QuantumLib: A Library for Quantum Computing in Coq

Jacob Zweifler[1], Kesha Hietala[2], and Robert Rand[1]

[1] University of Chicago, U.S.A.
[2] University of Maryland, U.S.A.

Overview and Motivation We present the INQWIRE QuantumLib,[1] a library for representing and reasoning about quantum computing in the Coq proof assistant. Originally developed to provide a semantics to the QWIRE programming language [4], QuantumLib now provides the mathematical foundation for a range of quantum verification projects, including QWIRE, SQIR/VOQC,[2] Stabilizer Types,[3] and VyZX.[4] QuantumLib emphasizes both computation and proof (unlike libraries such as Mathematical Components or CoqEAL that focus on one or the other) and is specialized to the use-case of verified quantum computing in Coq. Using the foundations of QuantumLib, both Shor's algorithm and Grover's algorithm have been formalized[2].

Linear Algebra over $\mathbb{C}$ Quantum computing operates on finite Hilbert spaces, which we can represent using vectors of complex numbers. QuantumLib builds upon Coquelicot's complex numbers [1], adding lemmas about Euler's identity and other numerical facts pertinent to quantum computing. Additionally, we define polynomials over $\mathbb{C}$, a notion of limits and continuous functions, and open/closed subsets of the complex numbers. We then prove some topological facts including the Heine-Borel theorem and the fundamental theorem of algebra. We also define the group of polar coordinates (r, θ) and show that $(r, \theta) \to re^{i\theta}$ produces a group isomorphic between polar coordinates and nonzero complex numbers, thereby allowing us to take *nth* roots of complex numbers.

With the basics established, we build up a comprehensive toolbox of objects and concepts from linear algebra. Matrices are defined as the type `Matrix (n m : nat) := nat -> nat -> C`. Note that this means a matrix is technically an infinite 2D array. We include the bounds `n` and `m` in matrix types as *phantom types* [5] and provide a well-formedness predicate (`WF_Matrix`) that ensures a matrix is only nonzero within its specified bounds. While this representation allows us to write nonsensical matrices and operations that don't respect well-formedness, it makes splicing and tensoring matrices much easier. This design choice gives users an alternative to other matrix libraries that rely on dependent types or list representations for matrices, which are more awkward when dealing with Kronecker product. This formulation of matrices comes with the following support:

- Usual matrix operations such as addition, multiplication, scalar products, transposition, and Kronecker product, as well proofs of algebraic facts about these operations such as associativity, commutativity, and distributivity.
- Row and column operations, a few types of matrix splicing, and associated proofs that row/column operations correspond to multiplication by elementary matrices.
- Matrix properties such as linear independence, invertibility, orthogonality, and diagonalizability. We also define the norm of a vector and prove that a full rank set of orthonormal vectors gives a unitary matrix and vice versa.
- Determinants, basis vectors, and the Gram-Schmidt algorithm.
- Proofs that being invertible, being linearly independent, and having nonzero determinant are all equivalent. We also show that right invertibility of a square matrix implies that a left inverse exists and that it is equal to the right inverse.

[1] https://github.com/inQWIRE/QuantumLib
[2] https://github.com/inQWIRE/SQIR
[3] https://github.com/inQWIRE/Heisenberg-Foundations
[4] https://github.com/inQWIRE/VyZX

- Notions of eigenvectors and eigenvalues and various lemmas, including a proof that all square matrices over $\mathbb{C}$ have at least one nonzero eigenvector. We also show that diagonalizable matrices with the same eigenvector/eigenvalue pairs are equal.

Quantum Operations Using the machinery described in the previous section, we develop constructs useful for describing and reasoning about quantum programs:

- Dirac notation for quantum states, allowing one to define an n-qubit quantum state, represented as a 2^n-length vector over $\mathbb{C}$. We also define a variety of standard 2-by-2 unitary matrices. More generally, we include a translation function from IBM Qiskit's universal U_3 gate to our matrices, allowing us to represent any operator on a single qubit by specifying angles θ, ψ, and λ.
- Padding functions that apply a single qubit unitary U to the ith bit of a state vector. Furthermore, we define a `control (U : Unitary n)` whose action is to apply a control-U operation to $n+1$ specified qubits. We also prove that applying the 2-by-2 unitary U to bit i is the same as multiplying the corresponding vector by $I^{\otimes i-1} \otimes U \otimes I^{\otimes k}$ and we prove similar lemmas for multi-qubit unitaries.
- A definition of a permutation σ over $[0, \ldots, 2^n - 1]$, and various lemmas about permutations. We also define a function turning a permutation into a corresponding swap matrix where 1's are located at entries $(i, \sigma(i))$.
- A separate notion of a state vector allowing for updates and shifts, which originally came from SQIR [2, §4.6]. We prove that this notion corresponds to standard vectors over $\mathbb{C}$.
- A notion of measurement and partial measurement. Specifically, we created functions that compute the probability of certain events occurring given some input vector state.
- Support for density matrices, including predicates that describe both pure and mixed states. We prove that unitaries preserve pure states and that measurement preserves mixed states. We also define the notion of a superoperator acting on these density matrices.
- A representation of discrete probability distributions along with a function to map a quantum state to a probability distribution.

Automation We create a few different matrix tactics to check for equality and well-formedness such as `lma`, `solve_matrix`, and `gridify`. Tactics `lma` and `solve_matrix` prove equality between matrix expressions. Both rely heavily on `lca`, which applies Coq's `lra` solver to the imaginary and real parts of a complex number equality. The `gridify` tactic rewrites symbolic matrices into a normal form so the solver can more easily check equality [3, §4.5]. We also develop automation and rewrite databases to check the well-formedness and unitarity of matrices, and to simplify matrix expressions. All of these tactics drastically simplify the task of checking the equivalence of quantum operations.

References

[1] Sylvie Boldo, Catherine Lelay, and Guillaume Melquiond. Coquelicot: A user-friendly library of real analysis for Coq. *Mathematics in Computer Science*, 9(1):41–62, 2015.

[2] Kesha Hietala, Robert Rand, Shih-Han Hung, Liyi Li, and Michael Hicks. Proving Quantum Programs Correct. *12th International Conference on Interactive Theorem Proving (ITP 2021)*, 193(21):1–19, 2021.

[3] Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. A verified optimizer for quantum circuits. In *Proceedings of the ACM on Programming Languages*, volume 5, pages 1–29, 2021.

[4] Robert Rand, Jennifer Paykin, and Steve Zdancewic. QWIRE practice: Formal verification of quantum circuits in Coq. In *Proceedings of the 14th International Conference on Quantum Physics and Logic, QPL*, 2017.

[5] Robert Rand, Jennifer Paykin, and Steve Zdancewic. Phantom types for quantum programs. In *The Fourth International Workshop on Coq for Programming Languages*, 2018.