Formal Verification of Quantum Stabilizer Code

Qiuyi Feng
University of Melbourne
Melbourne, Australia
qiuyif@student.unimelb.edu.au

Udaya Parampalli
University of Melbourne
Melbourne, Australia
udaya@unimelb.edu.au

Christine Rizkallah
University of Melbourne
Melbourne, Australia
christine.rizkallah@unimelb.edu.au

1 Motivation

The Quantum Error Correction Code (QECC) in quantum computing adds redundant bits to a quantum state to enable the detection and correction of certain types of correctable errors [3]. From an implementation perspective, it can be viewed as an *encoding* circuit that takes a logical state as input, complementing it with the additional required quantum bits (qubits), and mapping these qubits to a higher-dimensional state, called *codewords*, where certain errors are detected and corrected. Figure 1 provides such an example.

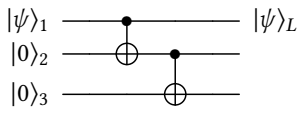

Figure 1. Encoding circuit for the 3-qubit code. A quantum state $|\psi\rangle$ is entangled with two redundancy qubits $|0\rangle$ to create an encoded state $|\psi\rangle_L$. This QECC allows correcting a single bit-flip error during the runtime.

QECCs are particularly crucial in achieving fault-tolerant quantum computing, therefore, it is beneficial to provide a correctness guarantee for them. The most important property of a QECC is the set of *correctable errors* and the *distance* of codewords, indicating how many errors, the QECC can detect and correct.

One way of verifying such codes is by interpreting them as a linear transformation in a 2^n dimensional complex vector space, where n is the number of qubits involved in the QECC. For example, the program in Figure 1 can be viewed as a function between two 8-dimensional quantum states (2^3). For each combination of correctable errors (by both the type and position of errors), one needs to prove that the encoder circuit can detect and correct the error by essentially performing matrix multiplication. This method suffers from a few shortcomings. First, the dimension of the vector space grows exponentially as the number of qubits n grows, making the verification of large-scale QECCs by brute-force problematic. Second, it relies on an enumeration of each combination of errors, which requires a lot of manual proof effort.

Then pen-and-paper reasoning about QECCs sometimes takes another approach – by the *stabilizer formalism*, in which a QECC is associated with an algebraic group structure called a *stabilizer group*) [2]. In this approach, operations on groups could substitute matrix multiplication in reasoning about correctable errors. This provides more general and more compositional pr. It can also avoid enumerating every possible combination of errors through a combinatorial brute-force proof approach. A QECC that can be described by a stabilizer group is called a *stabilizer code*.

Inspired by this pen-and-paper approach, we hope to verify QECCs using the stabilizer formalism. Our primary goal is to investigate whether the stabilizer formalism can reduce the efforts in verifying QECC programs, and provide generalizability to verify generalized stabilizer code. We can then easily provide several verified concrete QECC programs by instantiating as instantiations of our formalism. The algebraic structures are expected to be reusable for the verification of other quantum computing protocols and algorithms, such as the quantum key distribution protocol.

2 Preliminary Plan

Overview of Research Plan. We plan to investigate the formal verification of the stabilizer code. This goal implies two objectives. First, we plan to formalize the stabilizer group and its underlying mathematical structures. Second, we plan to use our stabilizer groups formalism to verify instances of stabilizer code encoders.

Selection of Tools. We plan to formalize the stabilizer group and verify the stabilizer code in Coq. The Coq proof assistant has the expressive power to formally specify quantum computing processes. This is demonstrated by existing work on certified quantum programs (see Section 3).

Among existing tools for verifying quantum programs, we base our work on a language called SQIR (Small Quantum Intermediate Representation) [5] for two reasons. First, SQIR has a dedicated mathematical library for quantum computing, called `quantumlib` [5], that provides a formalism for complex vector spaces. Second, it also has an intermediate formal language for quantum programs that can provides the right level of abstraction (quantum circuits) for specifying and verifying concrete stabilizer codes. We provide an overview of other Coq quantum verification formalisms in Section 3.

Formalism of the Stabilizer Group. We hope to formalize the algebraic structures concerns with the stabilizer group. Mathematically, the stabilizer group is a subgroup of the Pauli group. Therefore, the Pauli group is the essential component for this goal. Specially, we hope to formally verify these mathematical structures and their properties:

- The construction of the Pauli group. Followed by its definition, an element of the n-qubit Pauli group is a combination of a scalar from $\{+1, -1, i, -i\}$, and an n-qubit Pauli string, which is produced by a tensor product of n Pauli matrices $\{I, X, Y, Z\}$. For example, $+XXYY$ is a valid element of the 4-qubit Pauli group.
- The correctness of operations on Pauli group. There is a direct interpretation that translates a n-qubit group element into a 2^n matrix, which is defined by the mathematical meaning of tensor product. Therefore, we hope to verify the operations on the group respects with the matrix multiplication. To illustrate, we define $X \cdot Y = iZ$ and verify the matrix interpretation is indeed correct. By doing so, we can discard the need to do matrix multiplication next time we see $X \cdot Y$.
- The properties of *stabilizer*, in which the stabilize relation is the most important. Informally, an operator S stabilizes a quantum state $|\psi\rangle$ if $S|\psi\rangle = |\psi\rangle$. That is, the operator doesn't change the quantum state. In other words, the state is in the operator's +1 eigenspace. By definition, a stabilizer group is essentially a Pauli group in which all of the elements stabilize some states. Therefore, the stabilizing relation lies at the core of this work. Other relations might also be helpful in our work, like the commute and anti-commute relation between elements of the Pauli group.

We have formalized the construction of the 1-qubit Pauli group and verified the correctness of the operations defined on the group. We are working on further extending it to the generalized n-qubit Pauli group. The up-to-date development of this project can be found in our online repository [1].

Verification of Stabilizer Code. We hypothesize that by using the formalism of the stabilizer group, the efforts of verification can be reduced. To test this hypothesis, we hope to conduct the verification of the stabilizer code in the following way:

- Verify a small-scale stabilizer code encoding program directly using SQIR.
- Verify the same stabilizer code encoding program using the stabilizer group formalism. Compare it with the previous verification process to identify what has been improved.
- Verify a relatively larger scale stabilizer code encoding program using the stabilizer group formalism. This is intended to be a demonstration that our work can scale to larger programs.
- Verify the generalized stabilizer encoding program using the stabilizer group formalism [3]. This is intended to evaluate the generalizability of our stabilizer formalism.

[1] https://github.com/ExcitedSpider/SQIR/tree/main/examples/stabilizer

Currently, we have verified a 4-qubit stabilizer code by the matrix semantics. We hope to revise it with the stabilizer group once we have completed the formalism.

3 Related Work

Formal Verification of Quantum Programs in Coq. Apart from SQIR, there are some other works in the verification of quantum programs using Coq. One of the first quantum programming languages with verifiable programs is QWIRE [7], and QWIRE and SQIR share overlapping authorship. QWIRE exploits the expressiveness of Coq to implement a linear type system to enforce constraints of quantum computing, such as the no-cloning theorem. In addition, QWIRE is published with a dedicated mathematical library, which is reused in SQIR later. The difficulty introduced by the expressiveness of QWIRE in verification led to the later design of SQIR [5], which has had various improvements over it.

Recently, a recent work CoqQ represents attempts to verify quantum higher-level algorithms [10]. As its name suggests, it is implemented in Coq. Unlike SQIR, which is more like a domain-specific language for quantum programs. CoqQ allows quantum programs to be defined in a way that resembles classical programming languages like C. That means programmers don't need to separate an algorithm's quantum computing and classical parts. CoqQ adapted the *Quantum Hoare logic* [9] as inference rules and `mathcomp` [6] as its underlying mathematical library. CoqQ has been used in verifying many novel quantum algorithms, such as algorithms for solving linear systems of equations [4].

Verification of QECCs. In 2021, a framework QECV has been proposed to build and verify QEC programs [8]. However, QECV is based on the symbolic execution of quantum programs, which differs from our intended work.

Chancellor et. al. employed a high-level graphical language (the ZX-Calculus) to verify QECCs [1]. However, this work is mainly pen-and-paper without machine checks. It does not include the stabilizer formalism.

In the recent development of SQIR in its online repository, a 3-qubit QECC, and a 9-qubit QECC are formalized and verified by matrix semantics, without introducing the concept of stabilizer code. While it differs from our intended work, this formalization greatly inspires our study.

4 Acknowledgement

We would like to thank Robert Rand for providing us with valuable feedback on research directions. We use SQIR as a basis for our work and are inspired to undertake this project by the QECC examples mentioned above.

References

[1] Chancellor, N., Kissinger, A., Zohren, S., Roffe, J., and Horsman, D. Graphical structures for design and verification of quantum error correction. *Quantum Science and Technology 8*, 4 (2023), 045028.

[2] Dymarsky, A., and Shapere, A. Quantum stabilizer codes, lattices, and cfts. *Journal of High Energy Physics 2021*, 3 (2021), 1–84.

[3] Gottesman, D. *Stabilizer codes and quantum error correction*. California Institute of Technology, 1997.

[4] Harrow, A. W., Hassidim, A., and Lloyd, S. Quantum algorithm for linear systems of equations. *Physical review letters 103*, 15 (2009), 150502.

[5] Hietala, K., Rand, R., Hung, S.-H., Wu, X., and Hicks, M. A verified optimizer for quantum circuits. *Proceedings of the ACM on Programming Languages 5*, POPL (2021), 1–29.

[6] Mahboubi, A., and Tassi, E. Mathematical components. *Online book* (2021).

[7] Paykin, J., Rand, R., and Zdancewic, S. Qwire: a core language for quantum circuits. *ACM SIGPLAN Notices 52*, 1 (2017), 846–858.

[8] Wu, A., Li, G., Zhang, H., Guerreschi, G. G., Xie, Y., and Ding, Y. Qecv: Quantum error correction verification. *arXiv preprint arXiv:2111.13728* (2021).

[9] Ying, M. Floyd–hoare logic for quantum programs. *ACM Transactions on Programming Languages and Systems (TOPLAS) 33*, 6 (2012), 1–49.

[10] Zhou, L., Barthe, G., Strub, P.-Y., Liu, J., and Ying, M. Coqq: Foundational verification of quantum programs. *Proceedings of the ACM on Programming Languages 7*, POPL (2023), 833–865.