PNAS RESEARCH ARTICLE | COMPUTER SCIENCES

A formally certified end-to-end implementation of Shor's factorization algorithm

Yuxiang Peng[a,b], Kesha Hietala[a], Runzhou Tao[c], Liyi Li[a,b], Robert Rand[d], Michael Hicks[a,b], and Xiaodi Wu[a,b,1]

Edited by Jeffrey Ullman, Stanford University (Retired), Stanford, CA; received November 2, 2022; accepted April 21, 2023

Quantum computing technology may soon deliver revolutionary improvements in algorithmic performance, but it is useful only if computed answers are correct. While hardware-level decoherence errors have garnered significant attention, a less recognized obstacle to correctness is that of human programming errors—"bugs." Techniques familiar to most programmers from the classical domain for avoiding, discovering, and diagnosing bugs do not easily transfer, at scale, to the quantum domain because of its unique characteristics. To address this problem, we have been working to adapt formal methods to quantum programming. With such methods, a programmer writes a mathematical specification alongside the program and semiautomatically proves the program correct with respect to it. The proof's validity is automatically confirmed—certified—by a "proof assistant." Formal methods have successfully yielded high-assurance classical software artifacts, and the underlying technology has produced certified proofs of major mathematical theorems. As a demonstration of the feasibility of applying formal methods to quantum programming, we present a formally certified end-to-end implementation of Shor's prime factorization algorithm, developed as part of a framework for applying the certified approach to general applications. By leveraging our framework, one can significantly reduce the effects of human errors and obtain a high-assurance implementation of large-scale quantum applications in a principled way.

quantum programming | formal methods | Shor's algorithm | human errors

Significance

While hardware errors have garnered significant attention as the major obstacle to quantum computing, the error due to human factors in the implementation is less recognized. This paper identifies human programming errors as another important source of errors, whereas most existing techniques from the classical domain fail to transfer at scale to quantum programming. The adaptation of formal methods to quantum programming is proposed as a solution that circumvents quantum unique challenges and leads to the high-assurance implementation of large-scale quantum applications in a principle way. As a demonstration of the feasibility, this paper presents a formally certified end-to-end implementation of Shor's prime factorization algorithm due to its complexity and importance.

As developments in quantum computer hardware bring promising quantum applications closer to reality, a key question to contend with is "How can we be sure that a quantum computer program, when executed, will give the right answer?" A well-recognized threat to correctness is quantum computer hardware, which is susceptible to decoherence errors. Techniques to provide hardware-level fault tolerance are under active research (1, 2). A less recognized threat comes from errors—bugs—in the program itself, as well as errors in the software that prepares a program to run on a quantum computer (compilers, linkers, etc.). In the classical domain, program bugs are commonplace and are sometimes the source of expensive and catastrophic failures or security vulnerabilities.

Quantum programs that provide a performance advantage over their classical counterparts are even more challenging to write, understand, and certify (Fig. 1*A*). They often involve the use of randomized algorithms and leverage unfamiliar quantum-specific concepts, including superposition, entanglement, and destructive measurement. Quantum programs are also hard to test. To debug a failing test, programmers cannot easily observe (measure) an intermediate state due to the destructive nature of quantum measurement. Moreover, many quantum algorithms generate samples over an exponentially large output domain, whose statistical properties could require exponentially many samples to be verified information-theoretically. Simulating a quantum program on a classical computer can help but is limited by such computers' ability to faithfully represent a quantum state of even modest size (which is why we must build quantum hardware). The fact that near-term quantum computers are error-prone adds another layer of difficulty.

Proving Programs Correct with Formal Methods. As a potential remedy to these problems, we have been exploring how to use formal methods (aka formal verification) to develop quantum programs (Fig. 1*B*). Formal methods are processes and techniques by which one can mathematically prove that software does what it should, for all inputs; the proved-correct artifact is referred to as formally certified. The formal verification is usually conducted by using a proof assistant, which is a software tool for formalizing mathematical definitions and stating and proving properties about them. A proof assistant

Author contributions: M.H. and X.W. designed research; Y.P., K.H., R.T., L.L., R.R., M.H., and X.W. performed research; and Y.P., K.H., R.R., M.H., and X.W. wrote the paper.

The authors declare no competing interest.

This article is a PNAS Direct Submission.

[1]To whom correspondence may be addressed. Email: xiaodiwu@umd.edu.

This article contains supporting information online at http://www.pnas.org/lookup/suppl/doi:10.1073/pnas.2218775120/-/DCSupplemental.

Published May 15, 2023.

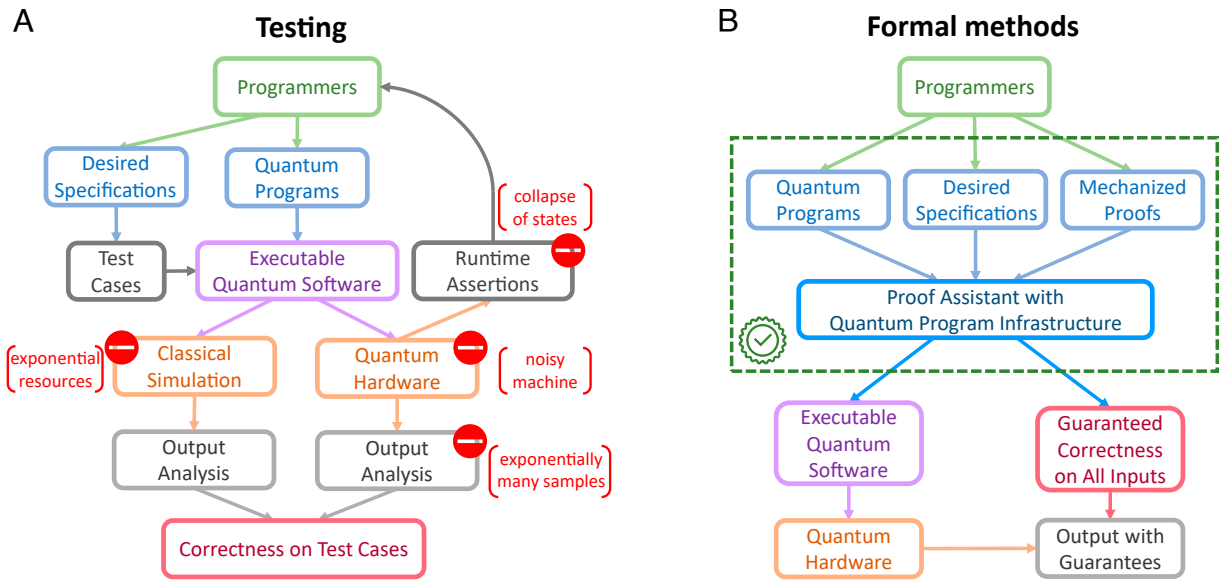

Fig. 1. Comparison of developing quantum programs with (*A*) testing and with (*B*) formal methods. The major classical routines of debugging with testing are blocked by quantum-specific features and can guarantee correctness only in test cases. In the formal methods approach, a proof assistant will mechanically certify quantum programs along with their specifications and proofs. If successful, the certified implementation is guaranteed to meet desired specifications on all possible inputs, even without running the program on real machines.

may produce proofs automatically or assist a human in doing so, interactively. Either way, the proof assistant confirms that a proof is correct by employing a proof verifier. Most modern proof assistants implement proof verification by leveraging the Curry–Howard correspondence, which embodies a surprising and powerful analogy between formal logic and programming language type systems (3, 4). Automating and confirming such proofs have, for more than 50 years, been a grand challenge for computing research (5) (*SI Appendix*, section 1).

While early developments of formal methods led to disappointment (6), the last two decades have seen remarkable progress. Notable successes include the development of the seL4 microkernel (7) and the CompCert C compiler (8). For the latter, the benefits of formal methods have been demonstrated empirically: Using sophisticated testing techniques, researchers found hundreds of bugs in the popular mainstream C compilers `gcc` and `clang`, but none in CompCert's verified core (9). Formal methods have also been successfully deployed to prove major mathematical theorems [e.g., the four color theorem (10)] and build computer-assisted proofs in the grand unification theory of mathematics (11, 12).

Formal Methods for Quantum Programs. Our key observation is that the symbolic reasoning that underlies formal methods is not limited by the aforementioned difficulties of testing directly on quantum machines or classically simulating them. As a result, it may be a viable alternative to certifying the correctness of quantum programs. Our research has explored how to put this observation into practice. Precisely, using the Coq proof assistant (13), we defined a simple quantum intermediate representation (14) (SQIR) for expressing a quantum program as a series of operations—essentially a kind of circuit—and specified those operations' mathematical meaning. Thus, we can state the mathematical properties of an SQIR program and prove that they always hold without needing to run that program. Assured that the program is correct, we can run it on specific inputs by asking Coq to extract the SQIR program to an OpenQASM 2.0 circuit and then run it on a real machine.

Adapting formal methods developed for classical programs to work on quantum ones is conceptually straightforward but pragmatically challenging. Consider that classical program states are (in the simplest terms) maps from addresses to bits (0 or 1); thus, a state is essentially a length-n vector of Booleans. Quantum states are much more involved: In SQIR, an n-qubit quantum state is represented as a length-2^n vector of complex numbers, and the meaning of an n-qubit operation is represented as a $2^n \times 2^n$ matrix—applying an operation to a state is tantamount to multiplying the operation's matrix with the state's vector. Proofs over all possible inputs thus involve translating such multiplications into symbolic formulae and then reasoning about them.

Given the potentially large size of quantum states, such formulae could become quite large and difficult to reason about. To cope, we developed automated tactics to translate symbolic states into normalized algebraic forms, making them more amenable to automated simplification. We also eschew matrix-based representations entirely when an operation can be expressed symbolically in terms of its action on basis states. With these techniques and others (15), we proved the correctness of key components of several quantum algorithms—Grover's search algorithm (16) and quantum phase estimation (QPE) (17)—and demonstrated advantages over competing approaches (18–21).

With this promising foundation in place, several challenges remain. First, both Grover's and QPE are parameterized by oracles, which are classical algorithmic components that must be implemented to run on quantum hardware. These must be reasoned about, too, but they can be large (many times larger than an algorithm's quantum scaffold) and can be challenging to encode for quantum processing, bug-free. Another challenge is proving the end-to-end properties of hybrid quantum/classical algorithms. These algorithms execute code on both classical and quantum computers to produce a final result. Such algorithms are likely to be common in near-term deployments in which quantum processors complement classical ones. Finally, end-to-end certified software must implement and reason about probabilistic algorithms, which are correct with a certain probability and may require multiple runs.

1. Certified Implementations

Shor's Algorithm and the Benefit of Formal Methods. To close these gaps, and thereby demonstrate the feasibility of the application of formal methods to quantum programming, we have produced a fully certified version of Shor's prime factorization algorithm (17), which is famous for breaking widely used RSA cryptographic systems. This algorithm has been a fundamental motivation for the development of quantum computers and is at a scale and complexity not reached in prior formalization efforts.

As shown in Fig. 2, Shor developed a sophisticated, quantum-classical hybrid algorithm to factor a number N: the key quantum part—order finding—preceded and followed by classical computation—primality testing before and conversion of found orders to prime factors, after. The algorithm's correctness proof critically relies on arguments about both its quantum and classical parts and also on several number-theoretical arguments.

While it is difficult to factor a number, it is easy to confirm a proposed factorization (the factoring problem is inside the NP complexity class). One might wonder why prove a program correct if we can always efficiently check its output? When the check shows that an output is wrong, this fact does not help with computing the correct output and provides no hint about the source of the implementation error. By contrast, formal verification allows us to identify the source of the error: It is precisely in the subprogram that we could not certify.

Moreover, because inputs are reasoned about symbolically, the complexity of all-input certification can be (much) less than the complexity of single-output correctness checking. For example, one can symbolically verify that a quantum circuit generates a uniform distribution over n bits, but directly checking whether the output samples from a uniform distribution over n bits could take as many as $2^{\Theta(n)}$ samples (22). As such, with formal methods, one could potentially certify implementations for major quantum applications, like quantum simulation which is BQP-complete (23) and believed to lie outside NP.

Overview of Our Implementation. An instantiation of the scheme in Fig. 1*B* for Shor's algorithm is given in Fig. 3 *A* and *B*. We have certified the implementation against both a correctness specification (e.g., the likelihood of success) and a resource specification (e.g., the gate count) (*SI Appendix*, Fig. S2). Note that the implementation is built on fault-tolerant logical qubits and does not include their realization based on physical qubits.

The core of the algorithm is the computation of the order r of a *modulo* N, where a is (uniformly) randomly drawn from the numbers 1 through N; this component is bounded by the dark box in Fig. 2. The quantum component of order finding applies QPE to an oracle implementing an in-place modular multiplier (IMM), whereas the correctness of QPE was previously proved in SQIR with respect to an abstract oracle (15). The IMM oracle corresponds to pure classical reversible computation when executed coherently, but SQIR was not able to leverage this fact to simplify the proof.

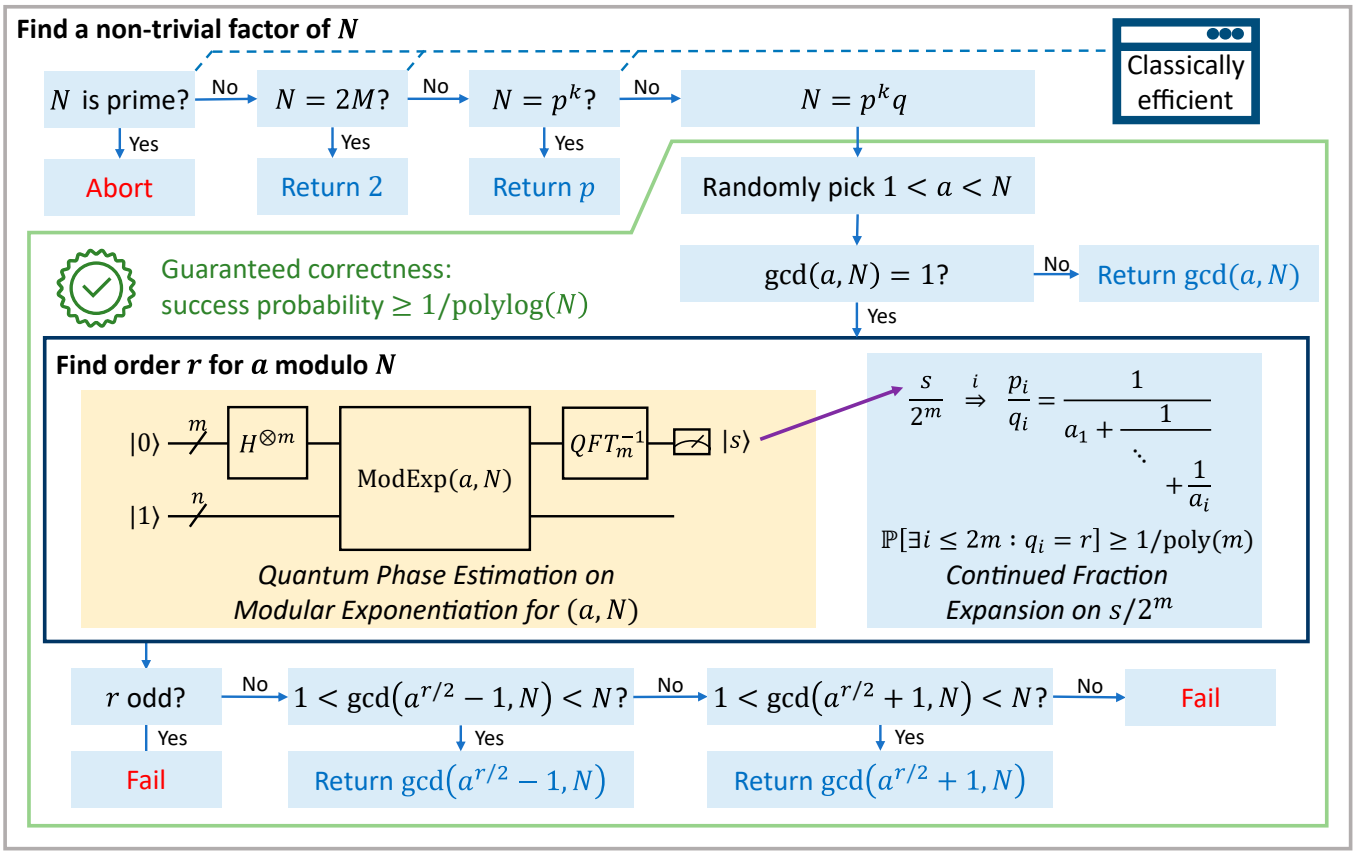

Fig. 2. Overview of Shor's factoring algorithm, which finds a nontrivial factor of integer N (not 1 or N) with high probability in polynomial time. This is a quantum-classical hybrid algorithm, whose quantum part (marked cream) is a subprocedure of finding multiplicative orders (enclosed in the blue frame). We implement and mechanically certified Shor's algorithm (enclosed in the green frame), for N not prime, even, or a prime power (these cases can be efficiently tested for and solved by classical algorithms). A detailed illustration of our implementation of ModExp(a, N) is in *SI Appendix*, Fig. S1.

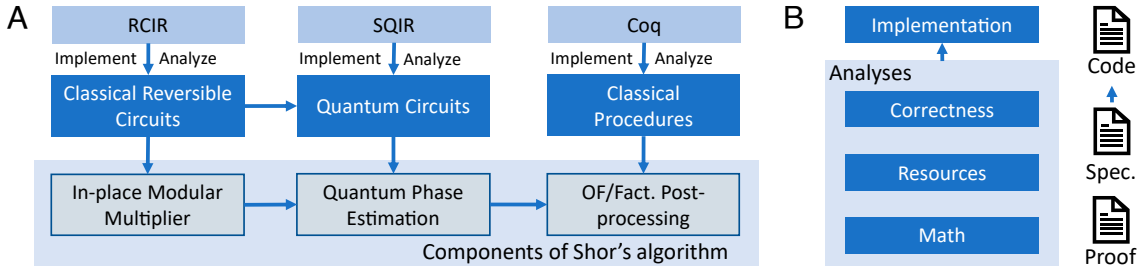

Fig. 3. Technical illustration of our fully certified implementation of Shor's algorithm. (*A*) The schematic framework of our implementation in Coq. Intermediate representations SQIR and RCIR are embedded in Coq, dealing with classical reversible and quantum circuits, respectively. (*B*) An instantiation of the formal methods scheme in Shor's implementation.

In response, we developed the reversible circuit intermediate representation (RCIR) in Coq to express classical functions and prove their correctness, which can be translated into SQIR as shown in Fig. 3*A*. RCIR helps us easily build the textbook version of IMM (24) (*SI Appendix*, Fig. S1) and prove its correctness and resource usage (*SI Appendix*, Fig. S2, *i*). Integrating the QPE implementation in SQIR with the translation of IMM's implementation from RCIR to SQIR, we implement the quantum component of order-finding as well as the proof for its correctness and gate count bound (*SI Appendix*, Fig. S2, *ii*). The QPE correctness statement asserts that the likelihood of obtaining the closest phase estimate of the input eigenstate is no less than $4/\pi^2$, mechanically proved through expressing the final state as matrix multiplications and scaling trigonometrical functions (details in *SI Appendix*, section 1.E). We then formulate a decomposition with the certified complex linear algebra in SQIR for the input state $|1\rangle$ in Fig. 2 into r eigenstates of IMM, $|1\rangle = \sum_{k=0}^{r-1} |\psi_k\rangle/\sqrt{r}$, where $|\psi_k\rangle$ possesses the eigenvalue $e^{-2\pi ik/r}$. Applying QPE over IMM while inputting this uniform superposition of eigenstates generates a close approximation of k/r for any k from $\{0, 1, \cdots, r-1\}$ with probability at least $4/\pi^2 r$ (*SI Appendix*, Fig. S2, *ii*) following a similar analysis of QPE.

After executing the quantum part of the algorithm, some classical code carries out continued fraction expansion (CFE) to recover the order r. CFE is an iterative algorithm and its efficiency to recover k/r in terms of the number of iterations is guaranteed by Legendre's theorem which we formulated and constructively proved in Coq with respect to the CFE implementation. When the recovered k and r are coprimes, the output r is the correct order. The algorithm is probabilistic, and the probability that coprime k and r are output is lower-bounded by the size of $\mathbb{Z}_r$, which consists of all positive integers that are smaller than r and coprime to it. The size of $\mathbb{Z}_r$ is the definition of the famous Euler's totient function $\varphi(r)$, which we proved is at least $e^{-2}/\lfloor\log(r)\rfloor^4$ in Coq based on the formalization of Euler's product formula and Euler's theorem by de Rauglaudre (25). By integrating the proofs for both quantum and classical components, we show that our implementation of the entire hybrid order-finding procedure will identify the correct order r for any a given that $\gcd(a, N) = 1$ with probability at least $4e^{-2}/\pi^2\lfloor\log_2(N)\rfloor^4$ (*SI Appendix*, Fig. S2, *iii*).

For the overall algorithm, we prove that the order finding procedure combined with the classical postprocessing will output a nontrivial factor with a success probability of at least $2e^{-2}/\pi^2\lfloor\log_2(N)\rfloor^4$, which is exactly half of the success probability of order finding. Namely, we prove that for at least a half of the integers a between 1 and N, the order r will be even and either $\gcd(a^{r/2}+1, N)$ or $\gcd(a^{r/2}-1, N)$ will be a nontrivial factor of N. Shor's original proof (17) of this property made use of the existence of the group generator of $\mathbb{Z}_{p^k}$, also known as primitive roots, for odd prime p. However, the known proof of the existence of primitive roots is nonconstructive (26) meaning that it makes use of axioms like the law of the excluded middle, whereas one needs to provide constructive proofs (27) in Coq and other proof assistants.

We provide a constructive proof without using primitive roots by resorting to the quadratic residues in modulus p^k and connecting whether a randomly chosen a leads to a nontrivial factor to the number of quadratic residues and nonresidues in modulus p^k. The counting of the latter is established based on Euler's criterion for distinguishing between quadratic residues and nonresidues modulo p^k in Coq.

Putting it all together, we have proved that our implementation of Shor's algorithm successfully outputs a nontrivial factor with a probability of at least $2e^{-2}/\pi^2\lfloor\log_2(N)\rfloor^4$ for one iteration. Furthermore, we also prove in Coq that its failure probability of t repetitions is upper-bounded by $(1 - 2e^{-2}/\pi^2\lfloor\log_2(N)\rfloor^4)^t$, which boosts the success probability of our implementation arbitrarily close to 1 after $O(\log^4(N))$ repetitions.

We also certify that the gate count in our implementation of Shor's algorithm using OpenQASM's gate set is upper-bounded by $(212n^2 + 975n + 1031)m + 4m + m^2$ in Coq, where n refers to the number of bits representing N and m the number of bits in QPE output. Note further $m, n = O(\log N)$, which leads to an $O(\log^3 N)$ overall asymptotic complexity that matches the original paper. All these certification details are in *SI Appendix*, section 3.

2. Executions

Although the proof assistant accepts the specification theorems of our implementation of Shor's algorithm, there still may be misalignment between the specification statements and the program's correctness. To empirically confirm the absence of such misalignment, we execute our certified-in-Coq implementation of Shor's algorithm through extraction (28) (Fig. 4*A*) and observe the correctness of example cases. Due to the limitation of existing quantum machines, we use a classical simulator called DDSIM (29) to execute these quantum circuits, which necessarily limits the scale of our empirical study. In Fig. 4*B*, we showcase the details of factorization with $N = 15$ based on the simulation (and see *SI Appendix*, Fig. S3 for order finding). Note that existing experimental demonstrations of Shor's algorithm for $N = 15$ or 21 (e.g., refs. 30 and 31) require fewer resources

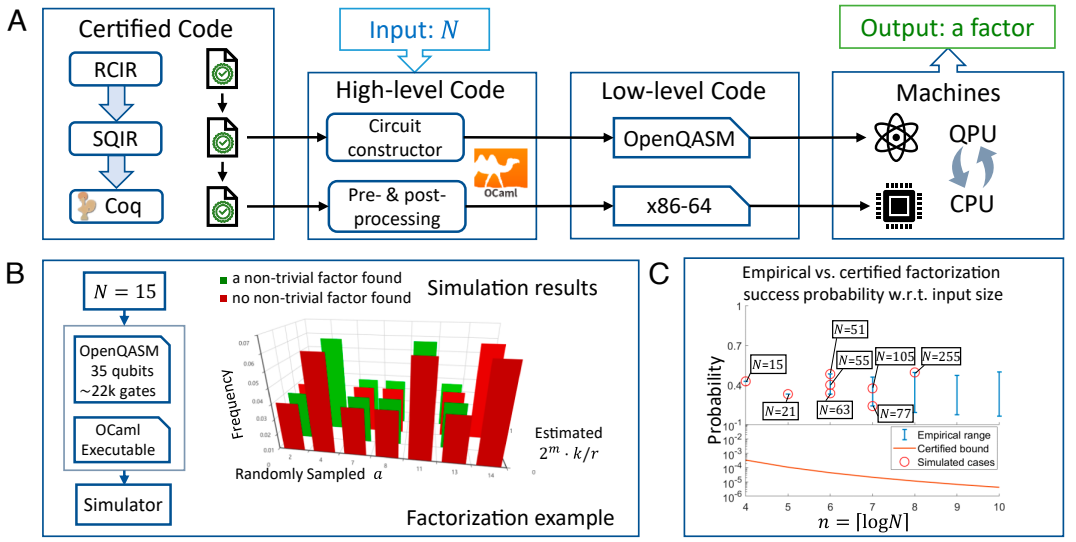

Fig. 4. End-to-End Execution of Our Implementation of Shor's algorithm. (*A*) A schematic illustration of the end-to-end quantum-classical hybrid execution. (*B*) An example of end-to-end execution and simulation of factorization. (*C*) Empirical statistics (i.e., minimal to maximal success probability) of the success probability of factorization for every valid input *N* with respect to input size *n* from 4 to 10 bits.

because their implementations are specially designed for fixed inputs and cannot extend to work for general ones. In Fig. 4*C*, we conduct a more comprehensive empirical study on the gate count and success probability of factorization instances with input size ($\log(N)$) from 4 to 10 bits, i.e., $N \leq 1{,}024$. Red circles refer to instances (i.e., a few specific Ns) that can be simulated by DDSIM. The empirical success probabilities for other Ns up to 1,024 are calculated directly using formulas in Shor's original analysis with specific inputs and are displayed in a blue interval called the empirical range per input size. It is observed that 1) certified bounds hold for all instances and 2) empirical bounds are considerably better than certified ones for studied instances. The latter is likely due to the nonoptimality of our proofs in Coq and the fact that we investigated only small-size instances. *SI Appendix*, section 4 for details.

3. Related Work

Researchers have been actively attempting to verify large-scale quantum algorithms recently. Boender et al. (32) verified a quantum teleportation protocol over a single qubit, through the use of matrix multiplication. Verification work on the QWIRE quantum circuit language (18) produced proofs of the correctness of a simple coin toss (33) and later quantum teleportation, and Deutsch's algorithm (34), but still no proofs of scalable quantum algorithms. Bordg et al. (35) proved the correctness of the Deutsch–Jozsa algorithm over an arbitrary number of qubits, and Liu et al. (19) formalized the quantum Hoare logic (36) for reasoning about quantum programs, and verified Grover's algorithm, both in the Isabelle/HOL proof assistant. Chareton et al. (21) and Hietala et al. (15) also produced correctness proofs for both Grover's algorithm and quantum phase estimation, using the QBricks block language built in Why3 and SQIR circuit language built in Coq, respectively. Chareton et al. were able to extend their result to cover the quantum part of the order-finding component in Shor's factorization algorithm. However, the certified implementation of the classical part of order finding and the remainder of Shor's algorithm was not pursued. Moreover, QBricks's use of Why3 requires a larger trusted computing base than Coq because Why3 relies on SMT solvers in verifying proofs (37).

An important limitation of prior work is that it focuses only on quantum (and generally unitary) circuits. Sophisticated quantum algorithms like Shor's algorithm feature interplay between classical and quantum subroutines, and both must be reasoned about together to prove correctness. This requires a generalized framework capable of reasoning about the hybrid probabilistic behavior of both quantum and classical components (e.g., classical-quantum interaction and repeated runs), which is usually more challenging and has been neglected by prior work, whereas our work fills the gap.

Moreover, the arithmetic oracle realizing modular exponentiation is the bulkiest subroutine of Shor's algorithm's implementation, consisting of more than 95% gates. They are typically programmed as classical reversible circuits and transformed into quantum circuits via tools like Scaffold's C2QG module (38). We implement RCIR, the language for reversible circuits with certified transformation to quantum circuits, to help both with programming such oracles and the correctness of their implementation. Finally, none of the existing work allows the extraction of executable instructions on quantum devices (or simulators). Our end-to-end certified implementation completes this missing piece, which, among other benefits, allows an empirical examination of the potential loopholes in the specification.

4. Conclusions

The nature of quantum computing makes programming, testing, and debugging quantum programs difficult, and this difficulty is exacerbated by the error-prone nature of quantum hardware. As a long-term remedy to this problem, we develop a framework based on formal methods to mathematically certify that quantum programs do what they are meant to, which we believe is a principled approach to mitigating human errors in this critical domain and achieving high assurance for the implementation of important quantum applications.

Data, Materials, and Software Availability. Codes data have been deposited in https://github.com/inQWIRE/SQIR/tree/main/examples/shor (39).

ACKNOWLEDGMENTS. We thank Andrew Childs, Steven Girvin, Liang Jiang, and Peter Shor for helpful feedback on the manuscript. This material is based upon work supported by the Air Force Office of Scientific Research under award number FA95502110051, the US Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Quantum Testbed Pathfinder Program under Award Number DE-SC0019040, and the US National Science Foundation grant CCF-1942837 (CAREER). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of these agencies.

Author affiliations: [a]Department of Computer Science, University of Maryland, College Park, MD 20740; [b]Joint Center for Quantum Information and Computer Science, University of Maryland, College Park, MD 20740; [c]Department of Computer Science, Columbia University, New York, NY 10027; and [d]Department of Computer Science, University of Chicago, Chicago, IL 60637

1. E. T. Campbell, B. M. Terhal, C. Vuillot, Roads towards fault-tolerant universal quantum computation. *Nature* **549**, 172–179 (2017).
2. B. M. Terhal, Quantum error correction for quantum memories. *Rev. Mod. Phys.* **87**, 307–346 (2015).
3. H. B. Curry, Functionality in combinatory logic. *Proc. Natl. Acad. Sci. U.S.A.* **20**, 584 (1934).
4. W. A. Howard, "The formulæ-as-types notion of construction" in *The Curry-Howard Isomorphism*, P. D. Groote, Ed. (Academia, 1995).
5. T. Hoare, The verifying compiler: A grand challenge for computing research. *J. ACM* **50**, 63–69 (2003).
6. R. A. De Millo, R. J. Lipton, A. J. Perlis, Social processes and proofs of theorems and programs. *Commun. ACM* **22**, 271–280 (1979).
7. G. Klein *et al*., "SeL4: Formal verification of an OS kernel" in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP '09)* (Association for Computing Machinery, New York, NY, 2009), pp. 207–220.
8. X. Leroy, Formal verification of a realistic compiler. *Commun. ACM* **52**, 107–115 (2009).
9. X. Yang, Y. Chen, E. Eide, J. Regehr, "Finding and understanding bugs in C compilers" in *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '11)* (Association for Computing Machinery, New York, NY, 2011), pp. 283–294.
10. G. Gonthier *et al*., Formal proof-the four-color theorem. *Not. AMS* **55**, 1382–1393 (2008).
11. D. Castelvecchi, Mathematicians welcome computer-assisted proof in 'grand unification' theory. *Nature* **595**, 18–19 (2021).
12. K. Hartnett, Proof assistant makes jump to big-league math (2021). https://www.quantamagazine.org/lean-computer-program-confirms-peter-scholze-proof-20210728/. Accessed 29 April 2022.
13. T Coq Development Team, *The Coq Proof Assistant Reference Manual, Version 8.16* (Zenodo, 2022).
14. K. Hietala, R. Rand, S. H. Hung, X. Wu, M. Hicks, A verified optimizer for quantum circuits. *Proc. ACM Program. Lang.* **5**, 1–29 (2021).
15. K. Hietala, R. Rand, S. H. Hung, L. Li, M. Hicks, "Proving quantum programs correct" in *Proceedings of the Conference on Interative Theorem Proving (ITP)* (2021).
16. L. K. Grover, "A fast quantum mechanical algorithm for database search" in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing (STOC '96)* (Association for Computing Machinery, New York, NY, 1996), pp. 212–219.
17. P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.* **26**, 1484–1509 (1997).
18. J. Paykin, R. Rand, S. Zdancewic, "QWIRE: A core language for quantum circuits" in *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017).* (Association for Computing Machinery, New York, NY, 2017), pp. 846–858.
19. J. Liu *et al*., "Formal verification of quantum algorithms using quantum Hoare logic" in *Computer Aided Verification*, I. Dillig, S. Tasiran, Eds. (Springer International Publishing, Cham, 2019), pp. 187–207.
20. D. Unruh, Quantum relational Hoare logic. *Proc. ACM Program. Lang.* **3**, 1–33 (2019).
21. C. Chareton, S. Bardin, F. Bobot, V. Perrelle, B. Valiron, "An automated deductive verification framework for circuit-building quantum programs," in *Programming Languages and Systems: 30th European Symposium on Programming (ESOP 2021), Held Part European Joint Conferences on Theory and Practice of Software (ETAPS 2021)* (Luxembourg City, Luxembourg, March 27–April 1, 2021), vol. 12648, pp. 148–177.
22. L. Paninski, A coincidence-based test for uniformity given very sparsely sampled discrete data. *IEEE Trans. Inf. Theory* **54**, 4750–4755 (2008).
23. S. Lloyd, Universal quantum simulators. *Science* **273**, 1073–1078 (1996).
24. A. L. Ruiz, E. C. Morales, L. P. Roure, A. G. Ríos, *Algebraic Circuits* (Springer, 2014).
25. D. De Rauglaudre, Coq proof of the Euler product formula for the Riemann zeta function (2020). https://github.com/roglo/coq_euler_prod_form. Accessed 13 December 2020.
26. G. H. Hardy, E. M. Wright, *An Introduction to the Theory of Numbers* (Oxford, ed. 4, 1975).
27. A. Bauer, Five stages of accepting constructive mathematics. *Bull. Am. Math. Soc.* **54**, 481–498 (2016).
28. Program Extraction (2021). https://coq.inria.fr/refman/addendum/extraction.html. Accessed 24 September 2021.
29. JKQ, DDSIM–A quantum circuit simulator based on decision diagrams written in C++ (2021). https://github.com/iic-jku/ddsim. Accessed 24 February 2021.
30. L. M. Vandersypen *et al*., Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance. *Nature* **414**, 883–887 (2001).
31. E. Martin-Lopez *et al*., Experimental realization of Shor's quantum factoring algorithm using qubit recycling. *Nat. Photon.* **6**, 773–776 (2012).
32. J. Boender, F. Kammüller, R. Nagarajan, "Formalization of quantum protocols using Coq" in *Proceedings of the 12th International Workshop on Quantum Physics and Logic (QPL), Oxford, U.K., July 15–17, 2015, Electronic Proceedings in Theoretical Computer Science*, C. Heunen, P. Selinger, J. Vicary, Eds. (Open Publishing Association, Waterloo, NSW, Australia, 2015), vol. 195, pp. 71–83.
33. R. Rand, J. Paykin, S. Zdancewic, QWIRE practice: Formal verification of quantum circuits in coq in *Proceedings of the 14th International Conference on Quantum Physics and Logic (QPL), Nijmegen, the Netherlands, July 3–7, 2017, Electronic Proceedings in Theoretical Computer Science*, B. Coecke, A. Kissinger, Eds. (Open Publishing Association, Waterloo, NSW, Australia, 2018), vol. 266, pp. 119–132.
34. R. Rand, "Publicly accessible Penn dissertations, 3175," PhD thesis, University of Pennsylvania, Philadelphia, PA (2018).
35. A. Bordg, H. Lachnitt, Y. He, Certified quantum computation in Isabelle/HOL. *J. Autom. Reason.* **65**, 691–709 (2020).
36. M. Ying, Floyd–Hoare logic for quantum programs. *ACM Trans. Program. Lang. Syst.* **33**, 19 (2012).
37. J. C. Filliâtre, A. Paskevich, "Why3–Where programs meet provers" in *Programming Languages and Systems: 22nd European Symposium on Programming (ESOP 2013), Held as Part of the European Joint Conferences on Theory and Practice of Software (ETAPS 2013) (Proceedings 22, Rome, Italy, March 16–24, 2013)* (Springer, 2013), pp. 125–128.
38. A. J. Abhari *et al*., "Scaffold: Quantum programming language" (Technical Report, Department of Computer Science, Princeton University, Princeton, NJ, 2012).
39. Y. Peng *et al*., SQIR/examples/shor, Code to a SQIR formalization of Shor's factoring algorithm. *Github*. https://github.com/inQWIRE/SQIR/tree/main/examples/shor. Deposited 4 May 2023.