, the symbols interact with program constructs and this typically introduces constraints on the possible values that the symbols represent. At the end of the execution, these constraints can be solved to infer properties of the program under consideration.

Entanglement. A symbolic variable represents a boolean value that can be 0 or 1; this is similar to a qubit in a superposition $(1/\sqrt{2})(|0\rangle \pm |1\rangle)$. Thus, it appears that $H|0\rangle$ could be represented by a symbol x to denote the uncertainty. Surprisingly, this idea scales to even represent maximally entangled states. Fig. 8 (left) shows a circuit to generate the Bell state $(1/\sqrt{2})(|00\rangle + |11\rangle)$. By using the symbol x for $H|0\rangle$, the input to the CX-gate is $|x0\rangle$ which evolves to $|xx\rangle$. By sharing the same symbol in two positions, the symbolic state accurately represents the entangled Bell state. Similarly, for the circuit in Fig. 8 (right), the state after the Hadamard gate is $|x00\rangle$ which evolves to $|xx0\rangle$ and then to $|xxx\rangle$ again accurately capturing the entanglement correlations.

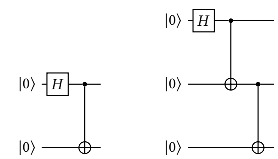


Figure 8. Bell and GHZ States

Given a maximally entangled state defined with respect to a particular tensor product decomposition, the same state may become unentangled in a different tensor product decomposition. Given the state:

$$|\Psi\rangle = |0\rangle + |3\rangle + |6\rangle + |9\rangle + |12\rangle + |15\rangle\,,$$

one can find a 4-qubit representation ($\mathcal{H} = \bigotimes_{i=1}^{4} \mathbb{C}^2$)

$$|\Psi\rangle = |0000\rangle + |0011\rangle + |0110\rangle + |1001\rangle + |1100\rangle + |1111\rangle\,,$$

where we used the following map $|m\rangle = \sum_{i=0}^{3} x_i 2^i$, with $m \in \mathbb{Z}$ and $x_i = 0, 1$. One can use the purity [?]

$$P_{|\Psi\rangle} = \frac{1}{4}\sum_{i=1}^{4}\sum_{\mu=x,y,z} \langle\Psi|\sigma_i^{\mu}|\Psi\rangle^2$$

and confirm that the state $|\Psi\rangle$ is maximally entangled, i.e., has $P_{|\Psi\rangle} = 0$. In contrast, in a qutrit basis ($\mathcal{H} = \bigotimes_{i=1}^{4} \mathbb{C}^3$), given the map $|m\rangle = \sum_{i=0}^{3} x_i 3^i$, with $x_i = 0, 1, 2$, the state

$$\begin{aligned} |\Psi\rangle &= |0000\rangle + |0010\rangle + |0020\rangle + |0100\rangle + |0110\rangle + |0120\rangle \\ &= |0\rangle \otimes (|0\rangle + |1\rangle) \otimes (|0\rangle + |1\rangle + |2\rangle) \otimes |0\rangle\,, \end{aligned}$$

is a product (unentangled) state.

Lazy Evaluation. Consider a program that searches for three different numbers x, y, and z each in the range $[1..n]$ and that sum to s. A well-established design principle for solving such problems is the *generate-and-test* computational paradigm. Following this principle, a simple program to solve this problem in the programming language Haskell is:

```
generate :: Int -> [(Int,Int,Int)]
generate n =  [(x,y,z) | x <- [1..n], y <- [1..n], z <- [1.

test :: Int -> [(Int,Int,Int)] -> [(Int,Int,Int)]
test s nums = [(x,y,z) | (x,y,z) <- nums, x /= y, x /= z, y
```

```
find :: Int -> Int -> (Int,Int,Int)
find s =  head . test s . generate
```

The program consists of three functions: `generate` that produces all triples `(x,y,z)` from `(1,1,1)` to `(n,n,n)`; `test` that checks that the numbers are different and that their sum is equal to `s`; and `find` that composes the two functions generating all triples, testing the ones that satisfy the condition, and returning the first solution. Running this program to find numbers in the range $[1..6]$ that sum to 15 immediately produces $(4, 5, 6)$ as expected.

But what if the range of interest was $[1..10000000]$? A naïve execution of the generate-and-test method would be prohibitively expensive as it would spend all its time generating an enormous number of triples that are un-needed. Lazy demand-driven evaluation as implemented in Haskell succeeds in a few seconds with the result $(1, 2, 12)$, however. The idea is simple: instead of eagerly generating all the triples, generate a process that, when queried, produces one triple at a time on demand. Conceptually the execution starts from the observer site which is asking for the first element of a list; this demand is propagated to the function `test` which itself propagates the demand to the function `generate` As each triple is generated, it is tested until one triple passes the test. This triple is immediately returned without having to generate any additional values.

Partial Evaluation. Below is a Haskell program that computes a^n by repeated squaring:

```
power :: Int -> Int -> Int
power a n
  | n == 0      = 1
  | n == 1      = a
  | even n      = let r = power a (n `div` 2) in r * r
  | otherwise   = a * power a (n-1)
```

When both inputs are known, e.g., `a = 3` and `n = 5` the program evaluates as follows:

```
   power 3 5
=  3 * power 3 4
=  3 * (let r1 = power 3 2 in r1 * r1)
=  3 * (let r1 = (let r2 = power 3 1 in r2 * r2) in r1 * r1)
=  3 * (let r1 = (let r2 = 3 in r2 * r2) in r1 * r1)
=  3 * (let r1 = 9 in r1 * r1)
=  243
```

Partial evaluation is used when we only have partial information about the inputs. Say we only know $n = 5$. A partial evaluator then attempts to evaluate `power` with symbolic input `a` and actual input `n=5` This evaluation proceeds as follows:

```
   power a 5
=  a * power a 4
=  a * (let r1 = power a 2 in r1 * r1)
=  a * (let r1 = (let r2 = power a 1 in r2 * r2) in r1 * r1)
=  a * (let r1 = (let r2 = a in r2 * r2) in r1 * r1)
=  a * (let r1 = a * a in r1 * r1)
=  let r1 = a * a in a * r1 * r1
```

All of this evaluation, simplification, and specialization happens without knowledge of `a`. Just knowing `n` was enough to produce a residual program that is much simpler.

Algebraic Normal Form (ANF).. The semantics of a generalized Toffoli gate with n control qubits: $a_{n_1}, \cdots, c_0$ and one target qubit b is $b \oplus \bigwedge_i a_i$, the exclusive-or of the target b with the conjunction of all the control qubits. That is precisely the definition of the algebraic normal form of boolean expressions. We note that circuits that only use x and cx-gates never generate any conjunctions and hence lead to formulae that efficiently solvable classically [? ?].

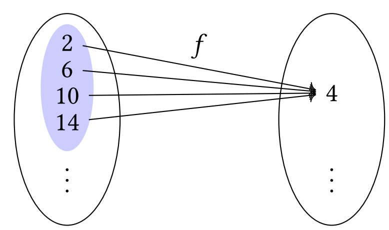


Figure 9. The pre-image of 4 under $f(x) = 7^x \mod 15$.

Complexity Analysis. Given finite sets A and B, a function $f : A \rightarrow B$ and an element $y \in B$, we define $\{\cdot \xleftarrow{f} y\}$, the pre-image of y under f, as the set $\{x \in A \mid f(x) = y\}$. For example, let $A = B = [2^4]$ and let $f(x) = 7^x \mod 15$, then the collection of values that f maps to 4, $\{\cdot \xleftarrow{f} 4\}$, is the set $\{2, 6, 10, 14\}$ as shown in Fig. 9. Symbolic retrodictive execution can be seen as a method to generate boolean formulae that describe the pre-image of the function f under study. For the example in Fig. 9, retrodictive execution might generate the formulae $x_1 = 1$ and $x_0 = 0$. The (trivial in this case) solution for the formulae is indeed the set $\{2, 6, 10, 14\}$. The critical points to note, however, are that: (i) solving the equations describing the pre-image is in general an intractable (even for quantum computers) *NP*-complete problem, and (ii) solving the equations is not needed for typical quantum algorithms. *Only some global properties of the pre-image are needed!* Indeed, we have already seen that for solving the Deutsch-Jozsa problem, the only thing needed was whether the formula contains some variables. For the Bernstein-Vazirani problem, the only thing needed was the indices of the variables occurring in the formula. For Grover's algorithm, we only need to extract the singleton element in the pre-image and for Shor's algorithm we "only" need to extract the periodicity of the elements in the pre-image.

To appreciate the difficulty of computing pre-images in general, note that finding the pre-image of a function subsumes several challenging computational problems such as pre-image attacks on hash functions [?], predicting environmental conditions that allow certain reactions to take place in computational biology [? ?], and finding the pre-image of feature vectors in the space induced by a kernel in neural networks [?]. More to the point, the boolean satisfiability problem SAT is expressible as a boolean function over the input variables and solving a SAT problem is asking for the pre-image of true. Indeed, based on the conjectured existence of one-way functions which itself implies $P \neq NP$, all these pre-images calculations are believed to be computationally intractable in their most general setting.

The bottleneck in retrodictive execution, as we presented it, is therefore in the symbolic execution of circuits in the ANF domain. Not only does the execution scale with the number of gates in the circuit but also with the size of the intermediate ANF formulae, which could become exponential in the number of variables.

Software. The entire suite of programs including synthesis of reversible circuits, standard evaluation, retrodictive evaluation under various modes, testing, debugging, and alternative representations off ANF formulae is only 1,500 lines of Haskell. The heart of the implementation is this simple function:

```
peG :: Value v => GToffoli s v -> ST s ()
peG (GToffoli bs cs t) = do
  controls <- mapM readSTRef cs
  tv <- readSTRef t
  let funs = map (\b -> if b then id else snot) bs
  let r = sxor tv (foldr sand one (zipWith ($) funs controls))
  writeSTRef t r
```

The function performs symbolic evaluation of one generalized Toffoli gates, reading the current ANF formulae for each control and producing an appropriate ANF formula for the target.

Data Availability. All execution results will be made available and can be replicated by executing the associated software.

Code Availability. The computer programs used to generate the circuits and symbolically execute the quantum algorithms retrodictively will be made publicly available.

Author Contributions. The idea of symbolic evaluation is due to A.S. The connection to retrodictive quantum mechanics is due to G.O. The connection to partial evaluation is due to J.C. Both A.S. and J.C. contributed to the software code to run the experiments. Both A.S. and G.O. contributed to the analysis of the quantum algorithms and their dequantization. All authors contributed to the writing of the document.

Competing Interests. No competing interests.

Materials & Correspondence. The corresponding author is Gerardo Ortiz.

Supplementary Information. The equations generated by retrodictive execution of the optimized circuit for 4^x mod 21 starting from observed result 1 and unknown x are:

$$1 \oplus x_0 \oplus x_1 \oplus x_2 \oplus x_0x_2 \oplus x_0x_1x_2 \oplus x_3 \oplus x_1x_3 \oplus x_0x_1x_3 \oplus x_0x_2x_3 \oplus x_1x_2x_3 \oplus x_4 \oplus x_0x_4 \oplus x_0x_1x_4 \oplus x_2x_4 \oplus x_1x_2x_4 \oplus x_0x_1x_2x_4 \oplus x_0x_3x_4 \oplus x_1x_3x_4 \oplus x_2x_3x_4 \oplus x_0x_2x_3x_4 \oplus x_0x_1x_2x_3x_4 \oplus x_5 \oplus x_1x_5 \oplus x_0x_1x_5 \oplus x_0x_2x_5 \oplus x_1x_2x_5 \oplus x_3x_5 \oplus x_0x_3x_5 \oplus x_0x_1x_3x_5 \oplus x_2x_3x_5 \oplus x_1x_2x_3x_5 \oplus x_0x_1x_2x_3x_5 \oplus x_0x_4x_5 \oplus x_1x_4x_5 \oplus x_2x_4x_5 \oplus x_0x_2x_4x_5 \oplus x_0x_1x_2x_4x_5 \oplus x_3x_4x_5 \oplus x_1x_3x_4x_5 \oplus x_0x_1x_3x_4x_5 \oplus x_0x_2x_3x_4x_5 \oplus x_1x_2x_3x_4x_5 = 1$$

$$x_1 \oplus x_0x_1 \oplus x_0x_2 \oplus x_1x_2 \oplus x_3 \oplus x_0x_3 \oplus x_0x_1x_3 \oplus x_2x_3 \oplus x_1x_2x_3 \oplus x_0x_1x_2x_3 \oplus x_0x_4 \oplus x_1x_4 \oplus x_2x_4 \oplus x_0x_2x_4 \oplus x_0x_1x_2x_4 \oplus x_3x_4 \oplus x_1x_3x_4 \oplus x_0x_1x_3x_4 \oplus x_0x_2x_3x_4 \oplus x_1x_2x_3x_4 \oplus x_5 \oplus x_0x_5 \oplus x_0x_1x_5 \oplus x_2x_5 \oplus x_1x_2x_5 \oplus x_0x_1x_2x_5 \oplus x_0x_3x_5 \oplus x_1x_3x_5 \oplus x_2x_3x_5 \oplus x_0x_2x_3x_5 \oplus x_0x_1x_2x_3x_5 \oplus x_4x_5 \oplus x_1x_4x_5 \oplus x_0x_1x_4x_5 \oplus x_0x_2x_4x_5 \oplus x_1x_2x_4x_5 \oplus x_3x_4x_5 \oplus x_0x_3x_4x_5 \oplus x_0x_1x_3x_4x_5 \oplus x_2x_3x_4x_5 \oplus x_1x_2x_3x_4x_5 \oplus x_0x_1x_2x_3x_4x_5 = 0$$

$$x_0 \oplus x_0x_1 \oplus x_2 \oplus x_1x_2 \oplus x_0x_1x_2 \oplus x_0x_3 \oplus x_1x_3 \oplus x_2x_3 \oplus x_0x_2x_3 \oplus x_0x_1x_2x_3 \oplus x_4 \oplus x_1x_4 \oplus x_0x_1x_4 \oplus x_0x_2x_4 \oplus x_1x_2x_4 \oplus x_3x_4 \oplus x_0x_3x_4 \oplus x_0x_1x_3x_4 \oplus x_2x_3x_4 \oplus x_1x_2x_3x_4 \oplus x_0x_1x_2x_3x_4 \oplus x_0x_5 \oplus x_1x_5 \oplus x_2x_5 \oplus x_0x_2x_5 \oplus x_0x_1x_2x_5 \oplus x_3x_5 \oplus x_1x_3x_5 \oplus x_0x_1x_3x_5 \oplus x_0x_2x_3x_5 \oplus x_1x_2x_3x_5 \oplus x_4x_5 \oplus x_0x_4x_5 \oplus x_0x_1x_4x_5 \oplus x_2x_4x_5 \oplus x_1x_2x_4x_5 \oplus x_0x_1x_2x_4x_5 \oplus x_0x_3x_4x_5 \oplus x_1x_3x_4x_5 \oplus x_2x_3x_4x_5 \oplus x_0x_2x_3x_4x_5 \oplus x_0x_1x_2x_3x_4x_5 = 0$$

The equations generated by retrodictive execution of the unoptimized 4^x mod 21 starting from observed result 1 and unknown x. The circuit consists of 36,400 CX-gates, 38,200 CCX-gates, and 4,000 CCCX-gates. There are only three equations but each equation is exponentially large.

References

[1] Abbott, A. A. The Deutsch-Jozsa problem: de-quantization and entanglement. *Natural Computing 11* (2012).

[2] Aharonov, Y., and Vaidman, L. *The Two-State Vector Formalism: An Updated Review.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 399–447.

[3] Akutsu, T., Hayashida, M., Zhang, S.-Q., Ching, W.-K., and Ng, M. K. Analyses and algorithms for predecessor and control problems for boolean networks of bounded indegree. *Information and Media Technologies 4*, 2 (2009), 338–349.

[4] Baldoni, R., Coppa, E., D'elia, D. C., Demetrescu, C., and Finocchi, I. A survey of symbolic execution techniques. *ACM Comput. Surv. 51*, 3 (may 2018).

[5] Barnett, S. M., Jeffers, J., and Pegg, D. T. Quantum retrodiction: Foundations and controversies. *Symmetry 13*, 4 (2021).

[6] Barnum, H., Knill, E., Ortiz, G., Somma, R., and Viola, L. A subsystem-independent generalization of entanglement. *Phys. Rev. Lett. 92* (Mar 2004), 107902.

[7] Bernstein, E., and Vazirani, U. Quantum complexity theory. *SIAM Journal on Computing 26*, 5 (1997), 1411–1473.

[8] Bocharov, A., Cui, S. X., Roetteler, M., and Svore, K. M. Improved quantum ternary arithmetic. *Quantum Info. Comput. 16*, 9–10 (jul 2016), 862–884.

[9] Boyer, R. S., Elspas, B., and Levitt, K. N. Select—a formal system for testing and debugging programs by symbolic execution. *SIGPLAN Not. 10*, 6 (apr 1975), 234–245.

[10] Burnett, L., Millan, W., Dawson, E., and Clark, A. Simpler methods for generating better boolean functions with good cryptographic properties. *Australasian Journal of Combinatorics 29* (2004), 231–247.

[11] Clarke, L. A. A program testing system. In *Proceedings of the 1976 Annual Conference* (New York, NY, USA, 1976), ACM '76, Association for Computing Machinery, p. 488–491.

[12] Cook, S. A. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 1971), STOC '71, Association for Computing Machinery, p. 151–158.

[13] Deutsch, D. Quantum theory, the church–turing principle and the universal quantum computer. *Proc. R. Soc. Lond. A 400* (1985).

[14] Deutsch, D., and Jozsa, R. Rapid solution of problems by quantum computation. *Proc. R. Soc. Lond. A 439* (1992).

[15] Futamura, Y. Partial computation of programs. In *RIMS Symposia on Software Science and Engineering* (Berlin, Heidelberg, 1983), E. Goto, K. Furukawa, R. Nakajima, I. Nakata, and A. Yonezawa, Eds., Springer Berlin Heidelberg, pp. 1–35.

[16] Geller, M. R., and Zhou, Z. Factoring 51 and 85 with 8 qubits. *Scientific Reports (3023) 3*, 1 (2013).

[17] Grover, L. K. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 1996), STOC '96, Association for Computing Machinery, p. 212–219.

[18] Henderson, P., and Morris, J. H. A lazy evaluator. In *Proceedings of the 3rd ACM SIGACT-SIGPLAN Symposium on Principles on Programming Languages* (New York, NY, USA, 1976), POPL '76, Association for Computing Machinery, p. 95–103.

[19] Howden, W. E. Experiments with a symbolic evaluation system. In *Proceedings of the National Computer Conference* (1976).

[20] Jozsa, R., and Linden, N. On the role of entanglement in quantum-computational speed-up. *Proceedings: Mathematical, Physical and Engineering Sciences 459*, 2036 (2003), 2011–2032.

[21] Karp, R. M. *Reducibility among Combinatorial Problems.* Springer US, Boston, MA, 1972, pp. 85–103.

[22] King, J. C. Symbolic execution and program testing. *Commun. ACM 19*, 7 (jul 1976), 385–394.

[23] Klotz, J. G., Bossert, M., and Schober, S. Computing preimages of boolean networks. *BMC Bioinformatics 14*, 10 (Aug 2013), S4.

[24] Komargodski, I., Naor, M., and Yogev, E. White-box vs. black-box complexity of search problems: Ramsey and graph property testing. *J. ACM 66*, 5 (jul 2019).

[25] Kwok, J.-Y., and Tsang, I.-H. The pre-image problem in kernel methods. *IEEE Transactions on Neural Networks 15*, 6 (2004), 1517–1525.

[26] Nielsen, M. A., and Chuang, I. L. *Quantum Computation and Quantum Information: 10th Anniversary Edition.* Cambridge University Press, 2010.

[27] Rogaway, P., and Shrimpton, T. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *Fast Software Encryption* (Berlin, Heidelberg, 2004), B. Roy and W. Meier, Eds., Springer Berlin Heidelberg, pp. 371–388.

[28] Shor, P. W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing 26*, 5 (1997), 1484–1509.

[29] Simon, D. On the power of quantum computation. In *Proceedings 35th Annual Symposium on Foundations of Computer Science* (1994), pp. 116–123.

[30] Tokareva, N. Chapter 1 - Boolean functions. In *Bent Functions*, N. Tokareva, Ed. Academic Press, Boston, 2015, pp. 1–15.

[31] Trakhtenbrot, B. A survey of russian approaches to perebor (brute-force searches) algorithms. *Annals of the History of Computing 6*, 4 (1984), 384–400.

[32] Vedral, V., Barenco, A., and Ekert, A. Quantum networks for elementary arithmetic operations. *Phys. Rev. A 54* (Jul 1996), 147–153.

[33] Watanabe, S. Symmetry of physical laws. Part III. prediction and retrodiction. *Rev. Mod. Phys. 27* (Apr 1955), 179–186.

[34] Wegener, I. *The Complexity of Boolean Functions.* John Wiley & Sons, Inc., USA, 1987.