Symbolic Execution for Quantum Error Correction Programs

WANG FANG, Institute of Software at Chinese Academy of Sciences, China and University of Chinese Academy of Sciences, China
MINGSHENG YING, Institute of Software at Chinese Academy of Sciences, China and Tsinghua University, China

We define QSE, a symbolic execution framework for quantum programs by integrating symbolic variables into quantum states and the outcomes of quantum measurements. The soundness of QSE is established through a theorem that ensures the correctness of symbolic execution within operational semantics. We further introduce symbolic stabilizer states, which symbolize the phases of stabilizer generators, for the efficient analysis of quantum error correction (QEC) programs. Within the QSE framework, we can use symbolic expressions to characterize the possible discrete Pauli errors in QEC, providing a significant improvement over existing methods that rely on sampling with simulators. We implement QSE with the support of symbolic stabilizer states in a prototype tool named QuantumSE.jl. Our experiments on representative QEC codes, including quantum repetition codes, Kitaev's toric codes, and quantum Tanner codes, demonstrate the efficiency of QuantumSE.jl for debugging QEC programs with over 1000 qubits. In addition, by substituting concrete values in symbolic expressions of measurement results, QuantumSE.jl is also equipped with a sampling feature for stabilizer circuits. Despite a longer initialization time than the state-of-the-art stabilizer simulator, Google's Stim, QuantumSE.jl offers a quicker sampling rate in the experiments.

CCS Concepts: • **Software and its engineering** → **Automated static analysis**; • **Theory of computation** → **Automated reasoning**; • **Hardware** → *Quantum error correction and fault tolerance*; • **Computer systems organization** → **Quantum computing**.

Additional Key Words and Phrases: symbolic execution, stabilizer formalism

ACM Reference Format:
Wang Fang and Mingsheng Ying. 2024. Symbolic Execution for Quantum Error Correction Programs. *Proc. ACM Program. Lang.* 8, PLDI, Article 189 (June 2024), 26 pages. https://doi.org/10.1145/3656419

1 INTRODUCTION

Nowadays, quantum computing hardware is developing rapidly [Arute et al. 2019; Bluvstein et al. 2024; Madsen et al. 2022; Wu et al. 2021a], and the number of its physical qubits is also gradually increasing. For instance, IBM has introduced the IBM Condor, an advanced quantum processor featuring 1,121 superconducting qubits [IBM 2023]. However, these quantum hardware suffer from errors caused by quantum noise and inaccurate quantum gate implementations, so deploying quantum error correction (QEC) programs on quantum hardware is the key to large-scale quantum computing in the future [Aharonov and Ben-Or 1997; Gottesman 2014; Shor 1996]. There are already several experimental studies exploring QEC in quantum hardware [Abobeih et al. 2022; Acharya et al. 2023; Bluvstein et al. 2024; Chen et al. 2021; Zhao et al. 2022], among which the recent result of Bluvstein et al. [2024] has successfully encoded 48 logical qubits on neutral atom arrays with

Authors' Contact Information: Wang Fang, Institute of Software at Chinese Academy of Sciences, Beijing, China and University of Chinese Academy of Sciences, Beijing, China, fangw@ios.ac.cn; Mingsheng Ying, Institute of Software at Chinese Academy of Sciences, Beijing, China and Tsinghua University, Beijing, China, yingms@ios.ac.cn.

ACM 2475-1421/2024/6-ART189
https://doi.org/10.1145/3656419

up to 280 physical qubits. On the other hand, as diverse and more complicated QEC protocols are introduced, we will need to conduct prior analysis and verification of them to guarantee their correctness before deployment. This leads us to the following basic question:

How can we check whether a QEC program (e.g., the one depicted in Fig. 1), as a hybrid quantum-classical program, has any bugs, especially the part involving quantum variables; and more importantly, if the QEC program has bugs, how can we automatically find them?

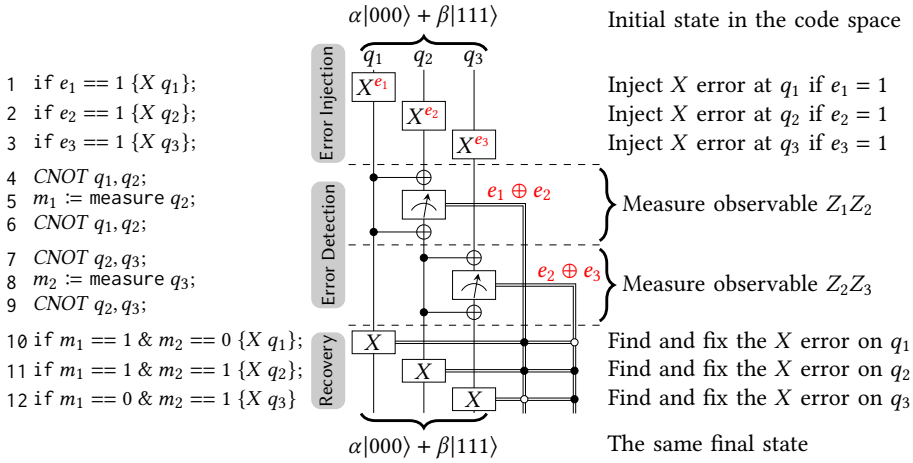


Fig. 1. A QEC program (Lines 4-12) for the three-qubit bit-flip code with code distance 3. Ensuring the program is bug-free requires that for any initial state $\alpha|000\rangle + \beta|111\rangle$ within the code space, the final state remains $\alpha|000\rangle + \beta|111\rangle$ after error injection (Lines 1-3) and execution of the QEC program (Lines 4-12). Error injection injects errors that the QEC code can tolerate into the quantum state, and then the QEC program corrects these errors. Here, the possible errors correspond to variables $e_1, e_2, e_3 \in \{0, 1\}$ that satisfy $e_1 + e_2 + e_3 \leq 1$ (up to one X error occurs). We also refer to the errors $X^{e_1}, X^{e_2}, X^{e_3}$ introduced during error injection as "adversarial" because of their potential to disrupt the integrity of information in quantum states.

Fortunately, current QEC programs only involve stabilizer circuits and thus can be efficiently simulated on classical computers [Gottesman 1997]. Indeed, several specific stabilizer circuit simulators have been developed [Aaronson and Gottesman 2004; Anders and Briegel 2006; Gidney 2021; Krastanov 2019] for analyzing QEC circuits. For instance, Google's recent QEC experiment [Acharya et al. 2023] utilized a fast simulator, Stim [Gidney 2021], to sample a prior distribution for detectors of the experimental QEC circuit. For the example program in Fig. 1, the current practice involves enumerating/sampling the potential values of $e_i \in \{0, 1\}$ such that $e_1 + e_2 + e_3 \leq 1$, then executing the program with them by using a fast simulator. However, for a general QEC code with n qubits and a code distance of d, the potential errors during error injection correspond to variables $e_1, e_2, \ldots, e_n \in \{0, 1\}$ with the condition $e_1 + e_2 + \cdots e_n \leq \lfloor \frac{d}{2} \rfloor$. The count of these errors exceeds $\binom{n}{\lfloor \frac{d}{2} \rfloor}$, which goes beyond polynomial-time complexity. Despite the simulator's rapid performance[1], such number of errors would make this approach unscalable at a large scale. Additionally, as noted by Gidney [2021], *generating samples of QEC circuits is the bottleneck in analyzing QEC programs.*

In classical computing, symbolic execution (SE) was proposed [King 1976] to handle such a large number of samples or test cases. It is an automated method for finding bugs in programs and has been

[1] e.g., the simulator Stim [Gidney 2021] can simulate a distance 100 surface code circuit (20 thousand qubits, 8 million gates, 1 million measurements) in 15 seconds.

successfully applied in many different fields, including software testing [Cadar et al. 2008; Cadar and Sen 2013; Godefroid et al. 2005; Poeplau and Francillon 2020] and security and privacy [Aizatulin et al. 2011; Farina et al. 2019], and extended to non-deterministic programs [Luckow et al. 2014; Siegel et al. 2008; Yu et al. 2020] and probabilistic programs [Gehr et al. 2016; Susag et al. 2022]. Recently, SE has also been introduced into quantum computing in several pioneering papers [Bauer-Marquart et al. 2023; Carette et al. 2023; Tao et al. 2022]. But all of them only deal with quantum circuits *without control flows*. As is well-known, one of the most essential advantages of SE is that it can systematically explore many possible execution paths (determined by control flows) at the same time. Therefore, the power of SE has not been fully exploited in quantum programming.

In this paper, we significantly extend SE to a quantum programming language with *(classical) control flows*, which is particularly suited to writing QEC programs. *Our SE framework introduces symbols to represent the possible (probabilistic) outcomes of quantum measurements in a uniform manner*, rather than splitting the measurement outcomes into cases as in previous work [Bauer-Marquart et al. 2023]. This approach circumvents the potential exponential growth of execution paths that comes with quantum measurements, which are a fundamental part of QEC programs. To make our SE framework efficient in handling QEC programs, we further introduce *symbolic stabilizer states* by a partial symbolization of stabilizer states, the underlying quantum states in stabilizer circuits. Thanks to the fast simulation algorithm of stabilizer circuits [Aaronson and Gottesman 2004], our symbolic stabilizer states can be efficiently manipulated during SE.

More importantly, symbolic stabilizer states allow us to handle conditionally applied Pauli gates (see Lines 1-3 and 10-12 of Fig. 1) without forking into multiple execution paths: the conditional guards, e.g., $e_1 == 1$ and $e_2 == 1$, are absorbed into the symbolic stabilizer state, yielding only a single execution path and transforming subsequent measurement outcomes into symbolic expressions, e.g., $e_1 \oplus e_2$ for variable m_1 in Line 5 of Fig. 1. Consequently, we can use symbolic expressions, e.g., $e_1 + e_2 + e_3 \leq 1$, to characterize the possible errors in error injection and use SMT solver to deduce the correctness, obviating the need for enumerating concrete values that satisfy the condition.

Contributions and outline. After reviewing some background knowledge (§2) and demonstrating our approach on a running example (§3), our major contributions are presented as follows:

- We develop *a framework for symbolic execution of quantum programs* (QSE) with classical control flows (§4) and prove a *soundness theorem* for our QSE framework (§5).
- We introduce symbolic stabilizer states to enable our QSE framework to efficiently analyze QEC programs and prove an *adequacy theorem* for symbolic stabilizer states in the analysis and verification of QEC programs (§6).
- We introduce *symbolic Pauli gates* with a newly designed SE rule that can handle the conditional application of Pauli gates without forking like classical SE and demonstrate how it can be used to characterize the possible adversarial errors for QEC programs (§7).
- We implement our QSE framework together with the special support of symbolic stabilizer states in a *prototype tool* named QuantumSE.jl and demonstrate its efficiency and the ability to outperform existing tools in experimental evaluation (§8).

The paper is concluded by discussions about related work (§9) and issues for further research (§10).

2 BACKGROUND

In this section, we provide a minimal background of quantum computation and QEC. The reader can consult the book [Nielsen and Chuang 2010, Chapter 2,4,10] for more details.

2.1 Quantum Preliminary

We assume basic knowledge of linear algebra, including the concepts of vector space and tensor product. For a d-dimensional complex vector space $\mathbb{C}^d$, we use the Dirac notation $|\psi\rangle$ to denote a column vector in it. The conjugate transpose of $|\psi\rangle$ is then a row vector denoted by $\langle\psi|$. The *inner product* of $|\psi\rangle$ and $|\phi\rangle$ is a complex number denoted by $\langle\phi|\psi\rangle$. The *outer product* of $|\psi\rangle$ and $|\phi\rangle$ is a $d \times d$ matrix denoted by $|\psi\rangle\langle\phi| \in \mathbb{C}^{d\times d}$. The norm of a vector $|\psi\rangle$ is defined as $\||\psi\rangle\| = \sqrt{\langle\psi|\psi\rangle}$. In addition, the tensor product of $|\psi_1\rangle$ and $|\psi_2\rangle$ is denoted by $|\psi_1\rangle \otimes |\psi_2\rangle$, which is sometimes written as $|\psi_1\rangle|\psi_2\rangle$ or even $|\psi_1\psi_2\rangle$ for short.

Quantum states. The state space of a quantum bit (qubit) is the 2-dimensional vector space $\mathbb{C}^2$ with $|0\rangle = \binom{1}{0}, |1\rangle = \binom{0}{1}$ being the *computational basis*. In general, the state space of n-qubit system is the tensor product of n copies of the state space $\mathbb{C}^2$ of a single qubit, which is $(\mathbb{C}^2)^{\otimes n} \cong \mathbb{C}^{2^n}$, with $\{|x\rangle | x \in \{0,1\}^n\}$ being the computational basis. A *pure* quantum state is represented by a unit vector $|\psi\rangle$. Thus, an n-qubit pure state $|\psi\rangle \in (\mathbb{C}^2)^{\otimes n}$ can be expressed as $\sum_{x\in\{0,1\}^n} \alpha_x |x\rangle$, where $\alpha_x \in \mathbb{C}$ and $\sum_{x\in\{0,1\}^n} |\alpha_x|^2 = 1$. When the state of an n-qubit system is not completely known, one may think of it as a *mixed* state (an ensemble of pure states) $\{(p_j, |\psi_j\rangle)\}$ meaning that it is in state $|\psi_j\rangle$ with probability p_j. Such a mixed state can also be represented by a *density operator* $\rho = \sum_j p_j |\psi_j\rangle\langle\psi_j|$, which is a $2^n \times 2^n$ positive semidefinite complex matrix. In particular, a pure state $|\psi\rangle$ can be represented by the density operator $|\psi\rangle\langle\psi|$; for simplicity, we write $\psi = |\psi\rangle\langle\psi|$.

Unitary transformations. A $2^n \times 2^n$ complex matrix U is called unitary if $U^\dagger U = \mathbb{1}_{2^n}$, where $U^\dagger$ stands for the conjugate transpose of U, and $\mathbb{1}_{2^n}$ denotes the $2^n \times 2^n$ identity matrix. It models a transformation or an evolution of an n-qubit system from a pure state $|\psi\rangle \in \mathbb{C}^{2^n}$ to $U|\psi\rangle$. For mixed states, it transforms a density matrix $\rho \in \mathbb{C}^{2^n\times 2^n}$ to $U\rho U^\dagger$. Such a unitary transformation U is often called an n-qubit gate. Common quantum gates include the single-qubit gates H (Hadamard gate), S (Phase gate), and $I = \mathbb{1}_2, X, Y, Z$ (Pauli gates) as well as the 2-qubit gate *CNOT* (controlled-NOT gate):

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix},\ S = \begin{pmatrix}1 & 0\\ 0 & i\end{pmatrix},\ X = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix},\ Y = \begin{pmatrix}0 & -i\\ i & 0\end{pmatrix},\ Z = \begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix},\ CNOT = \begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix}$$

The H gate can produce another important basis, called the $\pm$ basis, $\{|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\}$ of $\mathbb{C}^2$ from the computational basis as $H|0\rangle = |+\rangle$ and $H|1\rangle = |-\rangle$. The Phase gate S leaves $|0\rangle$ unchanged and adds a phase of i to $|1\rangle$, i.e., $S|0\rangle = |0\rangle$, $S|1\rangle = i|1\rangle$; similarly, the Pauli Z gate only adds phase -1 to $|1\rangle$, i.e., $Z|0\rangle = |0\rangle, Z|1\rangle = -|1\rangle$. The Pauli X gate acts like a "NOT" gate, exchanging $|0\rangle, |1\rangle$, i.e., $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$; similarly the Pauli Y gate switches $|0\rangle, |1\rangle$ and introduces additional phases of i and $-i$, i.e., $Y|0\rangle = i|1\rangle, Y|1\rangle = -i|0\rangle$. In terms of the computational basis, the *CNOT* gate, which can be rewritten as $CNOT = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes X$, performs a "NOT" gate X if the first qubit is set to $|1\rangle$, otherwise does nothing.

For an n-qubit system with label $q_j, 1 \le j \le n$ for each qubit, a k-qubit unitary $U \in \mathbb{C}^{2^k\times 2^k}$ applied to qubits $\bar{q} = q_{j_1}, q_{j_2}, \ldots, q_{j_k}$ is expanded to the unitary $U_{\bar{q}} \otimes \mathbb{1}_{\{q_j, 1\le j\le n\}\setminus\bar{q}} \in \mathbb{C}^{2^n\times 2^n}$ that performs a local unitary U on qubits $\bar{q}$ and does nothing with the remaining qubits. We often write $U_{\bar{q}}$ that omits the identity operator for $U_{\bar{q}} \otimes \mathbb{1}_{\{q_j, 1\le j\le n\}\setminus\bar{q}}$, if there is no ambiguity. For example, consider a 3-qubit system with label j for the j-th qubit, we write Z_1 for $Z \otimes I \otimes I$ and write X_1Y_3 for $X \otimes I \otimes Y$.

Quantum measurements and observables. The information about a quantum system has to be acquired by quantum *measurements*. A measurement on n-qubit system is described by a collection $M = \{M_m\}_m$ of $2^n \times 2^n$ complex matrices with the normalization condition $\sum_m M_m^\dagger M_m = \mathbb{1}_{2^n}$.

When performing it on a pure state $|\psi\rangle$ and a mixed state ρ, the measurement outcome of index m occurs with probabilities $p_m = \langle\psi|M_m^\dagger M_m|\psi\rangle$ and $p_m = \mathrm{tr}(M_m\rho M_m^\dagger)$, respectively, and the state after the measurement with outcome m collapses into $|\psi_m\rangle = M_m|\psi\rangle/\sqrt{p_m}$ and $\rho_m = M_m\rho M_m^\dagger/p_m$, respectively. For example, the *computational basis measurement* $\{|0\rangle\langle 0|, |1\rangle\langle 1|\}$ performed on the state $|+\rangle$ will result in a state $|0\rangle\ \langle 0|+\rangle\ /\sqrt{1/2} = |0\rangle$ with probability $\langle +|0\rangle\ \langle 0|+\rangle = |\langle 0|+\rangle|^2 = \frac{1}{2}$ and state $|1\rangle\ \langle 1|+\rangle\ /\sqrt{1/2} = |1\rangle$ with probability $\langle +|1\rangle\ \langle 1|+\rangle = |\langle 1|+\rangle|^2 = \frac{1}{2}$.

For an n-qubit system with label q_j, $1 \le j \le n$ for each qubit, the computational basis measurement on a qubit $q \in \{q_1, \ldots, q_n\}$ is given as

$$\{M_0 = |0\rangle_q\langle 0| \otimes \mathbb{1}_{\{q_j, 1\le j\le n\}\setminus q}, M_1 = |1\rangle_q\langle 1| \otimes \mathbb{1}_{\{q_j, 1\le j\le n\}\setminus q}\},$$

where we use the same notation as in unitary transformations and also write $|j\rangle_q\langle j|$ for $|j\rangle_q\langle j| \otimes \mathbb{1}_{\{q_j, 1\le j\le n\}\setminus q}$.

We say a linear operator O is an *observable* if $O^\dagger = O$. The spectral decomposition of an observable $O = \sum_m mP_m$ (a sum over its eigenvalues and corresponding projectors) corresponds to a quantum measurement $\{P_m\}_m$ with outcome m for each projector P_m. For example, Pauli gates X, Y, Z are all observables. In this paper, for those observables with eigenvalues ± 1, e.g., Z_1Z_2, we use Boolean values 0 and 1 to indicate the measurement outcomes $(-1)^0 = 1$ and $(-1)^1 = -1$, respectively, to make it convenient in presentations.

2.2 Stabilizer States and Quantum Error Correction

Pauli strings and Clifford gates. An n-qubit *Pauli string* P is defined as the tensor products of n Pauli gates with a phase in $\{\pm 1, \pm i\}$, i.e.,

$$P \triangleq i^k P_1 \otimes P_2 \otimes \cdots \otimes P_n,$$

where $k \in \{0, 1, 2, 3\}$, $P_\ell \in \{I, X, Y, Z\}$ for $1 \le \ell \le n$. The set of all n-qubit Pauli strings forms a group with matrix multiplication. A *Clifford gate* V is a unitary such that for any Pauli string P, $VPV^\dagger$ (conjugation by V) is still a Pauli string. In particular, each Pauli gate is a Clifford gate. Clifford gates have a nice structure; that is, any Clifford gate can be constructed from the three gates: H, S, and $CNOT$ [Gottesman 1997], e.g., $I = HH$, $X = HSSH$, $Y = SHSSHSSS$ and $Z = SS$.

Stabilizer and stabilizer states. A state $|\psi\rangle$ is *stabilized* by a gate U if $U|\psi\rangle = |\psi\rangle$, i.e., $|\psi\rangle$ is an eigenvector of U with eigenvalue 1. For example, $|+\rangle$ is stabilized by Pauli X gate and $|-\rangle$ is stabilized by $-X$ as $(-X)|-\rangle = |-\rangle$. For a list of n-qubit Pauli strings $P_1, P_2, \ldots, P_k \ne \pm I^{\otimes n}$, $1 \le k \le n$, that are commuting independent[2] and $P_j^2 \ne -I^{\otimes n}$ for $1 \le j \le k$, let $S = \langle P_1, P_2, \ldots, P_k\rangle$ be the group generated by $\{P_1, P_2, \ldots, P_k\}$. The subspace of states stabilized by all Pauli strings in S is denoted by V_S and S is said to be the *stabilizer* of V_S. We can check that $|\psi\rangle \in V_S$ if and only if $P_j|\psi\rangle = |\psi\rangle$ for any $1 \le j \le k$, thus $V_S = \cap_{j=1}^k V_{\langle P_j\rangle}$. Moreover, V_S is a non-trivial subspace with $\dim V_S = 2^{n-k}$ since each P_j halves the dimension of the space being stabilized.

In the case of $k = n$, which implies $\dim V_S = 1$, we can use the stabilizer $S = \langle P_1, P_2, \ldots, P_n\rangle$ to represent the only state $|\psi\rangle \in V_S$ with global phase ignored; we say S is the stabilizer of $|\psi\rangle$ and $|\psi\rangle$ is a *stabilizer state*. For example, we have $Z_1|000\rangle = Z_1Z_2|000\rangle = Z_2Z_3|000\rangle = |000\rangle$, so $\langle Z_1, Z_1Z_2, Z_2Z_3\rangle$ is the stabilizer of $|000\rangle$. Also, we have $(-Z_1)|111\rangle = Z_1Z_2|111\rangle = Z_2Z_3|111\rangle = |111\rangle$, and then $\langle -Z_1, Z_1Z_2, Z_2Z_3\rangle$ is the stabilizer of $|111\rangle$.

Evolution of stabilizer states by Clifford gates. Consider an n-qubit stabilizer state $|\psi\rangle$ with its stabilizer $S = \langle P_1, P_2, \ldots, P_n\rangle$ and a Clifford gate V, the group $VSV^\dagger = \langle VP_1V^\dagger, VP_2V^\dagger, \ldots, VP_nV^\dagger\rangle$ is the stabilizer of the state $V|\psi\rangle$ as $VUV^\dagger(V|\psi\rangle) = VU|\psi\rangle = V|\psi\rangle$ for any element $VUV^\dagger$ of $VSV^\dagger$

[2]Each P_j commutes with each other and can't be written as a product of others, i.e., $P_j \notin \langle\{P_1, \ldots, P_k\} \setminus \{P_j\}\rangle$.

with $U \in S$. Thus, the unitary transformation by a Clifford gate V for stabilizer state $|\psi\rangle$ is linked with the conjugation by V for the stabilizer S of $|\psi\rangle$.

It is well-known that *stabilizer circuits* consisting of Clifford gates and the computational basis measurements can be formalized into the dynamics of stabilizers, thus can be efficiently simulated on a classical computer by tracking the generators of stabilizers [Aaronson and Gottesman 2004; Gottesman 1998]. For example, consider the stabilizer state $|00\rangle$ with stabilizer $S = \langle Z_1, Z_1Z_2\rangle$, an H_1 gate followed by a $CNOT_{1,2}$ gate will change the generators Z_1, Z_1Z_2 to

$$CNOT_{1,2}H_1Z_1H_1CNOT_{1,2} = X_1X_2, \qquad CNOT_{1,2}H_1(Z_1Z_2)H_1CNOT_{1,2} = -Y_1Y_2.$$

Then the resulted stabilizer $S' = \langle X_1X_2, -Y_1Y_2\rangle$. The stabilizer state of S' is $(|00\rangle + |11\rangle)/\sqrt{2}$, which is exactly the state $CNOT_{1,2}H_1|00\rangle$.

Quantum error correction and stabilizer codes. The idea of quantum error correction is using a large number n of *physical* qubits to encode k *logical* qubits to protect logical states against the effects of quantum noise. A QEC code is identified by the subspace C, where $C \subseteq \mathbb{C}^{2^n}$ and is isomorphic to $\mathbb{C}^{2^k}$, of all logical states. A QEC program (decoder) for C consists of two-stage procedure of *error detection* and *recovery*:

- In the error detection stage, a list of *syndrome measurements* is performed on physical qubits to detect potential errors, i.e., to check whether the measured state is in the code space C. The measurement results are called the *error syndromes*.
- In the recovery stage, error syndromes are used to decide how to recover the initial state before noise occurs, which can also be said to recover the measured state back to the code space C.

Quantum noise can be continuous, e.g., arbitrary single qubit unitary occurs at some physical qubits, but fortunately, it has been proved that QEC codes that tolerate a set of discrete errors, e.g., Pauli errors (Pauli operators $\{I, X, Y, Z\}$), can also be used to correct continuous errors [Nielsen and Chuang 2010, Theorem 10.2]. For QEC codes, the design of stabilizer codes allows us only to consider Pauli X errors (bit-flip errors) and Pauli Z errors (phase-flip errors).

Example 2.1 (Stabilizer Code). For a list of n-qubit Pauli strings $P_1, P_2, \ldots, P_{n-k}, \bar{Z}_1, \bar{Z}_2, \ldots, \bar{Z}_k \neq \pm I^{\otimes n}$ that are commuting independent with $1 \leq k \leq n$ and $P_j^2 \neq -I^{\otimes n}, \bar{Z}_j^2 \neq -I^{\otimes n}$, an n-qubit stabilizer code encodes the logical computational basis state $|x_1, x_2, \ldots, x_k\rangle_L$ of k logical qubits as an n-qubit stabilizer state of the stabilizer

$$\langle (-1)^{x_1}\bar{Z}_1, (-1)^{x_2}\bar{Z}_2, \ldots, (-1)^{x_k}\bar{Z}_k, P_1, P_2, \ldots, P_{n-k}, \rangle.$$

In particular, the space $V_{\langle P_1, P_2, \ldots, P_{n-k}\rangle}$ equals to

$$\text{span}\{|x_1, x_2, \ldots, x_k\rangle_L | x_i = 0, 1 \ (i = 1, 2, \ldots, k)\} \cong \mathbb{C}^{2^k},$$

which is the code space of this stabilizer code. $P_1, P_2, \ldots, P_{n-k}$ are *stabilizer checks* that take on the role of syndrome measurements, and $\bar{Z}_1, \bar{Z}_2, \ldots, \bar{Z}_k$ are called *logical operators*. ◁

3 RUNNING EXAMPLE: THREE-QUBIT BIT-FLIP CODE

For better understanding, in this section, we illustrate the basic idea of our QSE technique through the example of Fig. 1. Let us recall the classical bit-flip error, which flips a bit $0 \to 1$ and $1 \to 0$. A simple way to protect bits of information against this error is the three-repetition code that encodes each bit with three copies of itself: $0 \to 000$ and $1 \to 111$. Suppose a bit-flip error changes an encoded bit string 000 into 001, then through *majority voting*, it can be recovered back to 000.

Quantum error correction. The program with a two-stage procedure of error detection (Lines 4-9) and recovery (Lines 10-12) in Fig. 1 implements the decoding of the *three-qubit bit-flip code* [Nielsen and Chuang 2010, §10.1.1], which is the quantum counterpart of the above three-repetition code. The code encodes a qubit state $\alpha|0\rangle + \beta|1\rangle$ in three qubits as $\alpha|000\rangle + \beta|111\rangle$ to protect qubits against the quantum bit-flip error: a Pauli X operator at one qubit that takes a qubit state $\alpha|0\rangle + \beta|1\rangle$ to $X(\alpha|0\rangle + \beta|1\rangle) = \alpha X|0\rangle + \beta X|1\rangle = \alpha|1\rangle + \beta|0\rangle$. Suppose an encoded state $\alpha|000\rangle + \beta|111\rangle$ is changed by a quantum bit-flip error into $\alpha|001\rangle + \beta|110\rangle$, which is a superposition of $|001\rangle$ and $|110\rangle$. To recover this state back to $\alpha|000\rangle + \beta|111\rangle$, we need to design some clever quantum measurements (syndrome measurements) so that we can extract useful information for both $|001\rangle$ and $|110\rangle$ without destroying the superposition state. Fortunately, there exist two such measurements, the observables Z_1Z_2 and Z_2Z_3, which are executed in Lines 4-9 of Fig. 1. Here, the *CNOT* gates serve to disentangle the measured qubits, thereby safeguarding the system's information during measurement. Z_1Z_2 checks whether the first and second qubits are the same, and Z_2Z_3 checks whether the second and third qubits are the same. The measurement outcomes of Z_1Z_2 and Z_2Z_3 on state $\alpha|001\rangle + \beta|110\rangle$ will tell us that the first and second qubit are the same, while the second and third qubit are not the same. Thus we can conclude that the bit-flip error occurs at the third qubit, and then we can perform an X gate at the third qubit to recover $\alpha|001\rangle + \beta|110\rangle$ back to $\alpha|000\rangle + \beta|111\rangle$.

Verification of QEC programs. Our goal is to verify the above statement automatically. More generally, given a QEC code and its QEC program with error detection and recovery stages, how can we verify that this program can correct all errors allowed by the QEC code for any state in the code space, or provide some useful information for debugging/testing this program if there are some bugs inside it.

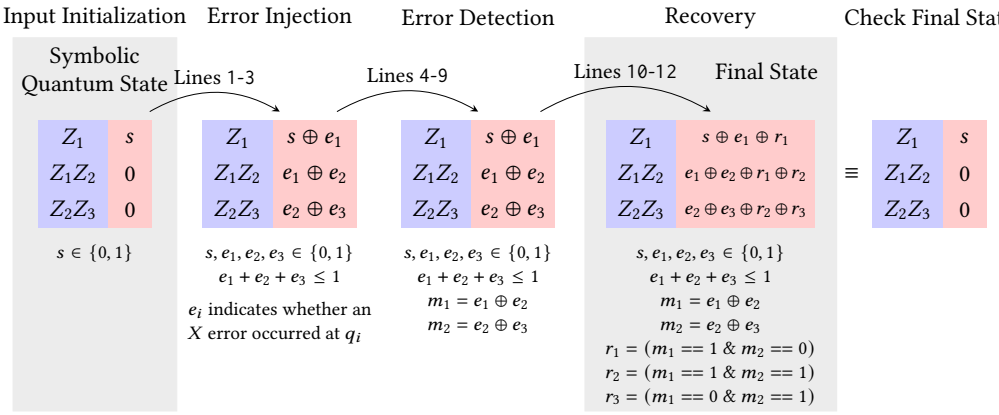


Fig. 2. Illustration of symbolic execution for the quantum program in Fig. 1.

Our solution — symbolic execution. To solve this problem, we develop a symbolic execution method for quantum programs. By introducing a *symbolic quantum state* that can (partially) express possible input quantum states, the quantum program is then "run" on this symbolic quantum state. For the program in Fig. 1, its symbolic execution is illustrated in Fig. 2. More explicitly:

- In the input initialization stage, the symbolic quantum state is the stabilizer $\langle(-1)^s Z_1, Z_1Z_2, Z_2Z_3\rangle$, a list of Pauli operator strings (Z_1, Z_1Z_2, Z_2Z_3) (the light blue part in Fig. 2) with their phases $(s, 0, 0)$ (the light red part in Fig. 2) containing symbolic variable s, that represents the basis $\{|000\rangle, |111\rangle\}$ of the code space. This is because $|000\rangle$ is the stabilizer state of $\langle(-1)^0 Z_1, Z_1Z_2, Z_2Z_3\rangle$ and $|111\rangle$ is the stabilizer state of $\langle(-1)^1 Z_1, Z_1Z_2, Z_2Z_3\rangle$.

- In the error injection stage, instead of enumerating concrete values of e_i, we consider e_i as symbolic variables over $\{0,1\}$ and define $X^{e_1}, X^{e_2}, X^{e_3}$ as symbolic X errors. The constraint $e_1+e_2+e_3 \leq 1$ indicates that at most one X error occurs, thus covering all allowed errors. Define $\bar{X}$ as $X_1^{e_1}X_2^{e_2}X_3^{e_3}$. It can be verified that $\bar{X}(Z_1)\bar{X}^\dagger = (-1)^{s\oplus e_1}Z_1$, $\bar{X}(Z_1Z_2)\bar{X}^\dagger = (-1)^{e_1\oplus e_2}Z_1Z_2$ and $\bar{X}(Z_2Z_3)\bar{X}^\dagger = (-1)^{e_2\oplus e_3}Z_2Z_3$. Thus, as shown in Fig. 2, these symbolic X errors make the symbolic quantum state into $\langle (-1)^{s\oplus e_1}Z_1, (-1)^{e_1\oplus e_2}Z_1Z_2, (-1)^{e_2\oplus e_3}Z_2Z_3\rangle$ with constraint $e_1+e_2+e_3 \leq 1$.
- Then, in the error detection stage, we deduce that $(-1)^{e_1\oplus e_2}Z_1Z_2$ and $(-1)^{e_2\oplus e_3}$ belongs to the stabilizer (group) $\langle (-1)^{s\oplus e_1}Z_1, (-1)^{e_1\oplus e_2}Z_1Z_2, (-1)^{e_2\oplus e_3}Z_2Z_3\rangle$. Consequently, according to the stabilizer formalism [Nielsen and Chuang 2010, Chapter 10.5], the measurement outcomes for Z_1Z_2 and Z_2Z_3 correspond to *symbolic expressions* $m_1 = e_1 \oplus e_2$ and $m_2 = e_2 \oplus e_3$, respectively.
- Finally, in the recovery stage, the phase part of the symbolic quantum state becomes a list of symbolic expressions of $e_i, 1 \leq i \leq 3$ and $r_j, 1 \leq j \leq 3$ with r_j associating to boolean expressions of conditionals. This final state in Fig. 2 under its constraints (including $e_1+e_2+e_3 \leq 1$), can be verified by an SMT solver to be equivalent to the initial symbolic quantum state. That is, the program with error detection and recovery in Fig. 2 can correct at least one (≤ 1) X error for the symbolic quantum state $\langle (-1)^s Z_1, Z_1Z_2, Z_2Z_3\rangle$.

In the same way, the program with error detection and recovery in Fig. 2 can also correct at least one X error for the symbolic quantum state $\langle (-1)^s X_1X_2X_3, Z_1Z_2, Z_2Z_3\rangle$. Hence, according to our Theorem 6.6, we can deduce that any state in the code space is satisfied.

4 QUANTUM SYMBOLIC EXECUTION

The successful application of symbolic execution to our running example (Fig. 2) motivates us to develop a framework for the symbolic execution of quantum programs to be applied in formal analysis and verification of other QEC programs. We first introduce a simple quantum programming language that is suitable for describing QEC programs (§4.1). Then, by introducing symbolic quantum states (§4.2), we extend the framework of standard symbolic execution for classical programs to quantum programs (§4.3).

4.1 OpenQASM-like Language

We choose to extend IBM's OpenQASM2 [Cross et al. 2017], an imperative quantum programming language supporting classical flow control, with external calls (oracle calls) of classical functions.

Syntax. The set **qProgs** of quantum programs is defined by the following syntax:

$$S ::= x := e \mid \bar{y} := F(\bar{x}) \mid U\ \bar{q} \mid c := \texttt{measure}\ q \mid S_1; S_2 \mid \texttt{if}\ b\ \{S_1\}\ \texttt{else}\ \{S_2\}$$

As in classic imperative languages, the assignment $x := e$ evaluates the expression e and assigns the result to the classical variable x. The statement $\bar{y} := F(\bar{x})$ executes an external call of a classical function F, an m-ary function with n outputs, that accepts the values of a list of input variables $\bar{x} = x_1, x_2, \ldots, x_m$ and assigns the outputs $F(\bar{x})$ to a list of classical variables $\bar{y} = y_1, y_2, \ldots, y_n$ in turn. This statement is introduced to wrap some classical algorithms used in QEC programs, while independent of quantum variables. For example, the *Blossom* algorithm [Edmonds 1965; Kolmogorov 2009] for minimum weight perfect matching (MWPM) is used extensively in decoding surface codes (2D topological codes) [Dennis et al. 2002; Fowler et al. 2012]. Moreover, using such algorithms like the Blossom algorithm is usually by calling existing implementations; thus, we only need to care about the conditions that the input and output meet.

There are two quantum constructs involved with qubit variables. The unitary transformation $U\ \bar{q}$ performs a unitary U on a list of qubit variables $\bar{q} = q_1, q_2, \ldots, q_n$. For example, $X\ q$ performs a

Pauli X operator on the qubit q and *CNOT* q_1, q_2 performs a *CNOT* operator on qubits q_1, q_2. The measurement statement $c \coloneqq \texttt{measure}\ q$ measures the qubit variable q in the computational basis and assigns the measurement outcome to the classical variable c.

Control flow is implemented by sequential composition $S_1; S_2$, which executes subprograms S_1 and S_2 sequentially; and conditional statement $\texttt{if}\ b\ \{S_1\}\ \texttt{else}\ \{S_2\}$, which executes the subprogram S_1 if the Boolean expression b is evaluated to be `true`, and otherwise executes the subprogram S_2. For convenience, we use the notation $\texttt{if}\ b\ \{S_1\}$ that omits the `false` branch and does nothing when the Boolean expression b is evaluated to be `false`.

Operational semantics. A *classical-quantum configuration* is a triple $\langle S, \sigma, \rho\rangle$, where S is a quantum program or the termination symbol $\downarrow$, σ is a classical state, and ρ is a density operator denoting the state of qubit variables in S. A probabilistic transition is denoted by $\langle S, \sigma, \rho\rangle \xrightarrow{p} \langle S', \sigma', \rho'\rangle$, where the label p is the transition probability and omitted whenever $p = 1$. Then following the semantics of quantum **while**-language with classical variables given in [Ying and Feng 2011], the operational semantics of our language is defined as a *probabilistic transition relation* $\xrightarrow{p}$ between classical-quantum configurations by the transition rules presented in Fig. 3.

$$\text{(As)}\ \langle x \coloneqq e, \sigma, \rho\rangle \to \langle \downarrow, \sigma[\sigma(e)/x], \rho\rangle \qquad \text{(EC)}\ \langle \bar{y} \coloneqq F(\bar{x}), \sigma, \rho\rangle \to \langle \downarrow, \sigma[F(\sigma(\bar{x}))/\bar{y}], \rho\rangle$$

$$\text{(UT)}\ \langle U\ \bar{q}, \sigma, \rho\rangle \to \langle \downarrow, \sigma, U_{\bar{q}}\rho U_{\bar{q}}^{\dagger}\rangle \qquad \begin{aligned}\text{(M0)}\ &\langle c \coloneqq \texttt{measure}\ q, \sigma, \rho\rangle \xrightarrow{\mathrm{tr}(|0\rangle_q\langle 0|\rho)} \langle \downarrow, \sigma[0/c], \tfrac{|0\rangle_q\langle 0|\rho|0\rangle_q\langle 0|}{\mathrm{tr}(|0\rangle_q\langle 0|\rho)}\rangle\\ \text{(M1)}\ &\langle c \coloneqq \texttt{measure}\ q, \sigma, \rho\rangle \xrightarrow{\mathrm{tr}(|1\rangle_q\langle 1|\rho)} \langle \downarrow, \sigma[1/c], \tfrac{|1\rangle_q\langle 1|\rho|1\rangle_q\langle 1|}{\mathrm{tr}(|1\rangle_q\langle 1|\rho)}\rangle\end{aligned}$$

$$\text{(SC)}\ \frac{\langle S_1, \sigma, \rho\rangle \xrightarrow{p} \langle S_1', \sigma', \rho'\rangle}{\langle S_1; S_2, \sigma, \rho\rangle \xrightarrow{p} \langle S_1'; S_2, \sigma', \rho'\rangle} \qquad \begin{aligned}\text{(CT)}\ &\frac{\sigma \models b}{\langle \texttt{if}\ b\ \{S_1\}\ \texttt{else}\ \{S_2\}, \sigma, \rho\rangle \to \langle S_1, \sigma, \rho\rangle}\\ \text{(CF)}\ &\frac{\sigma \models \neg b}{\langle \texttt{if}\ b\ \{S_1\}\ \texttt{else}\ \{S_2\}, \sigma, \rho\rangle \to \langle S_2, \sigma, \rho\rangle}\end{aligned}$$

Fig. 3. Transition rules for operational semantics. For external call $\bar{y} \coloneqq F(\bar{x})$ with $\bar{x} = x_1, x_2, \ldots, x_m$ and $\bar{y} = y_1, y_2, \ldots, y_n$, we use $\sigma[F(\sigma(\bar{x}))/\bar{y}]$ as a shorthand for $\sigma[f_1/y_1, f_2/y_2, \ldots, f_n/y_n]$ provided by $(f_1, f_2, \ldots, f_n) = F(\sigma(x_1), \sigma(x_2), \ldots, \sigma(x_m))$. For sequential composition $S_1; S_2$, we make convention that $\downarrow; S_2 = S_2$.

The rules (As), (SC), (EC), (CT), and (CF) are the same as in classical or probabilistic programming. In (UT), quantum state ρ is transformed into $U\rho U^{\dagger}$ by the unitary transformation U. In (M0) and (M1), the computational basis measurement at qubit q brings two probabilistic branches with probability $\mathrm{tr}(|0\rangle_q\langle 0|\rho)$ and $\mathrm{tr}(|1\rangle_q\langle 1|\rho)$, respectively, that transform quantum state ρ into $\frac{|0\rangle_q\langle 0|\rho|0\rangle_q\langle 0|}{\mathrm{tr}(|0\rangle_q\langle 0|\rho)}$ and $\frac{|1\rangle_q\langle 1|\rho|1\rangle_q\langle 1|}{\mathrm{tr}(|1\rangle_q\langle 1|\rho)}$, respectively, and assign the corresponding measurement outcomes 0 and 1, respectively, to classical variable c.

For convenience, we also write $\langle S, \sigma, \rho\rangle \xrightarrow{p}{}^{*} \langle S', \sigma', \rho'\rangle$ if there is a sequence of transitions

$$\langle S, \sigma, \rho\rangle \xrightarrow{p_1} \langle S_1, \sigma_1, \rho_1\rangle \xrightarrow{p_2} \cdots \xrightarrow{p_n} \langle S', \sigma', \rho'\rangle$$

such that $n \geq 0$ and $p = \prod_{j=1}^{n} p_j$.

4.2 Symbolic Quantum States

Just like using symbols to represent a range of values and building expressions over constants and symbols, we want to build a term that depends on some symbols, and when concrete values substitute these symbols, this term becomes a quantum state—density operator. We begin with a few examples of density operators with parameters.

Example 4.1 (Handling quantum states of single-qubit). Consider a qubit system with state space $\mathcal{H} = \text{span}\{|0\rangle, |1\rangle\}$. A pure qubit state can be written as

$$|\psi(\theta,\phi)\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle, \theta \in [0,\pi], \phi \in [0,2\pi],$$

with a global phase ignored [Nielsen and Chuang 2010, Bloch Sphere]. Then, its corresponding density operator is $\psi(\theta,\phi) = |\psi(\theta,\phi)\rangle\langle\psi(\theta,\phi)|$. By replacing variable θ with a symbol s_0 over $[0,\pi]$ and variable ϕ with a symbol s_1 over $[0,2\pi]$, we would obtain a symbolic quantum state $\psi(s_0,s_1)$. ◁

Example 4.2 (Handling quantum states of n-qubit). Consider an n-qubit system with state space $\mathcal{H} = \text{span}\{|0\rangle, |1\rangle, \ldots, |2^n-1\rangle\}$.

(1) For any $(\alpha_0, \alpha_1, \ldots, \alpha_{2^n-1}) \in \mathbb{C}^{2^n}$, let $|g(\alpha_0,\alpha_1,\ldots,\alpha_{2^n-1})\rangle = 1/\sqrt{\sum_{k=0}^{2^n-1}|\alpha_k|^2}\sum_{i=0}^{2^n-1}\alpha_i|i\rangle$ parameterize all n-qubit pure states. Then, with each α_j replaced by a symbol s_j over $\mathbb{C}$, $g(s_0,s_1,\ldots,s_{2^n-1})$ would be the symbolic quantum state we want.

(2) Moreover, we can choose $|g_j(\alpha_{j,0},\alpha_{j,1},\ldots,\alpha_{j,2^n-1})\rangle = 1/\sqrt{\sum_{k=0}^{2^n-1}|\alpha_{j,k}|^2}\sum_{i=0}^{2^n-1}\alpha_{j,i}|i\rangle$ and $(\beta_0, \beta_1, \ldots, \beta_{2^n-1}) \in \mathbb{C}^{2^n}$ to define

$$h(\beta_0,\beta_1,\ldots,\beta_{2^n-1},\ldots,\alpha_{j,k},\ldots) = \sum_{j=0}^{2^n-1}\frac{|t_j|^2}{\sum_{j=0}^{2^n-1}|t_j|^2}g_j(\alpha_{j,0},\alpha_{j,1},\ldots,\alpha_{j,2^n-1}),$$

which parameterizes all n-qubit mixed states. Then, with $t_j, s_{i,j}$ being symbols over $\mathbb{C}$, $h(\beta_0, \beta_1, \ldots, \beta_{2^n-1}, \ldots, \alpha_{j,k}, \ldots)$ would be a symbolic quantum state for n-qubit mixed states.

◁

Building on Examples 4.1 and 4.2, we introduce the general form of symbolic quantum states.

Definition 4.3 (Symbolic Quantum State). A symbolic quantum state $\widetilde{\rho}$ is an n-ary map

$$G : C_1 \times C_2 \times \cdots \times C_n \to \mathcal{D}$$

with each argument substituted by a symbol over its domain, i.e.,

$$\widetilde{\rho} \triangleq G(s_1, s_2, \ldots, s_n),$$

where n is a positive integer, for each $1 \le i \le n$, C_i is a subset of complex numbers, and s_i is the symbol for the value ranging over C_i, and $\mathcal{D}$ is the set of quantum states under consideration, e.g., the space of m-qubit mixed states if the program contains m qubits.

Quantum operations over symbolic quantum states. To use symbolic quantum states to capture the quantum states in quantum programs, we need to further define unitary transformations and quantum measurements over them.

Definition 4.4 (Quantum operations over symbolic quantum states). Let $\widetilde{\rho}$ be a symbolic quantum state for qubit variables $q_1, q_2, \ldots, q_n$. Then:

- For a unitary statement $U\ \bar{q}$ with qubits $\bar{q} \subseteq \{q_1, q_2, \ldots, q_n\}$, the unitary transformation function ut is defined by

$$ut(U, \bar{q}, \widetilde{\rho}) = U_{\bar{q}}\widetilde{\rho}U_{\bar{q}}^{\dagger}, \tag{1}$$

where $U_{\bar{q}} \triangleq U_{\bar{q}} \otimes \mathbb{1}_{\{q_1,q_2,\ldots,q_n\}\setminus\bar{q}}$ is the operation that performs a local unitary U on qubits $\bar{q}$ and does nothing with the remaining qubits.

- For a measurement statement $c \coloneqq \texttt{measure}\ q$ with $q \in \{q_1, q_2, \ldots, q_n\}$, the measurement function m is defined by

$$m(q, \widetilde{\rho}) = \left(s, \mathrm{tr}\big(|s\rangle_q\langle s|\widetilde{\rho}\big), \frac{|s\rangle_q\langle s|\widetilde{\rho}|s\rangle_q\langle s|}{\mathrm{tr}(|s\rangle_q\langle s|\widetilde{\rho})}\right) \tag{2}$$

  with s a newly introduced symbol over $\{0, 1\}$, where $|s\rangle_q\langle s| \triangleq |s\rangle_q\langle s| \otimes \mathbb{1}_{\{q_1,q_2,\ldots,q_n\}\setminus\{q\}}$ is the measurement operator that act on qubit q as $|s\rangle\langle s|$ and does nothing with the remaining qubits. The output is a triple of a newly introduced symbol s ranging over measurement outcomes $\{0, 1\}$, a symbolic expression $p(s) = \mathrm{tr}\big(|s\rangle_q\langle s|\widetilde{\rho}\big)$ for the probability of obtaining outcome s when performing measurement at qubit q of state $\widetilde{\rho}$ and a symbolic quantum state $\widetilde{\rho}'(s) = \frac{|s\rangle_q\langle s|\widetilde{\rho}|s\rangle_q\langle s|}{\mathrm{tr}(|s\rangle_q\langle s|\widetilde{\rho})}$ that represents the state after this measurement.

Efficient representations. According to our Definition 4.3, there exist infinitely many symbolic quantum states $F : C_1 \times C_2 \times \cdots \times C_n \to \mathcal{D}$. However, in practical applications, we must carefully consider which type of F can be efficiently implemented on a classical computer. If we trivially employ the symbolic quantum state in Example 4.2, we would encounter several challenges:

- **Exponential complexity**: the number of symbols required, i.e., the dimension of the state space, grows exponentially with respect to the number of qubits, leading to computational complexity beyond what classical computers can handle.
- **Lack of powerful solvers**: without specialized solvers, addressing the problems arising from QSE would become challenging.

On the other hand, the development of SMT solvers promotes the application of classical SE. Similarly, we must make the symbolic quantum state take advantage of the existing solvers.

Fortunately, for QEC programs, we can construct suitable symbolic quantum states with efficient representations—symbolic stabilizer states, which we will discuss later in §6.

4.3 Quantum Symbolic Execution

Following the idea of classical SE, we come up with an SE framework for quantum programs (QSE) that maintains a symbolic configuration $\langle S, \widetilde{\sigma}, \widetilde{\rho}, P, \varphi\rangle$, where:

- S is the quantum program to be executed.
- $\widetilde{\sigma}$ is the symbolic classical state, which is the same as the symbolic state in classical SE, and maps classical variables of the program to symbolic expressions over constants and symbolic values.
- $\widetilde{\rho}$ is the *symbolic quantum state*, equipped with a unitary transformation function $ut(U, \bar{q}, \widetilde{\rho})$ for statement $U\ \bar{q}$ and a measurement function $m(q, \widetilde{\rho})$ for statement $c \coloneqq \texttt{measure}\ q$, for qubit variables of the program.
- P is a set that records the symbolic probabilities corresponding to the outcomes of quantum measurements in the program. At the beginning of the execution, $P = \emptyset$.
- φ is the path condition, i.e., a conjunctive formula that expresses a set of assumptions on the symbols due to external calls and branches taken in an execution path of the quantum program.

As a natural extension to the operational semantics of quantum programs in Fig. 3, the QSE changes the symbolic configuration according to the QSE rules presented in Fig. 4. We also write $\langle S, \widetilde{\sigma}, \widetilde{\rho}, P, \varphi\rangle \to^* \langle S', \widetilde{\sigma}', \widetilde{\rho}', P', \varphi'\rangle$ if there is a sequence of transitions

$$\langle S, \widetilde{\sigma}, \widetilde{\rho}, P, \varphi\rangle \to \langle S_1, \widetilde{\sigma}_1, \widetilde{\rho}_1, P_1, \varphi_1\rangle \to \cdots \to \langle S_n, \widetilde{\sigma}_n, \widetilde{\rho}_n, P_n, \varphi_n\rangle = \langle S', \widetilde{\sigma}', \widetilde{\rho}', P', \varphi'\rangle$$

such that $n \geq 0$.

We assume that the function F in an external call has a logical formula $C_F(\bar{x}, \bar{y})$ asserting the condition that input and output must meet, i.e., $C_F(\bar{x}, F(\bar{x})) \equiv \texttt{true}$. The existence of this condition

$$\text{(S-As)}\ \langle x \coloneqq e, \widetilde{\sigma}, \widetilde{\rho}, P, \varphi\rangle \to \langle\downarrow, \widetilde{\sigma}[\widetilde{\sigma}(e)/x], \widetilde{\rho}, P, \varphi\rangle$$

$$\text{(S-EC)}\ \langle \bar{y} \coloneqq F(\bar{x}), \widetilde{\sigma}, \widetilde{\rho}, P, \varphi\rangle \to \langle\downarrow, \widetilde{\sigma}[\bar{s}_{\bar{y}}/\bar{y}], \widetilde{\rho}, P, \varphi \wedge C_F(\widetilde{\sigma}(\bar{x}), \bar{s}_{\bar{y}})\rangle$$

$$\text{(S-UT)}\ \langle U\ \bar{q}, \widetilde{\sigma}, \widetilde{\rho}, P, \varphi\rangle \to \langle\downarrow, \widetilde{\sigma}, ut(U, \bar{q}, \widetilde{\rho}), P, \varphi\rangle$$

$$\text{(S-M)}\ \langle c \coloneqq \texttt{measure}\ q, \widetilde{\sigma}, \widetilde{\rho}, P, \varphi\rangle \to \langle\downarrow, \widetilde{\sigma}[s/c], \widetilde{\rho}'(s), P \cup \{(s, p(s))\}, \varphi\rangle \quad \text{where } (s, p(s), \widetilde{\rho}'(s)) = m(q, \widetilde{\rho})$$

$$\text{(S-SC)}\ \frac{\langle S_1, \widetilde{\sigma}, \widetilde{\rho}, P, \varphi\rangle \to \langle S_1', \widetilde{\sigma}', \widetilde{\rho}', P', \varphi'\rangle}{\langle S_1; S_2, \widetilde{\sigma}, \widetilde{\rho}, P, \varphi\rangle \to \langle S_1'; S_2, \widetilde{\sigma}', \widetilde{\rho}', P', \varphi'\rangle}$$

$$\text{(S-CT)}\ \langle \texttt{if}\ b\ \{S_1\}\ \texttt{else}\ \{S_2\}, \widetilde{\sigma}, \widetilde{\rho}, P, \varphi\rangle \to \langle S_1, \widetilde{\sigma}, \widetilde{\rho}, P, \varphi \wedge \widetilde{\sigma}(b)\rangle$$

$$\text{(S-CF)}\ \langle \texttt{if}\ b\ \{S_1\}\ \texttt{else}\ \{S_2\}, \widetilde{\sigma}, \widetilde{\rho}, P, \varphi\rangle \to \langle S_2, \widetilde{\sigma}, \widetilde{\rho}, P, \varphi \wedge \neg\widetilde{\sigma}(b)\rangle$$

Fig. 4. Symbolic execution rules for quantum programs. For external call $\bar{y} \coloneqq F(\bar{x})$, $C_F(\widetilde{\sigma}(\bar{x}), \bar{s}_{\bar{y}})$ is the condition that the input and output should satisfy for F, i.e., $C_F(\bar{x}, F(\bar{x})) \equiv \texttt{true}$ for any concrete values $\bar{x}$, where $\bar{s}_{\bar{y}} = s_{y_1}, s_{y_2}, \ldots, s_{y_n}$ is a list of newly introduced symbols for output variables $\bar{y} = y_1, y_2, \ldots, y_n$.

is obvious, e.g., we can choose $C_F(\bar{x}, \bar{y}) = (\bar{y} == F(\bar{x}))$, but such a choice may not be efficient to (symbolically) compute for any F.

For measurement statements, our defined function m in Definition 4.4 uses a symbol to represent the measurement outcomes, avoiding branches like conditional statements. Since a lot of measurements are commonly required in QEC, this trick has the advantage of not facing the exponential complexity over the number of measurements.

5 SOUNDNESS THEOREM

In this section, we present the soundness theorem for QSE (namely, the QSE rules in Fig. 4) with respect to the operational semantics. The proof for this section is located in Appendix C of the extended version [Fang and Ying 2023].

To precisely state the theorem, we need the notion of instantiation for symbolic quantum states.

Definition 5.1 (Instantiation). Let (σ, ρ) be a pair of classical state and quantum state, and let $(\widetilde{\sigma}, \widetilde{\rho})$ be a pair of symbolic classical state and symbolic quantum state. Then we say that (σ, ρ) is an instantiation of $(\widetilde{\sigma}, \widetilde{\rho})$ under a path condition φ, written

$$(\sigma, \rho) \models_\varphi (\widetilde{\sigma}, \widetilde{\rho})$$

if there is a *valuation* V that assigns every symbol in $\widetilde{\sigma}, \widetilde{\rho}$ and φ with concrete values such that $V(\varphi) = \texttt{true}$, $\sigma = V(\widetilde{\sigma})$ and $\rho = V(\widetilde{\rho})$. In particular, if $\varphi \equiv \texttt{true}$, we simply write $(\sigma, \rho) \models (\widetilde{\sigma}, \widetilde{\rho})$.

Our soundness theorem relies on the following two key lemmas about functions ut and m for symbolic quantum states introduced in Definition 4.4.

Lemma 5.2 (Correctness of function ut). *For a unitary transformation statement $U\ \bar{q}$, a path condition φ, a pair (σ, ρ) of classical state and quantum state, and a pair $(\widetilde{\sigma}, \widetilde{\rho})$ of symbolic classical state and symbolic quantum state, if $\langle U\ \bar{q}, \sigma, \rho\rangle \to \langle\downarrow, \sigma', \rho'\rangle$ and $(\sigma, \rho) \models_\varphi (\widetilde{\sigma}, \widetilde{\rho})$, then*

$$(\sigma', \rho') \models_\varphi (\widetilde{\sigma}, ut(U, \bar{q}, \widetilde{\rho})).$$

Lemma 5.3 (Correctness of function m). *For a measurement statement $c \coloneqq \texttt{measure}\ q$, a path condition φ, a pair (σ, ρ) of classical state and quantum state, and a pair $(\widetilde{\sigma}, \widetilde{\rho})$ of symbolic classical*

state and symbolic quantum state, if $\langle c \coloneqq \texttt{measure}\ q, \sigma, \rho\rangle \xrightarrow{p} \langle\downarrow, \sigma', \rho'\rangle$ *and* $(\sigma, \rho) \models_\varphi (\widetilde{\sigma}, \widetilde{\rho})$*, then*

$$(\sigma', \rho') \models_\varphi (\widetilde{\sigma}[s/c], \widetilde{\rho}'(s)),$$

where $(s, p(s), \widetilde{\rho}'(s)) = m(q, \widetilde{\rho}')$.

Now we are ready to present the soundness theorem.

Theorem 5.4 (Soundness of QSE). *For any quantum program S, any pair (σ, ρ) of classical state and quantum state, any pair $(\widetilde{\sigma}, \widetilde{\rho})$ of symbolic classical state and symbolic quantum state, any set P of symbolic probabilities and any path condition φ, if*

$$(\sigma, \rho) \models_\varphi (\widetilde{\sigma}, \widetilde{\rho}), \quad \langle S, \sigma, \rho\rangle \xrightarrow{p} \langle S', \sigma', \rho'\rangle, \quad \langle S, \widetilde{\sigma}, \widetilde{\rho}, P, \varphi\rangle \to \langle S', \widetilde{\sigma}', \widetilde{\rho}', P', \varphi'\rangle$$

then $(\sigma', \rho') \models_{\varphi'} (\widetilde{\sigma}', \widetilde{\rho}')$*. Moreover, there is a valuation V such that* $V(\widetilde{\sigma}) = \sigma, V(\widetilde{\rho}) = \rho, V(\varphi) = \texttt{true}, V(\widetilde{\sigma}') = \sigma', V(\widetilde{\rho}') = \rho', V(\varphi') = \texttt{true}$ *and if* $P' = P \cup \{(s, p(s)\}$*, then* $p = V(p(s))$.

6 SYMBOLIC STABILIZER STATES

As we briefly discussed in §4.2, a general symbolic quantum state cannot be represented in an efficient way. In this section, we propose a special class of symbolic quantum state with efficient representations, called *symbolic stabilizer state*. This class of symbolic quantum states is what we need in the analysis and verification of QEC programs. Then we extend this symbolic representation to unitaries and measurements so that we have an efficient QSE framework for our application in the analysis and verification of stabilizer codes.

6.1 Symbolic Stabilizer States

First, we present an efficient symbolic representation of stabilizer states by introducing symbols into the phases of Pauli strings as follows.

Definition 6.1 (Symbolic stabilizer state). For any commuting independent set $\{P_1, P_2, \ldots, P_n \mid p_j \neq \pm I^{\otimes n}, P_j^2 \neq -I^{\otimes n}\}$ of n-qubit Pauli strings with size n and n Boolean functions $f_1, f_2, \ldots, f_n$ over m Boolean variables, let $|\psi(b_1, b_2, \ldots, b_m)\rangle$ denote a stabilizer state of

$$\langle (-1)^{f_1(b_1,\ldots,b_m)} P_1, \ldots, (-1)^{f_n(b_1,\ldots,b_m)} P_n\rangle.$$

A symbolic stabilizer state is defined as a symbolic quantum state

$$\widetilde{\rho} = \psi(s_1, s_2, \ldots, s_m) = |\psi(s_1, s_2, \ldots, s_m)\rangle\langle\psi(s_1, s_2, \ldots, s_m)|$$

with $s_1, s_2, \ldots, s_m$ being symbols over Boolean values.

Since the global phase of $|\psi(b_1, b_2, \ldots, b_m)\rangle$ is canceled out in $\psi(b_1, b_2, \ldots, b_m)$, the stabilizer $\langle (-1)^{f_1(b_1,\ldots,b_m)} P_1, \ldots, (-1)^{f_n(b_1,\ldots,b_m)} P_n\rangle$ corresponds to a unique density operator $\psi(b_1, b_2, \ldots, b_m)$. Therefore, we slightly abuse the notation and also write the density operator

$$\psi(b_1, b_2, \ldots, b_m) = \langle (-1)^{f_1(b_1,\ldots,b_m)} P_1, \ldots, (-1)^{f_n(b_1,\ldots,b_m)} P_n\rangle,$$

and the symbolic stabilizer state

$$\widetilde{\rho} = \langle (-1)^{f_1(s_1,\ldots,s_m)} P_1, \ldots, (-1)^{f_n(s_1,\ldots,s_m)} P_n\rangle.$$

Example 6.2. In our running example (Fig. 2), the initial symbolic stabilizer state is

$$\langle (-1)^s Z_1, (-1)^0 Z_1Z_2, (-1)^0 Z_2Z_3\rangle$$

with s being a Boolean symbol. It represents a set of quantum states $\{|000\rangle, |111\rangle\}$ as $|000\rangle$ and $|111\rangle$ are the stabilizer states of $\langle Z_1, Z_1Z_2, Z_2Z_3\rangle$ and $\langle -Z_1, Z_1Z_2, Z_2Z_3\rangle$, respectively. ◁

6.2 Quantum Operations over Symbolic Stabilizer States

Now we can extend the symbolic representation for stabilizer states to unitaries and measurements. Indeed, the unitary transformation function ut and measurement function m defined in §4.2 can be naturally lifted to functions on stabilizers by the stabilizer formalism [Nielsen and Chuang 2010, Chapter 10.5]. The definitions are as follows, and their correctness for Lemmas 5.2 and 5.3 is available in Appendix D of the extended version [Fang and Ying 2023].

Definition 6.3 (Quantum operations over symbolic stabilizer states). Let

$$\widetilde{\rho} = \langle (-1)^{f_1(s_1,\ldots,s_m)} P_1, \ldots, (-1)^{f_n(s_1,\ldots,s_m)} P_n \rangle$$

be a symbolic stabilizer state for qubit variables $q_1, q_2, \ldots, q_n$. Then

- For a Clifford unitary statement $V\ \bar{q}$ with qubits $\bar{q} \subseteq \{q_1, \ldots, q_n\}$, the unitary transformation function ut is defined as $ut(V, \bar{q}, \widetilde{\rho}) =$

$$\langle (-1)^{f_1(s_1,\ldots,s_m)} V_{\bar{q}} P_1 V_{\bar{q}}^\dagger, \ldots, (-1)^{f_n(s_1,\ldots,s_m)} V_{\bar{q}} P_n V_{\bar{q}}^\dagger \rangle.$$

- For a measurement statement $c \coloneqq \texttt{measure}\ q$ with $q \in \{q_1, q_2, \ldots, q_n\}$, the definition of measurement function m is divided into two cases:

(1) Z_q commutes with all $P_j, 1 \le j \le n$. In this case, there exist a Boolean value b and a list of indexes $1 \le j_1 \le j_2 \le \ldots \le j_k \le n$ with $1 \le k \le n$ such that $Z_q = (-1)^b P_{j_1} P_{j_2} \cdots P_{j_k}$. The measurement does not change the state and implies a determinate outcome [Nielsen and Chuang 2010, §10.5.3]. The function m is defined as $m(q, \widetilde{\rho}) =$

$$(b \oplus f_{j_1}(s_1, \ldots, s_m) \oplus \cdots \oplus f_{j_k}(s_1, \ldots, s_m), \{\}, \widetilde{\rho}).$$

(2) Z_q anti-commutes one or more of P_j. In this case, without loss of generality, we can assume P_1 anti-commutes with Z_q and $P_2, \ldots, P_n$ commute with Z_q[3]. Then, the function m is defined as $m(q, \widetilde{\rho}) =$

$$\left(s, \frac{1}{2}, \langle (-1)^s Z_q, (-1)^{f_2(s_1,\ldots,s_m)} P_2, \ldots, (-1)^{f_n(s_1,\ldots,s_m)} P_n \rangle\right),$$

where s is a newly introduced Boolean symbol for the measurement outcome.

Example 6.4. In our running example Fig. 2, measuring Z_1Z_2 (Lines 4-6 in Fig. 1) is implemented by $CNOT\ q_1, q_2; m_1 \coloneqq \texttt{measure}\ q_2; CNOT\ q_1, q_2$. It changes the symbolic stabilizer state $\widetilde{\rho_2} = \langle (-1)^{s \oplus e_1} Z_1, (-1)^{e_1 \oplus e_2} Z_1Z_2, (-1)^{e_2 \oplus e_3} Z_2Z_3 \rangle$ as follows:

(1) $CNOT\ q_1, q_2$ transforms $\widetilde{\rho_2}$ to $\widetilde{\rho_3}$

$$= \langle (-1)^{s \oplus e_1} CNOT_{1,2} Z_1 CNOT_{1,2}, (-1)^{e_1 \oplus e_2} CNOT_{1,2}(Z_1Z_2) CNOT_{1,2}, (-1)^{e_2 \oplus e_3} CNOT_{1,2}(Z_2Z_3) CNOT_{1,2} \rangle$$
$$= \langle (-1)^{s \oplus e_1} Z_1, (-1)^{e_1 \oplus e_2} Z_2, (-1)^{e_2 \oplus e_3} Z_1Z_2Z_3 \rangle.$$

(2) Then $m_1 \coloneqq \texttt{measure}\ q_2$ performs measurement Z_2 on $\widetilde{\rho_3}$. Since Z_2 commutes with all Pauli strings in $\widetilde{\rho_3}$, we follow the case 1 in Definition 6.3 to get $Z_2 = (-1)^0 Z_2$, thus $m(q_2, \widetilde{\rho_3}) = (e_1 \oplus e_2, \{\}, \widetilde{\rho_3})$ leading to outcome $m_1 = e_1 \oplus e_2$ and the quantum state remains $\widetilde{\rho_3}$.

(3) Finally, $CNOT\ q_1, q_2$ transforms $\widetilde{\rho_3}$ back to $\widetilde{\rho_2}$.

◁

[3] If there is other $P_j, j \ge 2$ anti-commutes with Z_q, we can replace the $(-1)^{f_j(s_1,\ldots,s_m)} P_j$ by $(-1)^{f_1(s_1,\ldots,s_m) + f_j(s_1,\ldots,s_m)} P_1P_j$ in the generating set of $\widetilde{\rho}$, and it will result in the same $\widetilde{\rho}$. Then Z_q only commute with P_1P_j.

Efficient implementation. The two functions ut and m defined above require three subroutines:

- A subroutine that computes $VPV^\dagger$ for a Pauli string P and a Clifford gate V.
- A subroutine that determines whether two Pauli strings P_1 and P_2 are commutable or not.
- A subroutine that produce a Boolean value b and a list of indexes $1 \le j_1, j_2, \ldots, j_k \le n$ with $1 \le k \le n$ such that $(-1)^b Z_q = P_{j_1} P_{j_2} \cdots P_{j_k}$ for a qubit q and a stabilizer $\langle P_1, P_2, \ldots, p_n \rangle$ if Z_q commutes with all P_j.

These subroutines are standard in stabilizer formalism, and there are already efficient implementations [Aaronson and Gottesman 2004; Gidney 2021] for them. We provide a detailed discussion of the implementation in Appendix B of the extended version [Fang and Ying 2023].

6.3 Adequacy

In this subsection, we further show that our symbolic representation is adequate for our target application in the analysis and verification of stabilizer codes. The proof for this section is located in Appendix D of the extended version [Fang and Ying 2023].

To answer whether a decoder of a stabilizer code is correct, we need to verify that for any quantum state ρ in the code space as the input state, the program S consisting of error injection (inject errors allowed by the stabilizer code) and this decoder will output the same state ρ. Thus, we need to verify whether the input quantum state and the output quantum state are the same.

Formally, let S be a quantum program that only uses Clifford gates and σ be a classical state. For $1 \le k \le n$, assume that an n-qubit stabilizer code encoding k logical qubits with syndrome operators $P_1, P_2, \ldots, P_{n-k}$ and logical operators $L_1, L_2, \ldots, L_k$. Let $|x_1, x_2, \ldots, x_k\rangle_L$ with $x_j \in \{0, 1\}$ denoting the logical computational basis state of the code as in Example 2.1, and $\bar{H} = H_{L,1} \otimes H_{L,2} \otimes \cdots \otimes H_{L,k}$ with $H_{L,j}$ being the logical Hardamard gate, which is a Clifford gate, for the j-th logical qubit. It is clear that states $|x_1, x_2, \ldots, x_k\rangle_L$ and $\bar{H}|x_1, x_2, \ldots, x_k\rangle_L$ are both stabilizer states. We write $\mathcal{T}$ for the set of (density operators of) these states with $x_j \in \{0, 1\} (1 \le j \le k)$. Then we have:

Lemma 6.5. *If for any $\rho \in \mathcal{T}$, it holds that*

$$\langle S, \sigma, \rho \rangle \xrightarrow{p} {}^* \langle \downarrow, \sigma', \rho' \rangle \text{ with } p > 0 \text{ implies } \rho' = \rho \tag{3}$$

then (3) holds for all quantum state ρ in the code space.

Theorem 6.6 (Adequacy of symbolic stabilizer states). *Let:*

(1) $\widetilde{\rho}_1$ be the symbolic stabilizer state $\langle P_1, \ldots, P_{n-k}, (-1)^{s_1} L_1, \ldots, (-1)^{s_k} L_k \rangle$; and
(2) $\widetilde{\rho}_2$ be the symbolic stabilizer state $\langle \bar{H} P_1 \bar{H}, \ldots, \bar{H} P_{n-k} \bar{H}, (-1)^{s_1} \bar{H} L_1 \bar{H}, \ldots, (-1)^{s_k} \bar{H} L_k \bar{H} \rangle$ with s_j symbols over Boolean values.
(3) $\widetilde{\sigma}$ be a classical symbolic state that does not contains $s_j, 1 \le j \le k$.

If for $\widetilde{\rho} = \widetilde{\rho}_1, \widetilde{\rho}_2$,

$$\langle S, \widetilde{\sigma}, \widetilde{\rho}, \emptyset, \mathtt{true} \rangle \to^* \langle \downarrow, \widetilde{\sigma}', \widetilde{\rho}', P', \varphi' \rangle \text{ implies } \varphi' \models \widetilde{\rho}' = \widetilde{\rho},$$

then for any quantum state ρ in the code space and any valuation V,

$$\langle S, V(\widetilde{\sigma}), \rho \rangle \xrightarrow{p} {}^* \langle \downarrow, \sigma', \rho' \rangle \text{ with } p > 0 \text{ implies } \rho' = \rho.$$

This theorem tells us that we only need to check two symbolic stabilizer states to verify that a program's initial and final quantum states are the same.

7 CONDITIONAL APPLICATION OF PAULI GATES WITHOUT FORKING

In classical SE, a branch of conditional triggers a path fork and update to the path constraints as rules (S-CT) and (S-CF) in Fig. 4. However, there may be many conditional statements in QEC programs, e.g., the conditionals in our running example (Lines `10-12` in Fig. 1) are designed for each physical qubit; thus, when the number of physical qubits is large, the number of conditional statements required is also large. Then, we will face the problem of path explosion. Fortunately, with a newly designed rule (see Eq. (4) later), our symbolic stabilizer states can solve this problem.

Symbolic Pauli gates. We observe that to correct the X or Z error, QEC programs usually use a conditional statement to determine whether to apply the X or Z gate in the recovery stage, e.g., the line `10` in Fig. 1:

$$\texttt{if } m_1 == 1 \ \& \ m_2 == 0 \ \{X \ q_1\}$$

This application of X and Z gates inspires us to define the following symbolic Pauli gates.

Definition 7.1 (Symbolic Pauli gates). For a symbolic Boolean expression e, a Pauli gate $\tau \in \{X, Y, Z\}$, and a qubit q, we define the symbolic Pauli gate τ^e at qubit q as the conditional statement `if` e $\{\tau \ q\}$; that is, τ^e applies τ to qubit q if e is evaluated to be `true` and I if e is evaluated to be `false`. For convenience, we also add a new statement

$$\tau[e] \ q \equiv \texttt{if } e \ \{\tau \ q\}$$

for symbolic Pauli gate τ^e.

QSE rule for symbolic Pauli gates. Our symbolic stabilizer states are very friendly to symbolic Pauli gates. For any Pauli string

$$P = (i)^k P_1 \otimes P_2 \otimes \cdots \otimes P_n,$$

conjugation by a symbolic Pauli gate τ^e at qubit q transforms it into

$$(i)^k P_1 \otimes \cdots \otimes (\tau^e P_q (\tau^\dagger)^e) \otimes \cdots \otimes P_n = (-1)^{e \cdot [\tau \neq P_q]} P,$$

where $[\tau \neq P_q] = 1$ if $\tau \neq P_q$, otherwise $[\tau \neq P_q] = 0$. We see that τ^e only changes the phase of the Pauli string. Thus, for symbolic stabilizer states, the unitary transformation by a symbolic Pauli gate τ^e can be rewritten as

$$\begin{aligned} &ut(\tau^e, q, \langle (-1)^{f_1(s_1,\ldots,s_m)} P_1, \ldots, (-1)^{f_n(s_1,\ldots,s_m)} P_n \rangle) \\ = &\langle (-1)^{f_1(s_1,\ldots,s_m) \oplus e[\tau \neq P_{1,q}]} P_1, \ldots, (-1)^{f_n(s_1,\ldots,s_m) \oplus e[\tau \neq P_{n,q}]} P_n \rangle. \end{aligned}$$

where the q-th Pauli gate of P_k is $P_{k,q}$. Then, the QSE rule for $\tau[e] \ q$ can be redefined as

$$\text{(S-SP)} \ \langle \tau[e] \ q, \widetilde{\sigma}, \widetilde{\rho}, P, \varphi \rangle \rightarrow \langle \downarrow, \widetilde{\sigma}, ut(\tau^{\widetilde{\sigma}(e)}, q, \widetilde{\rho}), P, \varphi \rangle \tag{4}$$

Conditional without forking. The rule (S-SP) provides a simpler and more effective way to handle the statement $\tau[e] \ q$ than rules (S-CT) and (S-CF) for conditionals in Fig. 4. For example, consider the program

$$X[m_1 == 0] \ q_1; X[m_2 == 0] \ q_2; X[m_3 == 0] \ q_3,$$

which is also

$$\texttt{if } m_1 == 0 \ \{X \ q_1\}; \texttt{if } m_2 == 0 \ \{X \ q_2\}; \texttt{if } m_3 == 0 \ \{X \ q_3\},$$

for a symbolic configuration, the rule (S-SP) will eventually produce one symbolic configuration; however, rules (S-CT) and (S-CF) will produce 8 symbolic configurations.

Inserting symbolic Pauli errors through symbolic Pauli gates. The error injection in our running example (see Fig. 1) can be done by the program

$$X[e_1 == 1]\ q_1; X[e_2 == 1]\ q_2; X[e_3 == 1]\ q_3$$

with constraints $e_1 + e_2 + e_3 \leq 1, 0 \leq e_1, e_2, e_3 \leq 1$, which characterizes all possible X errors with at most 1 location of physical qubits. More generally, the following program

$$X[e_1 == 1]\ q_1; X[e_2 == 1]\ q_2; \ldots; X[e_n == 1]\ q_n \tag{5}$$

with constraints $e_1 + e_2 + \cdots + e_n \leq d$, $0 \leq e_j \leq 1, 1 \leq j \leq n$, captures all possible X errors at most d locations of n physical qubits. By doing this, we efficiently overcome the challenge of dealing with a large number of samples of possible Pauli errors, which grows exponentially as $\binom{n}{d}$.

8 EXPERIMENTAL EVALUATION

We implemented our general QSE framework together with the special support of symbolic stabilizer states in a prototype tool called QuantumSE.jlas a Julia [Bezanson et al. 2017] package. For an initialized symbolic configuration $\langle S, \widetilde{\sigma}, \widetilde{\rho}, P, \varphi\rangle$, QuantumSE.jl constructs a set of the terminal configurations $\langle \downarrow, \widetilde{\sigma}', \widetilde{\rho}', P', \varphi'\rangle$ s.t. $\langle S, \widetilde{\sigma}, \widetilde{\rho}, P, \varphi\rangle \rightarrow^* \langle \downarrow, \widetilde{\sigma}', \widetilde{\rho}', P', \varphi'\rangle$ by the QSE rules in Fig. 4. Then, for an assertion that we are interested in at a terminal configuration $\langle \downarrow, \widetilde{\sigma}', \widetilde{\rho}', P', \varphi'\rangle$, e.g., $\varphi \wedge \varphi' \models \widetilde{\rho}' = \widetilde{\rho}$ for QEC decoders, QuantumSE.jl will call the Bitwuzla SMT solver [Niemetz and Preiner 2023] to prove it[4]. If the SMT solver is unable to prove it, the SMT solver will provide a counterexample, from which QuantumSE.jl will report that the program S is buggy and generate the test case.

To evaluate QuantumSE.jl, we consider the following research questions (RQs):

- **RQ1.** *Is QuantumSE.jl scalable at finding bugs in QEC programs over different kinds of QEC codes?*
- **RQ2.** *Can QuantumSE.jl outperform other tools?*
- **RQ3.** *What factors affect the performance of QuantumSE.jl?*

All our experiments are carried out on a desktop with Intel(R) Core(TM) i7-9700 CPU @3.00GHz and 16G of RAM, running Ubuntu 22.04.2 LTS.

8.1 RQ1: Finding Bugs in QEC Programs

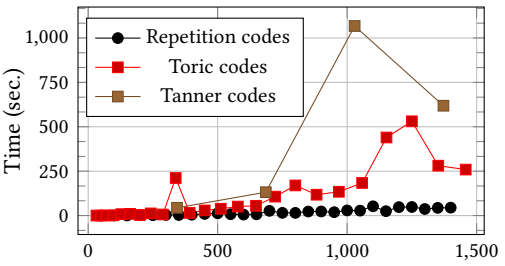
To address RQ1, we selected three representative QEC codes, i.e., quantum repetition codes, Kitaev's toric codes [Kitaev 2003, 1997], and quantum Tanner codes [Leverrier and Zémor 2022]. The repetition codes and toric codes belong to surface codes [Dennis et al. 2002], a variant of which was implemented in Google's recent QEC experiment [Acharya et al. 2023] (with 72 physical qubits). The quantum Tanner codes, which follow the recent major breakthrough [Panteleev and Kalachev 2022] of QEC codes, are pretty complicated. Fig. 5a shows one of the QEC programs evaluated in this experiment. It has a strange bug that it will fail to correct errors when X errors occur at qubits $q_1, q_2, \ldots, q_{\lfloor \frac{n-1}{2} \rfloor}$. A detailed description of the QEC codes and programs used in this experiment is available in Appendix A of the extended version [Fang and Ying 2023].

Results. Based on the construction of these QEC codes, we evaluated QuantumSE.jl on repetition codes with $50j$ physical qubits for $1 \leq j \leq 28$, toric codes with $2d^2$ physical qubits for $4 \leq d \leq 27$, and quantum Tanner codes with $343k$ physical qubits for $1 \leq k \leq 4$. Fig. 5b shows the running times for QuantumSE.jl to find bugs in the QEC programs for these different sizes of QEC codes. We can see that the running time does not tend to increase dramatically with the number of qubits. QuantumSE.jl can analyze QEC programs with over 1000 qubits in a short period of time, exceeding

[4]We choose Bitwuzla as it is a winner of the competition in the quantifier-free bit-vector logic category in SMT-COMP 2022 (see https://smt-comp.github.io/2022/results/qf-bitvec-single-query).

$CNOT\ q_1, q_2; s_1 \coloneqq \texttt{measure}\ q_2; CNOT\ q_1, q_2;$ // measure Z_1Z_2
$CNOT\ q_2, q_3; s_2 \coloneqq \texttt{measure}\ q_3; CNOT\ q_2, q_3;$ // measure Z_2Z_3
...
$CNOT\ q_{n-1}, q_n; s_{n-1} \coloneqq \texttt{measure}\ q_n; CNOT\ q_{n-1}, q_n;$ // measure $Z_{n-1}Z_n$
$CNOT\ q_n, q_1; s_n \coloneqq \texttt{measure}\ q_1; CNOT\ q_n, q_1;$ // measure Z_nZ_1
$r_1, r_2, \ldots, r_n \coloneqq \text{MWPM}(s_1, s_2, \ldots, s_n);$ // call MWPM
$X[r_1 == 1]\ q_1;$ // apply X if $r_1 = 1$
$X[r_2 == 1]\ q_2;$ // apply X if $r_2 = 1$
...
$X[r_n == 1]\ q_n;$ // apply X if $r_n = 1$
$X[r_1 * r_2 * \cdots * r_{\lfloor \frac{n-1}{2} \rfloor} == 1]\ q_1$ // a strange bug!

(a) A QEC program with a strange bug for quantum repetition codes. The quantum repetition code with n physical qubits has stabilizer checks $Z_1Z_2, Z_2Z_3, \ldots, Z_{n-1}Z_n$. After measuring these checks, we additionally measure the observable Z_nZ_1 for the convenience of calling MWPM.



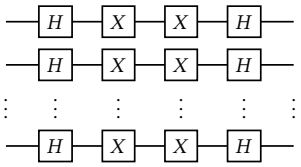
(b) Performance results of finding bugs in QEC programs by QuantumSE.jl. The runtime here is mainly consumed by the SMT solver, and its irregular trend is also due to performance anomalies of the solver in some special cases. See §8.3 for detailed analysis.

Fig. 5. An example of QEC programs evaluated in RQ1 and the performance results of RQ1.

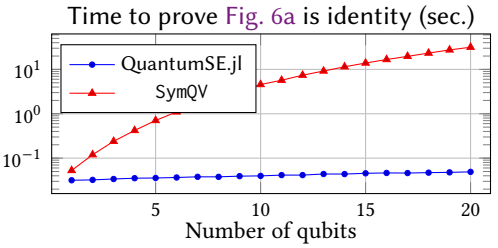
the current physical experiments that use about 280 qubits. Thus, we believe that QSE will be useful in future QEC experiments for debugging QEC programs.

8.2 RQ2: Comparing with Other Tools

Comparing with other SE tools. As we will discuss in related work (see §9.2), none of the existing work on SE can handle QEC programs. Nevertheless, since SymQV [Bauer-Marquart et al. 2023] provides an easy-to-use implementation, we compare QuantumSE.jl with it on a simple class of circuits that only contains H and X gates (see Fig. 6a). We use SymQV and QuantumSE.jl respectively to check whether the input states and output states of Fig. 6a are equivalent. Since SymQV uses Z3 as the SMT solver, we also use Z3 in this comparison.



(a) A simple class of quantum circuits that contains four layers of H, X, X, H gates, which are canceled out into identity.



(b) Performance results of SymQV and QuantumSE.jl.

Fig. 6. Comparison between SymQV and QuantumSE.jl. QuantumSE.jl's runtime is fast, while SymQV's is slower even though we set up SymQV with product state and overapproximation.

Results. The runtime of SymQV and QuantumSE.jl are presented in Fig. 6b. The main reason for the results in Fig. 6b is that QuantumSE.jl is based on stabilizers, while SymQV is based on quantum states, which has a much larger dimension that grows exponentially with the number of qubits. But the strange thing is that we have set up the product state approximation for SymQV, which should not take too long as in Fig. 6b. We believe there is room for optimization in implementing SymQV.

Comparing with state-of-the-art simulators. Several other tools can handle QEC programs (circuits), but since the dimension of the state space of qubits grows exponentially with the number of qubits, the only existing work that can efficiently handle QEC circuits with over 1000 qubits

are stabilizer-based simulators (see discussion in §9.1). However, finding bugs with the simulator requires considering all possible errors. For example, in the QEC program of Fig. 5a with $n = 1000$, there are $\binom{1000}{499} \approx 10^{299}$ possible locations of 499 X errors, but the bug only occurs when $q_1, q_2, \ldots, q_{499}$ have X errors, which is in one of these possibilities. *Even though the state-of-the-art stabilizer simulator, Stim [Gidney 2021], is really fast at sampling measurement results, it is unrealistic to sample one case out of* 10^{299} *possibilities even with a supercomputer.*

To give Stim a fair shake, we have added the ability to sample stabilizer circuits' measurement results to QuantumSE.jl[5] and compare it to Stim in terms of sampling functionality. A sample of a stabilizer circuit refer to the (random) measurement outcomes obtained by running the circuit once. For samples of a stabilizer circuit, Stim and QuantumSE.jl first initialize a sampler and then use it to generate samples rapidly. To conduct a comparison, we choose the benchmark of layered random interaction circuits used in Stim [Gidney 2021], where Stim outperformed popular simulators such as Qiskit's stabilizer method [Qiskit Community 2017], Cirq's Clifford simulator [Cirq Developers 2018], Aaronson and Gottesman [2004]'s `chp.c` and GraphSim [Anders and Briegel 2006].

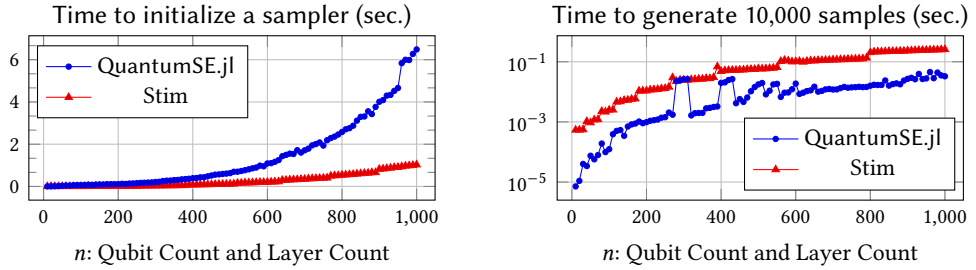


Fig. 7. Performance results of sampling layered random interaction circuits. Each circuit is made up of n qubits with n layers. Each layer randomly applies an H, S, and I gate to each qubit, then samples 10 pairing of the qubits to apply $CNOT$ gate, then samples 5% of the qubits to measure in the computational basis. At the end of the circuit, each qubit is measured in the computational basis.

Results. We present the comparisons of the time for QuantumSE.jl and Stim to initialize a sampler (i.e., the time to analyze the input circuit and create a sampler for generating the measurement results) and the time for QuantumSE.jl and Stim's samplers to generate 10,000 samples in Fig. 7. We can see that for $n > 600$, the sampler generated by QuantumSE.jl spends a shorter time than Stim's in generating samples, which means that *QuantumSE.jl can exhibit a faster sampling speed than the state-of-the-art stabilizer simulator*. We also notice that QuantumSE.jl spends more time than Stim to initialize the sampler. The main reason here is that Stim is well optimized in initializing the sampler, while our QuantumSE.jl is not specifically designed for it.

8.3 RQ3: Analyzing Performance Factors

To analyze the factors that affect the performance of QuantumSE.jl, we count the running time of QuantumSE.jl debugging QEC programs in three separate stages:

- **Init**: the initialization stage that prepares initial symbolic configurations for QSE. It takes time mainly to initialize the symbolic quantum states as in Theorem 6.6 and to insert symbolic errors as in Eq. (5).
- **QSE**: the quantum symbolic execution stage that performs symbolic execution of the quantum program. The time spent in this stage depends on how QuantumSE.jl maintains the symbolic configuration, especially its symbolic quantum state (symbolic stabilizer state).

[5]After symbolic execution, we only need to substitute concrete values for all symbols (with corresponding probabilities) in the symbolic expressions of measurement results to achieve sampling without having to go through the circuit again.

- **SMT**: the SMT solver stage that calls an SMT solver to solve the assertions constructed by QuantumSE.jl. The time spent in this stage depends on the performance of the solver.

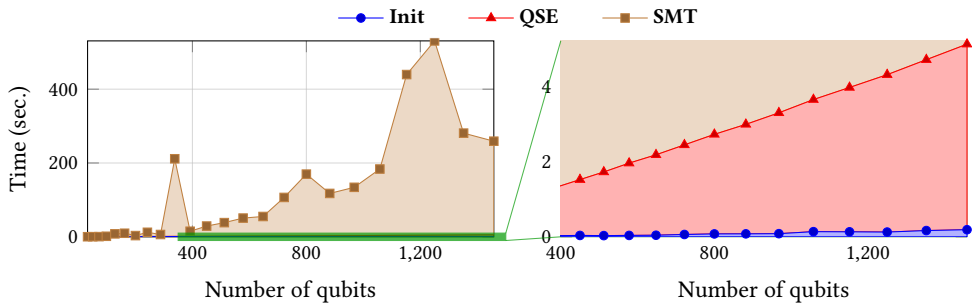


Fig. 8. The stack graph of time spent in three stages.

Results. We chose the QEC programs for Toric codes in RQ1 (§8.1) to analyze because of the nontrivial trend of Toric codes in the time curve (Fig. 5b) and enough number of data points available. We show the time spent by QuantumSE.jl in the three stages in a stack graph, Fig. 8, which allows us to see that the stage **SMT** is dominating the running time. We can see that the time used by the SMT solver does not increase regularly with the number of qubits. We find that this is due to anomalies in the performance of the SMT solver in some special cases[6]. For example, the adopted SMT solver Bitwuzla spent 210.3(s) in the case of 338 qubits; however, if we change the configuration of Bitwuzla, it can solve it with only 11(s). This raises an interesting issue regarding improving the combination of QSE and SMT solvers for further research. Additionally, we observe that the time share of stages **Init** and **QSE** is small and the trend of their time curves is smooth. Thus, these two parts are very scalable and do not restrict the performance of QuantumSE.jl.

9 RELATED WORK

9.1 Simulation-based Analysis for Quantum Computing

With the rapid development of quantum hardware, researchers have recently put a lot of effort into the simulation of quantum computation on classical computers, and with it comes a wealth of quantum softwares [Bergholm et al. 2022; Cirq Developers 2018; Luo et al. 2020; Qiskit Community 2017; Suzuki et al. 2021; Svore et al. 2018], which help us to test and debug quantum circuits and programs.

We first discuss the *stabilizer-based simulation* that is most relevant to our work. It is well-known that stabilizer circuits can be efficiently simulated [Aaronson and Gottesman 2004; Anders and Briegel 2006; Gidney 2021; Gottesman 1998; Krastanov 2019], where Gottesman [1998]'s tableau algorithm and Aaronson and Gottesman [2004]'s improved tableau algorithm play a significant role because of the tableau representation of stabilizers and destabilizers. Based on the tableau representation and algorithms, Rall et al. [2019] proposed the Pauli propagation that speeds up the rate of sampling Pauli noises, and it was adopted by Google's Stim [Gidney 2021], which was used in Google's recent QEC experiments [Acharya et al. 2023]. However, *even though Stim has excellent performance, it still cannot handle all possible errors for QEC programs as we discussed in §8.2.*

It is worth noting that, starting from the tableau representation again, Berent et al. [2022] proposed an SAT encoding for Clifford circuits and demonstrated the applicability of equivalence checking; Schneider et al. [2023] used a similar SAT encoding for Clifford circuit synthesis. However,

[6]We have tried many SMT solvers, including Z3, Z3++, Yices2, and cvc5. They also have similar problems, but the instances where the problem occurs are not the same instances that Bitwuzla encounters.

all of these works only dealt with quantum circuits without mid-circuit measurements and control flows, and thus cannot be directly used in debugging QEC programs considered in this paper. Additionally, Rand et al. [2021] uses a type system to describe the stabilizer formalism elegantly. This type system provides efficient verification of quantum circuits. However, it needs to deal with all possible errors for QEC programs on a case-by-case analysis like a simulator.

There are also other techniques for simulation of quantum computation, e.g., tensor networks-based simulation [Markov and Shi 2008; Orús 2019; Pan and Zhang 2022], (binary) decision diagrams-based simulation [Hong et al. 2022; Niemann et al. 2016; Sistla et al. 2024; Vinkhuijzen et al. 2023]. These simulation techniques are widely used in developing or testing quantum circuits and quantum programs [Burgholzer et al. 2023]. *But most works are limited to manipulating concrete data and do not introduce symbolic expressions like ours.*

9.2 Symbolic Techniques for Quantum Computing

Due to the difficulty of simulation and analysis of quantum computation on classical computers, symbolic techniques have already been employed to improve their efficiency and scalability. The current works can be roughly classified into the following two categories:

Symbolic simulation of quantum computation. Several work [Sistla et al. 2023, 2024; Tsai et al. 2021] used symbolic expressions or symbolic Boolean functions during the process of simulation of quantum computation to speed up the simulation time; for example, Tsai et al. [2021] built a series of Boolean formulas between binary decision diagrams for state evolution to replace matrix-vector multiplication. Furthermore, Chen et al. [2023] used the tree automata [Comon et al. 2008] to represent (set of) quantum states and introduced symbolic update formulae of tree automata for quantum gates. Huang et al. [2021] converts the variational quantum circuit into logical formulas, in which the parameters of the variational quantum circuit are temporarily symbolized. Then with concrete values assigned, logical formulas provide an efficient sampling of quantum circuits. *Although they make use of symbolic expressions, these works focused on simulation, and the idea of symbolic execution was not introduced there.*

Symbolic execution of quantum circuits. Carette et al. [2023] used the algebraic normal form of Boolean functions to perform the symbolic execution of Hardamard-Toffoli quantum circuits, which cannot express QEC. Another work close to ours is symQV [Bauer-Marquart et al. 2023], in which n-qubit quantum states are represented with 2^n symbolic complex numbers $|\psi\rangle \coloneqq (\alpha_1, \alpha_2, \ldots, \alpha_{2^n})$ or product state $|\psi\rangle \coloneqq \otimes_{j=1}^{n} |\psi_j\rangle$, where $|\psi_j\rangle$ is encoded by 4 symbols. Their experiments demonstrated their applicability to 24 qubits. *Although the target of symQV is the symbolic execution of quantum programs, it is actually designed for quantum circuits without control flows, and thus symQV cannot be directly used in the verification of QEC programs considered in this paper.* In addition, Giallar [Tao et al. 2022] introduced symbolic representation and execution for quantum circuits with 20 rewrite rules to check the equivalence of quantum circuits, which also lack support for control flows; Quartz [Xu et al. 2022] dealt with symbolic quantum circuits too, computing symbolic matrix representations of circuits to discover equivalent circuit transformations by SMT solvers. However, similar to Giallar, Quartz does not support control flows.

9.3 Verification and Analysis of Quantum Programs

There is a rich literature on the verification and analysis of quantum programs. We briefly discuss three categories of research relevant to our work.

Formal verification with program logic. Based on quantum predicates of observables [D'hondt and Panangaden 2006], Ying [2012] proposed the first sound and relatively complete quantum

Hoare logic (QHL) for quantum **while**-programs. To simplify the verification, Zhou et al. [2019] used projectors as predicates and proposed a variant of QHL. Since projectors may involve exponential complexity for QEC programs, several works suggested to use stabilizers as predicates and proposed corresponding variants of QHL [Sundaram et al. 2022; Wu et al. 2021b]. Their approaches share a similar idea with Rand et al. [2021]'s type system and still need to deal with all possible errors for QEC programs on a case-by-case verification. Beside quantum predicates of observables, Qbricks [Chareton et al. 2021] used parameterized path sum representations for quantum circuits and reasoned about the representation using quantifier-free predicate logic. It has good proof automation and deduction rules for reasoning on circuits, but currently lacks support for classical control flows.

Formal verification without program logic. The most representative work that does not use program logic is perhaps QWIRE [Paykin et al. 2017; Rand et al. 2018] and SQIR [Hietala et al. 2021a,b], both of which directly formalize the denotational semantics of quantum programs in terms of density matrices. In this way, it seems that they may also encounter exponential complexity, when applied for verification and analysis of QEC programs.

Assertion checking and debugging. In addition to formal verification, an interesting line of research is assertion checking for analyzing and debugging quantum programs. Some papers use dynamic assertions [Li et al. 2020; Liu et al. 2020] to detect bugs at run-time, which seems not suitable for QEC programs' correctness before deployment. Others use statistical assertions [Huang and Martonosi 2019], which may require repeated simulations and are thus inefficient for the case of a large number of qubits. To verify assertions more efficiently, Yu and Palsberg [2021] introduced a novel idea of abstract interpretation that abstracts concrete quantum states into the intersection of small projections, which are closely related to the definition of stabilizer. For instance, in their example, the abstract domain for GHZ is a stabilizer. However, their approach also requires a case-by-case check for all possible errors in QEC programs.

10 CONCLUSION AND FUTURE DIRECTIONS

In this paper, we presented a symbolic execution framework for quantum programs. Within this framework, we introduced symbolic stabilizer states, which facilitate the efficient analysis of QEC programs. We also developed a prototype tool based on this framework and demonstrated its effectiveness and efficiency. Issues for future research include:

(1) *Incorporating our QSE with the recent technique of probabilistic symbolic execution [Gehr et al. 2016; Susag et al. 2022]*: This will enable us to handle the analysis of random errors in QEC. Specifically, it can be used to analyze the performance of QEC programs against random errors when we have verified their correctness against adversarial errors.
(2) *Improving the combination of QSE and SMT solvers*: A key point here is to optimize the SMT solver for the specific assertions posed by QSE.
(3) *Exploring more applications of QSE and symbolic stabilizer states*: A possible application is the sampling task we have done in §8.2, promising to perform better with the existing engineering efforts. Another is to use symbolic stabilizer states for assertions, which may make existing work possible to handle all adversarial errors and all logical states in QEC without enumeration.

ACKNOWLEDGMENTS

We thank anonymous reviewers for helpful comments and suggestions that improved this paper. We are grateful to Craig Gidney for pointing out our previous inappropriate use of Stim. This work was partly supported by the National Key R&D Program of China under Grant No. 2023YFA1009403.

DATA AVAILABILITY STATEMENT

Our code is available at https://github.com/njuwfang/QuantumSE.jl.
An evaluated artifact [Fang and Ying 2024] is available at https://doi.org/10.5281/zenodo.10781381.

REFERENCES

Scott Aaronson and Daniel Gottesman. 2004. Improved simulation of stabilizer circuits. *Phys. Rev. A* 70 (Nov 2004), 052328. Issue 5. https://doi.org/10.1103/PhysRevA.70.052328

M. H. Abobeih, Y. Wang, J. Randall, S. J. H. Loenen, C. E. Bradley, M. Markham, D. J. Twitchen, B. M. Terhal, and T. H. Taminiau. 2022. Fault-tolerant operation of a logical qubit in a diamond quantum processor. *Nature* 606, 7916 (01 Jun 2022), 884–889. https://doi.org/10.1038/s41586-022-04819-6

Rajeev Acharya, Igor Aleiner, Richard Allen, Trond I Andersen, Markus Ansmann, Frank Arute, Kunal Arya, Abraham Asfaw, Juan Atalaya, et al. 2023. Suppressing quantum errors by scaling a surface code logical qubit. *Nature* 614, 7949 (01 Feb 2023), 676–681. https://doi.org/10.1038/s41586-022-05434-1

D. Aharonov and M. Ben-Or. 1997. Fault-Tolerant Quantum Computation with Constant Error. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing (STOC '97)*. Association for Computing Machinery, 176–188. https://doi.org/10.1145/258533.258579

Mihhail Aizatulin, Andrew D. Gordon, and Jan Jürjens. 2011. Extracting and Verifying Cryptographic Models from C Protocol Code by Symbolic Execution. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS '11)*. Association for Computing Machinery, 331–340. https://doi.org/10.1145/2046707.2046745

Simon Anders and Hans J. Briegel. 2006. Fast simulation of stabilizer circuits using a graph-state representation. *Phys. Rev. A* 73 (Feb 2006), 022334. Issue 2. https://doi.org/10.1103/PhysRevA.73.022334

Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574, 7779 (01 Oct 2019), 505–510. https://doi.org/10.1038/s41586-019-1666-5

Fabian Bauer-Marquart, Stefan Leue, and Christian Schilling. 2023. SymQV: Automated Symbolic Verification Of Quantum Programs. In *Formal Methods: 25th International Symposium, FM 2023*. Springer-Verlag, 181–198. https://doi.org/10.1007/978-3-031-27481-7_12

Lucas Berent, Lukas Burgholzer, and Robert Wille. 2022. Towards a SAT Encoding for Quantum Circuits: A Journey From Classical Circuits to Clifford Circuits and Beyond. In *25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022)*, Vol. 236. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 18:1–18:17. https://doi.org/10.4230/LIPIcs.SAT.2022.18

Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, Shahnawaz Ahmed, Vishnu Ajith, M Sohaib Alam, Guillermo Alonso-Linaje, B AkashNarayanan, et al. 2022. PennyLane: Automatic differentiation of hybrid quantum-classical computations. arXiv:1811.04968 [quant-ph]

Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. 2017. Julia: A Fresh Approach to Numerical Computing. *SIAM Rev.* 59, 1 (2017), 65–98. https://doi.org/10.1137/141000671

Dolev Bluvstein, Simon J Evered, Alexandra A Geim, Sophie H Li, Hengyun Zhou, Tom Manovitz, Sepehr Ebadi, Madelyn Cain, Marcin Kalinowski, et al. 2024. Logical quantum processor based on reconfigurable atom arrays. *Nature* 626, 7997 (2024), 58–65. https://doi.org/10.1038/s41586-023-06927-3

Lukas Burgholzer, Alexander Ploier, and Robert Wille. 2023. Tensor Networks or Decision Diagrams? Guidelines for Classical Quantum Circuit Simulation. arXiv:2302.06616 [quant-ph]

Cristian Cadar, Daniel Dunbar, and Dawson R. Engler. 2008. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs. In *8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008*. USENIX Association, 209–224. http://www.usenix.org/events/osdi08/tech/full_papers/cadar/cadar.pdf

Cristian Cadar and Koushik Sen. 2013. Symbolic Execution for Software Testing: Three Decades Later. *Commun. ACM* 56, 2 (feb 2013), 82–90. https://doi.org/10.1145/2408776.2408795

Jacques Carette, Gerardo Ortiz, and Amr Sabry. 2023. Symbolic Execution of Hadamard-Toffoli Quantum Circuits. In *Proceedings of the 2023 ACM SIGPLAN International Workshop on Partial Evaluation and Program Manipulation (PEPM 2023)*. Association for Computing Machinery, 14–26. https://doi.org/10.1145/3571786.3573018

Christophe Chareton, Sébastien Bardin, François Bobot, Valentin Perrelle, and Benoît Valiron. 2021. An Automated Deductive Verification Framework for Circuit-building Quantum Programs. In *Programming Languages and Systems: 30th European Symposium on Programming, ESOP 2021*. Springer International Publishing, 148–177. https://doi.org/10.1007/978-3-030-72019-3_6

Yu-Fang Chen, Kai-Min Chung, Ondřej Lengál, Jyun-Ao Lin, Wei-Lun Tsai, and Di-De Yen. 2023. An Automata-Based Framework for Verification and Bug Hunting in Quantum Circuits. *Proc. ACM Program. Lang.* 7, PLDI, Article 156 (jun 2023), 26 pages. https://doi.org/10.1145/3591270

Zijun Chen, Kevin J. Satzinger, Juan Atalaya, Alexander N. Korotkov, Andrew Dunsworth, Daniel Sank, Chris Quintana, Matt McEwen, Rami Barends, et al. 2021. Exponential suppression of bit or phase errors with cyclic error correction. *Nature* 595, 7867 (01 Jul 2021), 383–387. https://doi.org/10.1038/s41586-021-03588-y

Cirq Developers. 2018. Cirq. https://doi.org/10.5281/zenodo.4062499

Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. 2008. *Tree Automata Techniques and Applications*. 262 pages. https://inria.hal.science/hal-03367725

Andrew W. Cross, Lev S. Bishop, John A. Smolin, and Jay M. Gambetta. 2017. Open Quantum Assembly Language. arXiv:1707.03429 [quant-ph]

Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. 2002. Topological quantum memory. *J. Math. Phys.* 43, 9 (08 2002), 4452–4505. https://doi.org/10.1063/1.1499754

Ellie D'hondt and Prakash Panangaden. 2006. Quantum weakest preconditions. *Mathematical Structures in Computer Science* 16, 3 (2006), 429–451. https://doi.org/10.1017/S0960129506005251

Jack Edmonds. 1965. Paths, trees, and flowers. *Canadian Journal of mathematics* 17 (1965), 449–467. https://doi.org/10.4153/CJM-1965-045-4

Wang Fang and Mingsheng Ying. 2023. Symbolic Execution for Quantum Error Correction Programs (Extended Version). arXiv:2311.11313 [quant-ph]

Wang Fang and Mingsheng Ying. 2024. *Artifact: Symbolic Execution for Quantum Error Correction Programs*. Zenodo. https://doi.org/10.5281/zenodo.10781381

Gian Pietro Farina, Stephen Chong, and Marco Gaboardi. 2019. Relational Symbolic Execution. In *Proceedings of the 21st International Symposium on Principles and Practice of Declarative Programming (PPDP '19)*. Association for Computing Machinery, Article 10, 14 pages. https://doi.org/10.1145/3354166.3354175

Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. 2012. Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A* 86 (Sep 2012), 032324. Issue 3. https://doi.org/10.1103/PhysRevA.86.032324

Timon Gehr, Sasa Misailovic, and Martin T. Vechev. 2016. PSI: Exact Symbolic Inference for Probabilistic Programs. In *Computer Aided Verification - 28th International Conference, CAV 2016 (Lecture Notes in Computer Science, Vol. 9779)*. Springer, 62–83. https://doi.org/10.1007/978-3-319-41528-4_4

Craig Gidney. 2021. Stim: a fast stabilizer circuit simulator. *Quantum* 5 (July 2021), 497. https://doi.org/10.22331/q-2021-07-06-497

Patrice Godefroid, Nils Klarlund, and Koushik Sen. 2005. DART: Directed Automated Random Testing. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '05)*. Association for Computing Machinery, 213–223. https://doi.org/10.1145/1065010.1065036

Daniel Gottesman. 1997. *Stabilizer codes and quantum error correction*. Ph. D. Dissertation. California Institute of Technology. arXiv:quant-ph/9705052 [quant-ph]

Daniel Gottesman. 1998. The Heisenberg Representation of Quantum Computers. arXiv:quant-ph/9807006 [quant-ph]

Daniel Gottesman. 2014. Fault-Tolerant Quantum Computation with Constant Overhead. arXiv:1310.2984 [quant-ph]

Kesha Hietala, Robert Rand, Shih-Han Hung, Liyi Li, and Michael Hicks. 2021a. Proving Quantum Programs Correct. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. https://doi.org/10.4230/LIPICS.ITP.2021.21

Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. 2021b. A Verified Optimizer for Quantum Circuits. *Proc. ACM Program. Lang.* 5, POPL, Article 37 (jan 2021), 29 pages. https://doi.org/10.1145/3434318

Xin Hong, Xiangzhen Zhou, Sanjiang Li, Yuan Feng, and Mingsheng Ying. 2022. A Tensor Network Based Decision Diagram for Representation of Quantum Circuits. *ACM Trans. Des. Autom. Electron. Syst.* 27, 6, Article 60 (jun 2022), 30 pages. https://doi.org/10.1145/3514355

Yipeng Huang, Steven Holtzen, Todd Millstein, Guy Van den Broeck, and Margaret Martonosi. 2021. Logical Abstractions for Noisy Variational Quantum Algorithm Simulation. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21)*. Association for Computing Machinery, 456–472. https://doi.org/10.1145/3445814.3446750

Yipeng Huang and Margaret Martonosi. 2019. Statistical Assertions for Validating Patterns and Finding Bugs in Quantum Programs. In *Proceedings of the 46th International Symposium on Computer Architecture (ISCA '19)*. Association for Computing Machinery, 541–553. https://doi.org/10.1145/3307650.3322213

IBM. 2023. The hardware and software for the era of quantum utility is here. https://www.ibm.com/quantum/blog/quantum-roadmap-2033.

James C. King. 1976. Symbolic Execution and Program Testing. *Commun. ACM* 19, 7 (jul 1976), 385–394. https://doi.org/10.1145/360248.360252

A.Yu. Kitaev. 2003. Fault-tolerant quantum computation by anyons. *Annals of Physics* 303, 1 (2003), 2–30. https://doi.org/10.1016/S0003-4916(02)00018-0

A Yu Kitaev. 1997. Quantum error correction with imperfect gates. In *Quantum communication, computing, and measurement*. Springer, 181–188. https://doi.org/10.1007/978-1-4615-5923-8_19

Vladimir Kolmogorov. 2009. Blossom V: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation* 1, 1 (01 Jul 2009), 43–67. https://doi.org/10.1007/s12532-009-0002-8

Stefan Krastanov. 2019. https://quantumsavory.github.io/QuantumClifford.jl/stable/ Accessed on 2023-7-7.

Anthony Leverrier and Gilles Zémor. 2022. Quantum Tanner codes. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*. 872–883. https://doi.org/10.1109/FOCS54457.2022.00117

Gushu Li, Li Zhou, Nengkun Yu, Yufei Ding, Mingsheng Ying, and Yuan Xie. 2020. Projection-Based Runtime Assertions for Testing and Debugging Quantum Programs. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 150 (nov 2020), 29 pages. https://doi.org/10.1145/3428218

Ji Liu, Gregory T. Byrd, and Huiyang Zhou. 2020. Quantum Circuits for Dynamic Runtime Assertions in Quantum Computation. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20)*. Association for Computing Machinery, 1017–1030. https://doi.org/10.1145/3373376.3378488

Kasper Luckow, Corina S. Păsăreanu, Matthew B. Dwyer, Antonio Filieri, and Willem Visser. 2014. Exact and Approximate Probabilistic Symbolic Execution for Nondeterministic Programs. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE '14)*. Association for Computing Machinery, 575–586. https://doi.org/10.1145/2642937.2643011

Xiu-Zhe Luo, Jin-Guo Liu, Pan Zhang, and Lei Wang. 2020. Yao.jl: Extensible, Efficient Framework for Quantum Algorithm Design. *Quantum* 4 (Oct. 2020), 341. https://doi.org/10.22331/q-2020-10-11-341

Lars S. Madsen, Fabian Laudenbach, Mohsen Falamarzi. Askarani, Fabien Rortais, Trevor Vincent, Jacob F. F. Bulmer, Filippo M. Miatto, Leonhard Neuhaus, Lukas G. Helt, et al. 2022. Quantum computational advantage with a programmable photonic processor. *Nature* 606, 7912 (01 Jun 2022), 75–81. https://doi.org/10.1038/s41586-022-04725-x

Igor L. Markov and Yaoyun Shi. 2008. Simulating Quantum Computation by Contracting Tensor Networks. *SIAM J. Comput.* 38, 3 (2008), 963–981. https://doi.org/10.1137/050644756 arXiv:https://doi.org/10.1137/050644756

Michael A. Nielsen and Isaac L. Chuang. 2010. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press. https://doi.org/10.1017/CBO9780511976667

Philipp Niemann, Robert Wille, David Michael Miller, Mitchell A. Thornton, and Rolf Drechsler. 2016. QMDDs: Efficient Quantum Function Representation and Manipulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 1 (2016), 86–99. https://doi.org/10.1109/TCAD.2015.2459034

Aina Niemetz and Mathias Preiner. 2023. Bitwuzla. In *Computer Aided Verification - 35th International Conference, CAV 2023 (Lecture Notes in Computer Science, Vol. 13965)*. Springer, 3–17. https://doi.org/10.1007/978-3-031-37703-7_1

Román Orús. 2019. Tensor networks for complex quantum systems. *Nature Reviews Physics* 1, 9 (01 Sep 2019), 538–550. https://doi.org/10.1038/s42254-019-0086-7

Feng Pan and Pan Zhang. 2022. Simulation of Quantum Circuits Using the Big-Batch Tensor Network Method. *Phys. Rev. Lett.* 128 (Jan 2022), 030501. Issue 3. https://doi.org/10.1103/PhysRevLett.128.030501

Pavel Panteleev and Gleb Kalachev. 2022. Asymptotically Good Quantum and Locally Testable Classical LDPC Codes. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2022)*. Association for Computing Machinery, 375–388. https://doi.org/10.1145/3519935.3520017

Jennifer Paykin, Robert Rand, and Steve Zdancewic. 2017. QWIRE: A Core Language for Quantum Circuits. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL '17)*. Association for Computing Machinery, 846–858. https://doi.org/10.1145/3009837.3009894

Sebastian Poeplau and Aurélien Francillon. 2020. Symbolic execution with SymCC: Don't interpret, compile!. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 181–198. https://www.usenix.org/conference/usenixsecurity20/presentation/poeplau

Qiskit Community. 2017. Qiskit: An Open-Source Framework for Quantum Computing. https://doi.org/10.5281/zenodo.2562110

Patrick Rall, Daniel Liang, Jeremy Cook, and William Kretschmer. 2019. Simulation of qubit quantum circuits via Pauli propagation. *Phys. Rev. A* 99 (Jun 2019), 062337. Issue 6. https://doi.org/10.1103/PhysRevA.99.062337

Robert Rand, Jennifer Paykin, and Steve Zdancewic. 2018. QWIRE Practice: Formal Verification of Quantum Circuits in Coq. *Electronic Proceedings in Theoretical Computer Science* 266 (feb 2018), 119–132. https://doi.org/10.4204/eptcs.266.8

Robert Rand, Aarthi Sundaram, Kartik Singhal, and Brad Lackey. 2021. Gottesman Types for Quantum Programs. In *Proceedings of the 17th International Conference on Quantum Physics and Logic (QPL) (Electronic Proceedings in Theoretical Computer Science, Vol. 340)*. Open Publishing Association, 279–290. https://doi.org/10.4204/EPTCS.340.14

Sarah Schneider, Lukas Burgholzer, and Robert Wille. 2023. A SAT Encoding for Optimal Clifford Circuit Synthesis. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference* (Tokyo, Japan) *(ASPDAC '23)*. Association for Computing Machinery, New York, NY, USA, 190–195. https://doi.org/10.1145/3566097.3567929

P. W. Shor. 1996. Fault-tolerant quantum computation. In *Proceedings of 37th Conference on Foundations of Computer Science (Proceedings of 37th Conference on Foundations of Computer Science)*. 56–65. https://doi.org/10.1109/SFCS.1996.548464

Stephen F. Siegel, Anastasia Mironova, George S. Avrunin, and Lori A. Clarke. 2008. Combining Symbolic Execution with Model Checking to Verify Parallel Numerical Programs. *ACM Trans. Softw. Eng. Methodol.* 17, 2, Article 10 (may 2008), 34 pages. https://doi.org/10.1145/1348250.1348256

Meghana Sistla, Swarat Chaudhuri, and Thomas Reps. 2023. Symbolic quantum simulation with quasimodo. In *International Conference on Computer Aided Verification*. Springer, 213–225. https://doi.org/10.1007/978-3-031-37709-9_11

Meghana Aparna Sistla, Swarat Chaudhuri, and Thomas Reps. 2024. CFLOBDDs: Context-Free-Language Ordered Binary Decision Diagrams. *ACM Trans. Program. Lang. Syst.* (mar 2024). https://doi.org/10.1145/3651157 Just Accepted.

Aarthi Sundaram, Robert Rand, Kartik Singhal, and Brad Lackey. 2022. Hoare meets Heisenberg: A Lightweight Logic for Quantum Programs. https://ks.cs.uchicago.edu/publication/heisenberg/

Zachary Susag, Sumit Lahiri, Justin Hsu, and Subhajit Roy. 2022. Symbolic Execution for Randomized Programs. *Proc. ACM Program. Lang.* 6, OOPSLA2, Article 181 (oct 2022), 30 pages. https://doi.org/10.1145/3563344

Yasunari Suzuki, Yoshiaki Kawase, Yuya Masumura, Yuria Hiraga, Masahiro Nakadai, Jiabao Chen, Ken M. Nakanishi, Kosuke Mitarai, Ryosuke Imai, et al. 2021. Qulacs: a fast and versatile quantum circuit simulator for research purpose. *Quantum* 5 (Oct. 2021), 559. https://doi.org/10.22331/q-2021-10-06-559

Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. 2018. Q#: Enabling Scalable Quantum Computing and Development with a High-Level DSL. In *Proceedings of the Real World Domain Specific Languages Workshop 2018 (RWDSL2018)*. Association for Computing Machinery, Article 7, 10 pages. https://doi.org/10.1145/3183895.3183901

Runzhou Tao, Yunong Shi, Jianan Yao, Xupeng Li, Ali Javadi-Abhari, Andrew W. Cross, Frederic T. Chong, and Ronghui Gu. 2022. Giallar: Push-Button Verification for the Qiskit Quantum Compiler. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI 2022)*. Association for Computing Machinery, 641–656. https://doi.org/10.1145/3519939.3523431

Yuan-Hung Tsai, Jie-Hong R. Jiang, and Chiao-Shan Jhang. 2021. Bit-Slicing the Hilbert Space: Scaling Up Accurate Quantum Circuit Simulation. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 439–444. https://doi.org/10.1109/DAC18074.2021.9586191

Lieuwe Vinkhuijzen, Tim Coopmans, David Elkouss, Vedran Dunjko, and Alfons Laarman. 2023. LIMDD: A Decision Diagram for Simulation of Quantum Computing Including Stabilizer States. *Quantum* 7 (Sept. 2023), 1108. https://doi.org/10.22331/q-2023-09-11-1108

Anbang Wu, Gushu Li, Hezi Zhang, Gian Giacomo Guerreschi, Yuan Xie, and Yufei Ding. 2021b. QECV: Quantum Error Correction Verification. arXiv:2111.13728 [quant-ph]

Yulin Wu, Wan-Su Bao, Sirui Cao, Fusheng Chen, Ming-Cheng Chen, Xiawei Chen, Tung-Hsun Chung, Hui Deng, Yajie Du, et al. 2021a. Strong Quantum Computational Advantage Using a Superconducting Quantum Processor. *Phys. Rev. Lett.* 127 (Oct 2021), 180501. Issue 18. https://doi.org/10.1103/PhysRevLett.127.180501

Mingkuan Xu, Zikun Li, Oded Padon, Sina Lin, Jessica Pointing, Auguste Hirth, Henry Ma, Jens Palsberg, Alex Aiken, Umut A. Acar, and Zhihao Jia. 2022. Quartz: superoptimization of Quantum circuits. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI 2022)*. Association for Computing Machinery, 625–640. https://doi.org/10.1145/3519939.3523433

Mingsheng Ying. 2012. Floyd–hoare logic for quantum programs. *ACM Trans. Program. Lang. Syst.* 33, 6, Article 19 (jan 2012), 49 pages. https://doi.org/10.1145/2049706.2049708

Mingsheng Ying and Yuan Feng. 2011. A Flowchart Language for Quantum Programming. *IEEE Transactions on Software Engineering* 37, 4 (2011), 466–485. https://doi.org/10.1109/TSE.2010.94

Hengbiao Yu, Zhenbang Chen, Xianjin Fu, Ji Wang, Zhendong Su, Jun Sun, Chun Huang, and Wei Dong. 2020. Symbolic Verification of Message Passing Interface Programs. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE '20)*. Association for Computing Machinery, 1248–1260. https://doi.org/10.1145/3377811.3380419

Nengkun Yu and Jens Palsberg. 2021. Quantum Abstract Interpretation. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI 2021)*. Association for Computing Machinery, 542–558. https://doi.org/10.1145/3453483.3454061

Youwei Zhao, Yangsen Ye, He-Liang Huang, Yiming Zhang, Dachao Wu, Huijie Guan, Qingling Zhu, Zuolin Wei, Tan He, et al. 2022. Realization of an Error-Correcting Surface Code with Superconducting Qubits. *Phys. Rev. Lett.* 129 (Jul 2022), 030501. Issue 3. https://doi.org/10.1103/PhysRevLett.129.030501

Li Zhou, Nengkun Yu, and Mingsheng Ying. 2019. An Applied Quantum Hoare Logic. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2019)*. Association for Computing Machinery, 1149–1162. https://doi.org/10.1145/3314221.3314584

Received 2023-11-16; accepted 2024-03-31