Projection-Based Runtime Assertions for Testing and Debugging Quantum Programs

GUSHU LI*, University of California, Santa Barbara, USA
LI ZHOU*, Max Planck Institute for Security and Privacy, Germany
NENGKUN YU†, University of Technology, Sydney, Australia
YUFEI DING, University of California, Santa Barbara, USA
MINGSHENG YING, University of Technology, Sydney, Australia, Institute of Software, CAS, China, and Tsinghua University, China
YUAN XIE, University of California, Santa Barbara, USA

In this paper, we propose Proq, a runtime assertion scheme for testing and debugging quantum programs on a quantum computer. The predicates in Proq are represented by projections (or equivalently, closed subspaces of the state space), following Birkhoff-von Neumann quantum logic. The satisfaction of a projection by a quantum state can be directly checked upon a small number of projective measurements rather than a large number of repeated executions. On the theory side, we rigorously prove that checking projection-based assertions can help locate bugs or statistically assure that the semantic function of the tested program is close to what we expect, for both exact and approximate quantum programs. On the practice side, we consider hardware constraints and introduce several techniques to transform the assertions, making them directly executable on the measurement-restricted quantum computers. We also propose to achieve simplified assertion implementation using local projection technique with soundness guaranteed. We compare Proq with existing quantum program assertions and demonstrate the effectiveness and efficiency of Proq by its applications to assert two sophisticated quantum algorithms, the Harrow-Hassidim-Lloyd algorithm and Shor's algorithm.

CCS Concepts: • **Computer systems organization → Quantum computing**; • **Software and its engineering → Software testing and debugging**; • **Theory of computation → Assertions**.

Additional Key Words and Phrases: quantum computing, quantum programming, assertion, program testing

ACM Reference Format:
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 (November 2020), 29 pages. https://doi.org/10.1145/3428218

1 INTRODUCTION

Quantum computing is a promising computing paradigm with great potential in cryptography [Shor 1999], database [Grover 1996], linear systems [Harrow et al. 2009], chemistry simulation [Peruzzo

*The first two authors contribute equally.
†Corresponding author: Nengkun Yu

Authors' addresses: Gushu Li, University of California, Santa Barbara, USA, gushuli@ece.ucsb.edu; Li Zhou, Max Planck Institute for Security and Privacy, Germany, zhou31416@gmail.com; Nengkun Yu, University of Technology, Sydney, Australia, nengkunyu@gmail.com; Yufei Ding, University of California, Santa Barbara, USA, yufeiding@cs.ucsb.edu; Mingsheng Ying, University of Technology, Sydney, Australia , Institute of Software, CAS, China , Tsinghua University, China, Mingsheng.Ying@uts.edu.au; Yuan Xie, University of California, Santa Barbara, USA, yuanxie@ece.ucsb.edu.

2475-1421/2020/11-ART150
https://doi.org/10.1145/3428218

et al. 2014], etc. Several quantum program languages [Abhari et al. [n.d.]; Abraham et al. 2019; Google 2018; Green et al. 2013; Paykin et al. 2017; Rigetti Forest team 2019; Svore et al. 2018] have been published to write quantum programs for quantum computers. One of the key challenges that must be addressed during quantum program development is to compose correct quantum programs since it is easy for programmers living in the classical world to make mistakes in the counter-intuitive quantum programming. For example, Huang and Martonosi [Huang and Martonosi 2019a,b] reported a few bugs found in the example programs from the ScaffCC compiler project [JavadiAbhari et al. 2015]. Bugs have also been reported in IBM's OpenQASM project [IBM 2019] and Rigetti's PyQuil project [Rigetti 2019]. These erroneous quantum programs, written and reviewed by professional quantum computing experts, are sometimes even of very small size (with only 3 qubits)[1]. Such difficulty in writing correct quantum programs hinders practical quantum computing. Thus, effective and efficient quantum program debugging is naturally in urgent demand.

In this paper, we focus on runtime testing and debugging a quantum program on a quantum computer, and revisit *assertion*, one of the basic program testing and debugging approaches, in quantum computing. There have been two quantum program assertion designs in prior research. Huang and Martonosi proposed statistical assertions, which employed statistical tests on classical observations [Huang and Martonosi 2019b] to debug quantum programs. Motivated by indirect measurement and quantum error correction, Liu et al. proposed a runtime assertion [Liu et al. 2020], which introduces ancilla qubits to indirectly detect the system state. As early attempts towards quantum program testing and debugging, these studies suffer from the following drawbacks:

1) **Limited applicability with classical style predicates:** The properties of quantum program states can be much more complex than those in classical computing. Existing quantum assertions [Huang and Martonosi 2019b; Liu et al. 2020], which express the quantum program assertion predicates in a classical logic language, can only assert some simple quantum states of three special cases (detailed later in Section 5). A lot of complex intermediate program states cannot be tested by these assertions due to their limited expressive power. Hence, these assertions can only be injected at some special locations where the states are within the three supported types. Such restricted assertion types and injection locations will increase the difficulty in debugging as assertions may have to be injected far away from a bug.

2) **Inefficient assertion checking:** A general quantum state cannot be duplicated [Wootters and Zurek 1982], while the measurements, which are essential in assertions, usually only probe part of the state information and will destroy the tested state immediately. Thus, an assertion, together with the computation before it, must be repeated for a large number of times to achieve a precise estimation of the tested state in Huang and Martonosi's assertion design [Huang and Martonosi 2019b]. Another drawback of the destructive measurement is that the computation after an assertion will become meaningless. Even though multiple assertions can be injected at the same time, only one assertion could be inspected per execution, which will make the assertion checking more prolonged [Huang and Martonosi 2019b].

3) **Lacking theoretical foundations:** Different from a classical deterministic program, a quantum program has its intrinsic randomness and one execution may not cover all possible computations of even one specific input. Moreover, some quantum algorithms (e.g., Grover's search [Grover 1996], Quantum Phase Estimation [Nielsen and Chuang 2010], qPCA [Lloyd et al. 2014]) are designed to allow approximate program states, and the quantum program assertion checking itself is also probabilistic. Consequently, testing a quantum program usually requires multiple executions for one program configuration. It is important but rarely considered (to the best of our knowledge) what statistical information we can infer by testing those probabilistic quantum programs with

[1]We checked the issues raised in these projects' official GitHub repositories for this information.

assertions. Existing quantum program assertion studies [Huang and Martonosi 2019b; Liu et al. 2020], which mostly rely on empirical study, lack a rigorous theoretical foundation.

Potential and problem of projections: We observe that projection can be the key to address these issues due to its potential logical expressive power and unique mapping property. The logical expressive power of projection operators comes from the quantum logic by Birkhoff and von Neumann back in 1936 [Birkhoff and Von Neumann 1936]. The logical connectives (e.g., conjunction and disjunction) of projection operators can be defined by the set operations on their corresponding closed subspaces of a Hilbert space. Moreover, projections naturally match the projective measurement, which may not affect the measured state when the state is in one of its basis states [Li and Ying 2014]. However, only those projective measurements with a very limited set of projections can be directly implemented on a quantum computer due to the physical constraints on the measurement basis and measured qubit count, impeding the full utilization of the logical expressive power of projections.

To overcome all the problems mentioned above and fully exploit the potential of projections, we propose **Proq**, a projection-based runtime assertion for quantum programs. **First**, we employ projection operators to express the predicates in our runtime assertion. The logical expressive power of projection-based predicates allows us to assert much more types of states and enable more flexible assertion locations. **Second**, we define the semantics of our projection-based assertions by turning the projection-based predicates into corresponding projective measurements. Then the measurement in our assertion will not affect the tested state if the state satisfies the assertion predicate. This property leads to more efficient assertion checking and enables multi-assertion per execution. **Third**, we quantitatively show that after a sufficient number of testing executions with projection-based assertions, the semantics of the tested program can be guaranteed with a high confidence level. This result can serve as the theoretical foundation of quantum program testing with projection-based assertions. **Finally**, we consider the physical constraints on a quantum computer and introduce several transformation techniques, including *additional unitary transformation*, *combining projections*, and *using auxiliary qubits*, to make all projection-based assertions executable on a measurement-restricted quantum computer. We also propose *local projection*, which is a sound simplification of the original projections, to relax the constraints in the predicates for simplified assertion implementations.

The major contributions of this paper can be summarized as follows:

(1) We, first the time, propose to use projection operators to design runtime assertions that have strong logical expressive power and can be efficiently checked on a quantum computer.
(2) On the theory side, we prove that testing quantum programs with projection-based assertions is statistically effective in debugging or assuring the program semantics for both exact and approximate quantum programs.
(3) On the practice side, we propose several assertion transformation techniques to simplify the assertion implementation and make our assertions physically executable on a measurement-restricted quantum computer.
(4) Both theoretical analysis and experimental results show that our assertion outperforms existing quantum program assertions [Huang and Martonosi 2019b; Liu et al. 2020] with much stronger expressive power, more flexible assertion location, fewer executions, and lower implementation overhead.

2 PRELIMINARY

In this section, we introduce the necessary preliminary to help understand the proposed assertion scheme.

2.1 Quantum Computing

Quantum computing is based on quantum systems evolving under the law of quantum mechanics. The state space of a quantum system is a Hilbert space (denoted by $\mathcal{H}$), a complete complex vector space with inner product defined. A pure state of a quantum system is described by a unit vector $|\psi\rangle$ in its state space. When the exact state is unknown, but we know it could be in one of some pure states $|\psi_i\rangle$, with respective probabilities p_i, where $\sum_i p_i = 1$, a density operator ρ can be defined to represent such a mixed state with $\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i|$. A pure state is a special mixed state. Hence, in this paper, we adopt the more general density operator formulation most of the time since the state in a quantum program can be mixed upon measurement, an essential type of quantum operation.

For example, a qubit (the quantum counterpart of a bit in classical computing) has a two-dimensional state space $\mathcal{H}_2 = \{a|0\rangle + b|1\rangle\}$, where $a, b \in \mathbb{C}$ and $|0\rangle, |1\rangle$ are two computational basis states. Another commonly used basis is the Pauli-X basis, $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. For a quantum system with n qubits, the state space of the composite system is the tensor product of the state spaces of all its qubits: $\bigotimes_{i=1}^{n} \mathcal{H}_i = \mathcal{H}_{2^n}$. This paper only considers finite-dimensional quantum systems because realistic quantum computers only have a finite number of qubits.

There are mainly two types of operations performed on a quantum system, unitary transformation (also known as quantum gates) and quantum measurement.

Definition 2.1 (Unitary transformation). *A unitary transformation U on a quantum system in the finite-dimensional Hilbert space $\mathcal{H}$ is a linear operator satisfying $UU^\dagger = I_\mathcal{H}$, where $I_\mathcal{H}$ is the identity operator on $\mathcal{H}$.*

After a unitary transformation, a state vector $|\psi\rangle$ or a density operator ρ is changed to $U|\psi\rangle$ or $U\rho U^\dagger$, respectively. We list the definitions of the unitary transformations used in the rest of this paper as follows:

Single-qubit gates: H (Hadamard) $= \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$, $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

Two-qubit gates: CNOT(Controlled-NOT, Controlled-X), Swap:

$$\text{CNOT} = |0\rangle\langle 0| \otimes I_2 + |1\rangle\langle 1| \otimes X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \text{Swap} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Three-qubit gates: Toffoli, Fredkin (Controlled-Swap, CSwap):

$$\text{Toffoli} = |0\rangle\langle 0| \otimes I_4 + |1\rangle\langle 1| \otimes \text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\text{Fredkin} = |0\rangle\langle 0| \otimes I_4 + |1\rangle\langle 1| \otimes \text{Swap} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Definition 2.2 (Quantum measurement). *A quantum measurement on a quantum system in the Hilbert space $\mathcal{H}$ is a collection of linear operators $\{M_m\}$ satisfying $\sum_m M_m^\dagger M_m = I_\mathcal{H}$.*

After a quantum measurement on a pure state $|\psi\rangle$, an outcome m is returned with probability $p(m) = \langle\psi| M_m^\dagger M_m |\psi\rangle$ and then the state is changed to $|\psi_m\rangle = \frac{M_m|\psi\rangle}{\sqrt{p(m)}}$. Note that $\sum_m p(m) = 1$. For

a mixed state ρ, the probability that the outcome m occurs is $p(m) = tr(M_m^\dagger M_m \rho)$, and then the state will be changed to $\rho_m = \frac{M_m \rho M_m^\dagger}{p(m)}$.

2.2 Quantum Programming Language

For simplicity of presentation, this paper adopts the quantum **while**-language [Ying 2011] to describe the quantum algorithms. This language is purely quantum without classical variables but this selection will not affect the generality since the quantum **while**-language, which has been proved to be universal [Ying 2011], only keeps basic quantum computation elements that can be easily implemented by other quantum programming languages [Abhari et al. [n.d.]; Abraham et al. 2019; Google 2018; Green et al. 2013; Paykin et al. 2017; Rigetti Forest team 2019; Svore et al. 2018]. Thus, our assertion design and implementation based on this language can also be easily extended to other quantum programming languages

Definition 2.3 (Syntax [Ying 2011]). *The quantum **while**-programs are defined by the grammar:*

$$S ::= \text{skip} \mid S_1; S_2 \mid q := |0\rangle \mid \overline{q} := U[\overline{q}] \mid \text{if } (\square m \cdot M[\overline{q}] = m \rightarrow S_m) \text{ fi} \mid \text{while } M[\overline{q}] = 1 \text{ do } S \text{ od}$$

The language grammar is explained as follows. q represents a quantum variable while $\overline{q}$ means a quantum register, which consists of one or more variables with its corresponding Hilbert space denoted by $\mathcal{H}_{\overline{q}}$. $q := |0\rangle$ means that quantum variable q is initialized to be $|0\rangle$. $\overline{q} := U[\overline{q}]$ denotes that a unitary transformation U is applied to $\overline{q}$. Case statement if $\cdots$ fi means a quantum measurement M is performed on $\overline{q}$ to determine which subprogram S_m should be executed based on the measurement outcome m. The loop while $\cdots$ od means a measurement M with two possible outcomes $0, 1$ will determine whether the loop will terminate or the program will re-enter the loop body.

The *semantic function* of a quantum **while**-program S (denoted by $[\![S]\!]$) is a mapping from the program input state to its output state after executing program S. For example, $[\![S]\!](\rho)$ represents the output state of program S with input state ρ. A formal and comprehensive introduction to the semantics of quantum **while**-programs can be found in [Ying 2016].

2.3 Projection and Projective Measurement

One type of quantum measurement of particular interest is the projective measurement because all measurements that can be physically implemented on quantum computers are projective measurements. We first introduce projections and then define the projective measurement.

For each closed subspace X of $\mathcal{H}$, we can define a projection P_X. Note that every $|\psi\rangle \in \mathcal{H}$ ($|\psi\rangle$ does not have to be normalized) can be written as $|\psi\rangle = |\psi_X\rangle + |\psi_0\rangle$ with $|\psi_X\rangle \in X$ and $|\psi_0\rangle \in X^\perp$ (the orthocomplement of X).

Definition 2.4 (Projection). *The projection $P_X : \mathcal{H} \mapsto X$ is defined by*

$$P_X|\psi\rangle = |\psi_X\rangle.$$

for every $|\psi\rangle \in \mathcal{H}$.

In the rest of this paper, we denote P_X as P because there is a one-to-one correspondence between the closed subspaces of a Hilbert space and the projections in it. For simplicity, we do not distinguish a projection P from its corresponding subspace. Note that P is Hermitian ($P^\dagger = P$) and $P^2 = P$. If a pure state $|\psi\rangle$ (or a mixed state ρ) is in the corresponding subspace of a projection P, we have $P|\psi\rangle = |\psi\rangle$ ($P\rho P = \rho$). The **rank** of a projection P (denoted by $\text{rank } P$) is defined by the dimension of its corresponding subspace.

Definition 2.5 (Projective measurement). *A projective measurement M is a quantum measurement in which all the measurement operators are projections ($0_{\mathcal{H}}$ is the zero operator on $\mathcal{H}$):*

$$M = \{P_m\},\ \textit{where} \sum_m P_m = \mathcal{I}_{\mathcal{H}} \textit{ and } P_m P_n = \begin{cases} P_m & \textit{if } m = n, \\ 0_{\mathcal{H}} & \textit{otherwise.} \end{cases}$$

Note that if a state $|\psi\rangle$ (or ρ) is in the corresponding subspace of P_m, then a projective measurement with observed outcome m will not change the state since:

$$|\psi_m\rangle = \frac{P_m |\psi\rangle}{\sqrt{\langle\psi| P_m^\dagger P_m |\psi\rangle}} = \frac{|\psi\rangle}{\sqrt{\langle\psi|\psi\rangle}} = |\psi\rangle, \quad \left(\text{resp. } \rho_m = \frac{P_m \rho P_m^\dagger}{\mathrm{tr}\left(P_m^\dagger P_m \rho\right)} = \frac{\rho}{\mathrm{tr}(\rho)} = \rho \right)$$

2.4 Projection-Based Predicates and Quantum Logic

In addition to defining projective measurements, projection operators can also define the predicates in quantum programming. We introduce the definition of projection-based predicates.

Definition 2.6 (Projections-based predicates). *Suppose P is a projection operator on $\mathcal{H}$ and its corresponding closed subspace is X. A state ρ is said to satisfy a predicate P (written $\rho \models P$) if $\mathrm{supp}(\rho) \subseteq X$, where $\mathrm{supp}(\rho)$ is the subspace spanned by the eigenvectors of ρ with non-zero eigenvalues. Note that $\rho \models P \implies P\rho = \rho$.*

Some quantum algorithms (e.g., qPCA [Lloyd et al. 2014]) are not exact and their program states may only approximately satisfy a projection-based predicate. We first introduce two concepts, trace distance D and fidelity F, to evaluate the distance between two states. Then we define the approximate satisfactory of projection-based predicates.

Definition 2.7 (Trace distance of states). *For two states ρ and σ, the trace distance D, which measures the "distinguishability" of two quantum states, between ρ and σ is defined as*

$$D(\rho, \sigma) = \frac{1}{2} tr|\rho - \sigma|$$

where $tr|X| = tr\sqrt{X^\dagger X}$ and $\sqrt{X^\dagger X}$ refers to the positive square root which is unique because X is a density matrix which is Hermitian. Note that $0 \leq D(\rho, \sigma) \leq 1$ and $D(\rho, \sigma) = 0 \Leftrightarrow \rho = \sigma$. For two normalized states ρ and σ (pure states or density operators with trace 1), $D(\rho, \sigma) = 1 \Leftrightarrow \rho$ and σ are orthogonal. Trace distance is a metric and it satisfies the triangle inequality.

Definition 2.8 (Fidelity). *For two states ρ and σ, the fidelity F, which is not a metric but measures the "closeness" of two quantum states, between ρ and σ is defined as*

$$F(\rho, \sigma) = tr\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}}$$

where $\sqrt{\rho}$ is the unique positive square root given by the spectral theorem (the same with the square root in the above definition). For example, suppose the spectrum decomposition of ρ is $\sum_i p_i |\psi_i\rangle\langle\psi_i|$, then $\sqrt{\rho} = \sum_i \sqrt{p_i} |\psi_i\rangle\langle\psi_i|$ (we have $p_i \geq 0$ since a state ρ must be a positive semi-definite operator.). Note that $0 \leq F(\rho, \sigma) \leq 1$ and $F(\rho, \sigma) = 1 \Leftrightarrow \rho = \sigma$. $F(\rho, \sigma) = 0 \Leftrightarrow \rho$ and σ (may not be normalized) are orthogonal. Note that fidelity does not satisfy the triangle inequality. A frequently used metric induced by fidelity is the arccos of fidelity and it satisfies the triangle inequality.

Definition 2.9 (Approximate satisfactory of projection-based predicates). *A state ρ is said to approximately satisfy (projective) predicate P with error parameter ϵ, written $\rho \models_\epsilon P$ if there exists a σ with the same trace such that $\sigma \models P$ and $D(\rho, \sigma) \leq \epsilon$.*

In the rest of this paper, all predicates are projection-based predicates and we do not distinguish a predicate P, a projection P, and its corresponding closed subspace P. A quantum logic can be defined on the set of all closed subspaces of a Hilbert space [Birkhoff and Von Neumann 1936].

Definition 2.10 (Quantum logic on the projections [Birkhoff and Von Neumann 1936]). *Suppose $\mathcal{S}(\mathcal{H})$ is the set of all closed subspaces of Hilbert space $\mathcal{H}$. Then $(\mathcal{S}(\mathcal{H}), \wedge, \vee, ^{\perp})$ is an orthomodular lattice (or quantum logic). For any $P, Q \in \mathcal{S}(\mathcal{H})$, we define:*

$$P \wedge Q = P \cap Q,\ P \vee Q = \overline{\mathrm{span}(P \cup Q)},\ P^{\perp} = \{|\psi\rangle \in \mathcal{H} : \langle\psi| P |\psi\rangle = 0\}$$

and the notations are defined as follows. Suppose T is a set in $\mathcal{H}$. Then $\mathrm{span}(T)$ is the subspace spanned by T, and $\overline{T}$ is the closure of T. That is, in this quantum logic, the logic operations on the predicates are defined by the set operations on their corresponding subspaces.

2.5 Measurement-Restricted Quantum Computer

Although projective measurement has restricted all the measurement operators to be projection operators, most quantum computers which run on the well-adopted quantum circuit model [Nielsen and Chuang 2010] usually have more restrictions on the measurement.

First, they only support projective measurement in the **computational basis**. That is, only projective measurements with a specific set (which only contains all the computational basis states) of projection operators can be physically implemented. For example, such a projective measurement on n qubits can be described as $M = \{P_t\}$, where $P_t = |t\rangle\langle t|$ is the projection onto the 1-dimensional subspace spanned by the basis state $|t\rangle$, and t ranges over all n-bit strings; in particular, for a single qubit, this measurement is simply $M = \{P_0, P_1\}$ with $P_0 = |0\rangle\langle 0|$ and $P_1 = |1\rangle\langle 1|$.

Second, only projective measurements with projection operators of **special ranks** can be physically implemented. Suppose we have an n-qubit program with a 2^n-dimensional state space. After we measure one qubit, the state of that qubit will collapse to one of its basis states. The overall state space is reduced by half and becomes a 2^{n-1}-dimensional space. A projection P with $\mathrm{rank}\, P = 2^{n-1}$ can be implemented by measuring one qubit. If k qubits are measured, the remaining space will have 2^{n-k} dimensions, and projections with $\mathrm{rank}\, P = 2^{n-k}$ can be implemented by measuring k qubits. In reality, we can only measure an integer number of qubits but cannot measure a fraction number of qubits. For an n-qubit system, we can measure $\{1, 2, \cdots, n\}$ qubits so that only projections with $\mathrm{rank}\, P \in \{2^{n-1}, 2^{n-2}, \cdots, 1\}$ can be directly implemented.

3 PROJECTION-BASED ASSERTION: DESIGN AND THEORETICAL FOUNDATIONS

The goal of this paper is to provide a design of assertions which the programmers can insert in their quantum programs when testing and debugging their programs on a quantum computer. In particular, our design aims to achieve two objectives:

(1) The assertions should have strong logical expressive power and can be efficiently checked.
(2) The assertions should be executable on a quantum computer with restricted measurements.

In this section, we will focus on the first objective and introduce how to design quantum program assertions based on projection operators. We first discuss the reasons why projections are suitable for expressing predicates in a quantum program assertion. Then we formally define the syntax and semantics of a new projection-based assert statement. Finally, we rigorously formulate the theoretical foundations of program testing and debugging with projection-based assertions. We prove that running the assertion-injected program repeatedly can narrow down the potential location of a bug or assure that the semantics of the original program is close to what we expect.

3.1 Checking the Satisfaction of a Projection-Based Predicate

An assertion is a predicate at a point of a program. The key point of designing assertions for quantum programs is to first determine how to express predicates in the quantum scenario. Projection-based predicates has been used widely in static analysis and logic for quantum programming. For the first time, we employ projection-based predicates in runtime assertions for two reasons.

Strong logical expressive power: Figure 1 shows the orthomodular lattice based on all projections in a 2^n-dimensional Hilbert space and compares the logical expressive power of the predicates in existing assertions and the projections. All predicates expressed using a classical logical language in existing quantum program assertions [Huang and Martonosi 2019b; Liu et al. 2020] can be represented by very few elements of special ranks in this lattice (detailed discussion is in Section 5.1). But projections can naturally cover all elements in Figure 1. Therefore, projections have a much stronger expressive power compared with the classical logical language used in existing quantum assertions.

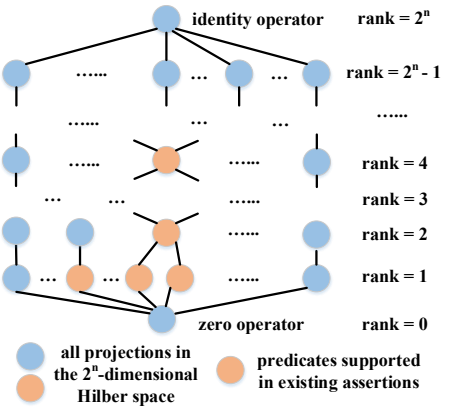

Fig. 1. Logical expressive power comparison

Efficient runtime checking: A quantum state ρ can be efficiently checked by a projection P because ρ will not be affected by the projective measurement with respect to P if it is in the subspace of P. We can construct a projective measurement $M = \{M_{\text{true}} = P, M_{\text{false}} = I - P\}$. When ρ is in the subspace of P, the outcome of this projective measurement is always "true" with probability of 1 and the state is still ρ. Then we know that ρ satisfies P without changing the state. When ρ is not in the subspace of P, which means that ρ does not satisfy P, the probability of outcome "true" or "false" in the constructed projective measurement is $tr(P\rho)$ or $1 - tr(P\rho)$, respectively. Suppose we perform such procedure k times, the probability that we do not observe any "false" outcome is $tr(P\rho)^k$. Since $tr(P\rho) < 1$, this probability approaches 0 very quickly when $tr(P\rho)$ is not close to 1 and we can conclude if ρ satisfies P with high certainty within very few executions. Moreover, even if the state ρ is not in the subspace of P, the projective measurement with outcome "true" will change the incorrect state ρ to a correct state that is in the subspace of P so that the following execution after the assertion is still valid.

When $1 - tr(P\rho) < \epsilon$ and ϵ is small, it is possible that we do not observe any 'false' outcome in very few executions because the probability of observing a 'false' outcome is small. In this situation, we have the following two cases. **First**, the program itself has some real bugs that makes a tested state very close to what we expect. Given $1 - tr(P\rho) < \epsilon$, the trace distance (Definition 2.7) of the tested state ρ and at least one desired state is bounded by a small number $\epsilon + \sqrt{\epsilon(1-\epsilon)}$ if we realize that $\frac{P\rho P}{tr(P\rho P)}$ is a desired state since it satisfies P (in Lemma 3.1). It is almost impossible to prove that no such bugs ever exist but such a bug is not severe since the final state of the program will also be close to the expected final state. This is because the trace distance is contractive under trace-nonincreasing quantum operations (the semantic function of any quantum program), i.e., $D([\![S]\!](\rho), [\![S]\!](\sigma)) \leq D(\rho, \sigma)$ where $[\![S]\!]$ is the semantic function of program S and D is the trace distance. Therefore, the trace distance between the final state of the tested program and the expected final state is also bounded by the small number $\epsilon + \sqrt{\epsilon(1-\epsilon)}$. Moreover, we have checked and confirmed that all types of bugs reported by Huang and Martonosi [Huang and Martonosi 2019a] (the only systematic report about bugs in real quantum programs to the best of our knowledge) can make $tr(P\rho)$ significantly smaller than 1. Therefore, checking a projection-based predicate

is effective for these known quantum program bugs. **Second**, the program itself is not an exact quantum program and its correct program states are supposed to only approximately satisfy the predicates. We will prove that projection-based assertions can still test and debug such approximate quantum programs later in Section 3.4.

3.2 Assertion Statement: Syntax and Semantics

We have demonstrated the advantages of using projections as predicates. Now we add a new runtime assertion statement to the quantum **while**-language grammar.

Definition 3.1 (**Syntax** of the assertion). *The **syntax** of the quantum assertion is defined as:*

$$\mathrm{assert}(\overline{q}; P)$$

where $\overline{q} = q_1, ..., q_n$ is a collection of quantum variables and P is a projection in the state space $\mathcal{H}_{\overline{q}}$.

As the original quantum **while**-language is already universal, we define the semantics of the new assertion statement using the quantum **while**-language. An auxiliary notation abort is employed to denote that the program terminates immediately and reports the termination location. The formal semantics of abort is $[\![\mathrm{abort}]\!](\rho) = 0_{\mathcal{H}}$ for all any input state ρ [Ying 2016]. Intuitively, this definition means we do not have any quantum state after abort.

Definition 3.2 (**Semantics**). *The **semantics** of the new assertion statement is defined as*

$$\begin{aligned}\mathrm{assert}(\overline{q}; P) \equiv \mathrm{if}\ M_P[\overline{q}] = m_0 &\rightarrow \mathrm{skip}\\ \square \qquad\quad\ \ m_1 &\rightarrow \mathrm{abort}\\ \mathrm{fi}\end{aligned}$$

where $M_P = \{M_{m_0} = P, M_{m_1} = I_{\mathcal{H}_{\overline{q}}} - P\}$.

The semantics of the assertion statement is explained as follows: We construct a projective measurement $M_P = \{M_{m_0} = P, M_{m_1} = I_{\mathcal{H}_{\overline{q}}} - P\}$ based on the projection operator P in the assertion. We apply this measurement of the corresponding qubit collection $\overline{q}$. If the measurement result is m_0, which means that the tested state is in the closed subspace of P, then we continue the execution of program without doing anything because the tested state satisfies the predicate in the assertion. If the measurement result is m_1, which means the tested state is not in the closed subspace of P, the program will terminate and report the termination location. Then we can know that the state at this location does not satisfy the corresponding predicate. Here the semantics of abort is slightly different from the original one because we need to report the termination location.

3.3 Statistical Effectiveness of Testing and Debugging with Projection-Based Assertions

As with classical program testing, quantum program testing can show the presence of bugs, lowering the risking of remaining bugs, but cannot assure the behavior of all possible computation. One testing execution cannot even check the program behavior thoroughly for one input due to the intrinsic randomness of quantum systems. Therefore, multiple executions are required to test a quantum program with one input. In this section, we show that, for a program with projection-based assertions and one specific input, running it repeatedly for enough times can locate bugs or statistically assure the behavior of the program under the specific input with high confidence.

We consider a quantum program S. When the programmers try to test a program with assertions, multiple assertions could be injected so that a potential bug could be revealed as early as possible. Suppose we insert l assertions whose predicates are $P_1, P_2, \ldots, P_l$ (P_l is the predicate for the final state). We define that a bug-free standard program S_{std} is a program that can satisfy all the predicates

throughout the program. We will show that after running the program with assertion inserted for a couple of times, we can locate the incorrect program segment if an error message occurs or conclude that output of the tested program S and the standard program S_{std} (under a specific input ρ) is close. We first formally define a debugging scheme for a quantum program.

Definition 3.3. *A debugging scheme for S is a new program S' with assertions being added between consecutive subprograms S_i and S_{i+1}:*

$$S' \equiv S_1;\ \mathrm{assert}(\overline{q}_1; P_1);\ S_2;\ \mathrm{assert}(\overline{q}_2; P_2);\ \cdots;\ S_{l-1};\ \mathrm{assert}(\overline{q}_{l-1}; P_{l-1});\ S_l;\ \mathrm{assert}(\overline{q}_l; P_l)$$

where $\overline{q}_i$ is the collection of quantum variables and P_i is a projection on $\mathcal{H}_{\overline{q}_i}$ for all $0 < i \leq l$.

In this debugging scheme, assertions are injected after every statement while this may not be necessary in practice. The assertion injection is flexible, and the programmers can inject assertions only on those locations where they hope to have assertions.

Now we discuss the statistical properties of this debugging scheme. A program segment S_i is considered to be correct if its output satisfies the predicate P_i when its input satisfied P_{i-1} as specified by the assertions. We show that running the program S' (defined in Definition 3.3) with assertions injected could effectively check the program by proving that the tested program S and a standard program S_{std} will have a similar semantic function under the tested input state. A quantitative and formal description of the effectiveness of our debugging scheme is illustrated by the following theorem.

Theorem 3.1 (Effectiveness of debugging scheme). *Suppose we repeatedly execute S' (with l assertions) with input ρ and collect all the error messages.*

(1) *If an error message occurs in $\mathrm{assert}(\overline{q}_i; P_i)$, then subprogram S_i is not correct, i.e., with the input satisfying precondition P_{i-1}, after executing S_i, the output can violate postcondition P_i.*
(2) *If no error message is reported after executing S' for k times ($k \gg l^2$), program S is close to the bug-free standard program; more precisely, with confidence level 95%,*
   (a) *the confidence interval of $\min_{S_{\mathrm{std}}} D\left([\![S]\!](\rho), [\![S_{\mathrm{std}}]\!](\rho)\right)$ is $\left[0, \frac{0.9l+\sqrt{l}}{\sqrt{k}}\right]$,*
   (b) *the confidence interval of $\max_{S_{\mathrm{std}}} F\left([\![S]\!](\rho), [\![S_{\mathrm{std}}]\!](\rho)\right)$ is $\left[\cos\frac{0.9l+\sqrt{l}}{\sqrt{k}}, 1\right]$,*
   where the minimum (maximum) is taken over all bug-free standard programs S_{std} that satisfy all assertions with input ρ. Here D is the trace distance (Definition 2.7) and F is the fidelity (Definition 2.8).

Moreover, within one testing execution, if the program s_m is not correct but $\mathrm{assert}(\overline{q}_m; P_m)$ is passed, then follow-up assertion $\mathrm{assert}(\overline{q}_{m+1}; P_{m+1})$ is still effective in checking the program S_{m+1}.

By Theorem 3.1, we conclude that we can use projection-based assertions to test a quantum program and find the locations of potential bugs with the proposed debugging scheme. When an error message occurs in $\mathrm{assert}(\overline{q}_i; P_i)$, we can know that there is at least one bug in the program segment S_i. Although we could not directly know how the bug happens nor repair a bug, our approach can help with debugging in practice, by narrowing down the potential location of a bug from the entire program to one specific program segment. After applying the proposed debugging scheme, programmers can manually investigate the target program segment to finally find the bug more quickly without searching in the entire program. If we could not have any error message after running the assertion checking program S' for a sufficiently large number of times, we can conclude that the semantics of the original program S for the tested input is at least close to what we expected (specified by the assertions) with high confidence. In the proposed theorem, we require $k \gg l^2$ because we hope to achieve the confidence level 95%. In practice, the number of test executions can be reduced with a lower confidence level.

Only one input tested: It can be noticed that only one input is tested when using the proposed debugging scheme in Theorem 3.1. However, in classical program testing, we usually prepare a large number of testing cases to increase the testing thoroughness. Here we argue that considering one input is already useful in testing many quantum programs because the input information of many practical quantum algorithms (e.g., Shor's algorithm [Shor 1999], Grover algorithm [Grover 1996], VQE algorithm [Peruzzo et al. 2014], HHL algorithm [Harrow et al. 2009]) are only encoded in the operations and the input state is always a trivial state $|00\cdots00\rangle$. Consequently, we do not need to check different inputs when testing these quantum algorithms. Checking for one specific input $\rho = |00\cdots00\rangle\langle 00\cdots00|$ will be sufficient.

3.4 Testing and Debugging Approximate Quantum Programs

We have shown that projection-based assertions can be used to check exact quantum programs but there are also other quantum algorithms (e.g., qPCA [Lloyd et al. 2014], Grover's search [Grover 1996], Quantum Phase Estimation [Nielsen and Chuang 2010]) of which the correct program states sometimes only approximately satisfy a projection. We generalize Theorem 3.1 by adding error parameters on all the program segments to represent the approximation throughout the program, and prove that we can still locate bugs or conclude about the semantics of the tested program with high confidence by checking projection-based assertions.

We first study how much a state ρ is changed after a projective measurement by proving a special case of the gentle measurement lemma [Winter 1999] with projections. The result is slightly stronger than the original one [Winter 1999] under the constraint of projection.

Lemma 3.1 (Gentle measurement with projections). *For projection P and density operator ρ, if $\mathrm{tr}(P\rho) \geq 1-\epsilon$, then we have*

(1) $D\left(\rho, \frac{P\rho P}{\mathrm{tr}(P\rho P)}\right) \leq \epsilon + \sqrt{\epsilon(1-\epsilon)}$, *$D$ is the trace distance (Definition 2.7).*

(2) $F\left(\rho, \frac{P\rho P}{\mathrm{tr}(P\rho P)}\right) \geq \sqrt{1-\epsilon}$, *$F$ is the fidelity (Definition 2.8).*

Suppose a state ρ satisfies P with error ϵ, then $\mathrm{tr}(P\rho) \geq 1-\epsilon$ which ensures that, applying the projective measurement $M_P = \{M_{\mathrm{true}} = P,\ M_{\mathrm{false}} = I - P\}$, we have the outcome "true" with probability at least $1-\epsilon$. Moreover, if the outcome is "true" and ϵ is small, the post-measurement state $\frac{P\rho P}{\mathrm{tr}(P\rho P)}$ is close to the original state ρ in the sense that their trace distance is at most $\epsilon + \sqrt{\epsilon(1-\epsilon)}$.

Consider a program $S = S_1; S_2; \cdots; S_l$ with l inserted assertions $\mathrm{assert}(\overline{q}_m, P_m)$ after each segments S_m for $1 \leq m \leq l$. Unlike the exact algorithms, here each program segment S_m is considered to be correct if its input satisfies P_{m-1}, then its output approximately satisfies P_m with error parameter ϵ_m. The following theorem states that the debugging scheme defined in Definition 3.3 is still effective for approximate quantum programs.

Theorem 3.2 (Effectiveness of debugging approximate quantum programs). *Assume that all ϵ_m are small ($\epsilon_m \ll 1$). Execute S' for k times ($k \gg l^2$) with input ρ, and we count k_m for the occurrence of error message for assertion $\mathrm{assert}(\overline{q}_m, P_m)$.*

(1) *The 95% confidence interval of real ϵ_m is $[w_m^-, w_m^+]$. Thus, with confidence 95%, if $\epsilon_m < w_m^-$, S_m is incorrect; and if $\epsilon_m > w_m^+$, we conclude S_m is correct. Here, w_m^-, w_m^+ and w_m^c are $B\left(\alpha, k_m + 1, k - \sum_{i=1}^{m} k_i\right)$ with $\alpha = 0.025, 0.975$ and 0.5 respectively, where $B(P, A, B)$ is the Pth quantile from a beta distribution with shape parameters A and B.*

(2) *If no segment appears to be incorrect, i.e., all $\epsilon_m \geq w_m^-$, then after executing the original program S with input ρ, the output state σ approximately satisfies P_l with error parameter δ, i.e., $\sigma \models_\delta P_l$, where $\delta = \sum_{m=1}^{l} \sqrt{w_m^c} + \sqrt{\sum_{m=1}^{l} (\sqrt{w_m^+} - \sqrt{w_m^c})^2}$.*

With this theorem, we can test and debug approximate quantum programs by counting the number of occurrences of the error messages from different assertions. If the observed assertion checking failure frequency is significantly higher or lower than the expected error parameter of a program segment, we can conclude that this program segment is correct or incorrect with high confidence. If all program segments appear to be correct, we can conclude that the final output of the original program approximately satisfies the last predicate within a bounded error parameter.

3.5 An Example of Using the Effectiveness Theorems

We give an example to illustrate using Theorem 3.1 and 3.2 in practical debugging. Suppose a bug-free standard program S has two qubits p, q:

$$S \equiv p := Z[p]; p := R_y(\pi/2)[p]; p, q := \mathrm{CNOT}[p, q]$$

where $R_y(\theta)$ is the rotation about the Y-axis, i.e.,

$$R_y(\theta) = \begin{bmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{bmatrix}.$$

When the input is $|00\rangle_{pq}$, the program produces a Bell state $|\Phi^+\rangle_{pq} = \frac{|00\rangle_{pq}+|11\rangle_{pq}}{\sqrt{2}}$. Now we consider a real program written by a careless programmer:

$$S_{\mathrm{real}} \equiv p := Z[p]; p := R_y(1.7)[p]; p, q := \mathrm{CNOT}[p, q]$$

which can be decomposed into two segments, $S_{\mathrm{real},1} \equiv p := Z[p]; p := R_y(1.7)[p]$, $S_{\mathrm{real},2} \equiv p, q := \mathrm{CNOT}[p, q]$. The careless programmer understands the program correctly and knows that if the input is $|00\rangle_{pq}$, the state after the first two unitary transformations on p should be $|+\rangle_q = \frac{|0\rangle_q+|1\rangle_q}{\sqrt{2}}$ and the final state should be Bell state. Thus he adds two assertions to S_{real}:

$$S'_{\mathrm{real}} \equiv p := Z[p]; p := R_y(1.7)[p]; \mathrm{assert}(p; P_1); p, q := \mathrm{CNOT}[p, q]; \mathrm{assert}(p, q; P_2)$$

where $P_1 = |+\rangle\langle+|$ and $P_2 = |\Phi^+\rangle\langle\Phi^+|$.

Theoretically it can be proved that S'_{real} is close to but not equal to the bug free program in the sense that:

$$D([\![S]\!](|00\rangle_{pq}), [\![S_{\mathrm{real}}]\!](|00\rangle_{pq})) = 0.065, \quad F([\![S]\!](|00\rangle_{pq}), [\![S_{\mathrm{real}}]\!](|00\rangle_{pq})) = 0.9979$$

Note that the final state is a pure state and thus all bug-free standard programs are equivalent to S when input is $|00\rangle_{pq}$.

We then consider three different testing cases. In the first two cases the programmer thinks the program should be accurate and in the last case the program is considered to be approximate.

Case 1. The programmer executes S'_{real} for 1000 times and surprisingly no error is reported. What can he learn from the result? By Theorem 3.1 (2) with $k = 1000$ and $l = 2$, we have: with confidence level 95%,

(1) the confidence interval of $D([\![S]\!](|00\rangle_{pq}), [\![S_{\mathrm{real}}]\!](|00\rangle_{pq}))$ is $[0, 0.101]$;
(2) the confidence interval of $F([\![S]\!](|00\rangle_{pq}), [\![S_{\mathrm{real}}]\!](|00\rangle_{pq}))$ is $[0.995, 1]$;

We can see that the real trace distance 0.065 and fidelity 0.9979 are indeed in the corresponding intervals $[0, 0.101]$ and $[0.995, 1]$, respectively.

Case 2. The programmer executes S'_{real} for 10,000 times and 37 errors are reported by the first assertion $\mathrm{assert}(p; P_1)$ but no error is reported by the second assertion. Now what can he conclude?

(1) By Theorem 3.1 (1), the first segment $S_{\mathrm{real},1} \equiv p := Z[p]; p := R_y(1.7)[p]$ is not correct; there must be some bugs;

(2) By Theorem 3.1 (2), the second segment $S_{\mathrm{real},2} \equiv p, q := \mathrm{CNOT}[p, q]$ is very likely to be true in the sense that: there exists a bug free standard program $S_1; S_2$ with two segments S_1 and S_2 such that: with confidence 95% ($k = 9963, l = 1$)
   (a) the confidence interval of $D(\llbracket S_1; S_2 \rrbracket(|00\rangle_{pq}), \llbracket S_1; S_{\mathrm{real},2} \rrbracket(|00\rangle_{pq}))$ is $[0, 0.019]$;
   (b) the confidence interval of $F(\llbracket S_1; S_2 \rrbracket(|00\rangle_{pq}), \llbracket S_1; S_{\mathrm{real},2} \rrbracket(|00\rangle_{pq}))$ is $[0.99982, 1]$;
   In fact, segment $S_{\mathrm{real},2}$ is exactly correct.

Case 3. Now, the programmer thinks that the program is approximate and a small error is acceptable. In detail, for both segments $S_{\mathrm{real},1}$ and $S_{\mathrm{real},2}$, he selects the same acceptable error parameters $\epsilon_1 = \epsilon_2 = 0.01$.
Fact: A straightforward calculation gives the *real value* of $\epsilon_{1,\mathrm{real}} = 0.0042$ and $\epsilon_{2,\mathrm{real}} = 0$, and the output $\llbracket S_{\mathrm{real}} \rrbracket(|00\rangle_{pq})$ approximately satisfies P_2 with error parameter 0.065.

Consider the execution results in the **case 2** above. According to Theorem 3.2 (1), we first calculate the 95% confidence intervals of real ϵ_1 and ϵ_2 are $[w_1^-, w_1^+]$ and $[w_2^-, w_2^+]$ where parameters are:

$$w_1^- = 0.0027, \quad w_1^+ = 0.0051, \quad w_1^c = 0.0038, \quad w_2^- = 0.00000, \quad w_2^+ = 0.00037, \quad w_2^c = 0.00007$$

Obviously, $\epsilon_1 > w_1^+$ and $\epsilon_2 > w_2^+$ and thus with confidence 95%, both of the segments are acceptable. Now, by Theorem 3.2 (2), we further know that the output $\llbracket S_{\mathrm{real}} \rrbracket(|00\rangle_{pq})$ approximately satisfies P_2 with error parameter $\delta = 0.0845 > 0.065$. All of these confidence intervals and parameters given by our theorems are consistent with the Fact.

4 TRANSFORMATION TECHNIQUES FOR IMPLEMENTATION ON QUANTUM COMPUTERS

In the previous section, we have illustrated how to test and debug a quantum program with the proposed projection-based assertions and proved its effectiveness. However, there exists a gap that makes the assertions not directly executable on a real quantum computer. There are two reasons for this incompatibility as explained in the following:

(1) **Limited measurement basis:** Not all projective measurements are supported on a quantum computer and only projective measurement that lie in the computational basis can be physically implemented directly with today's quantum computing underlying technologies (in Section 2.5). But there is no restriction on the projection operator P in the assertions so that P could be arbitrary projection operator in the Hilbert space. For example, $P = |+\rangle\langle+| = \frac{1}{2}(|0\rangle + |1\rangle)(\langle 0| + \langle 1|)$ is on a basis of $\{|+\rangle, |-\rangle\}$. These assertions with projections not in the computational basis cannot be directly executed on a real quantum computer.
(2) **Dimension mismatch:** A projective measurement, which is already in the computational basis, may still not be executable because the number of dimensions of its corresponding subspace cannot be directly implemented by measuring an integer number of qubits. For an n-qubit system, only projections with $\mathrm{rank}\, P \in \{2^{n-1}, 2^{n-2}, \cdots, 1\}$ can be directly implemented (in Section 2.5). But the rank of the projection in an assertion can be any integer between 0 and 2^n. For example, a projection in a 2-qubit system can be $P = |00\rangle\langle 00| + |01\rangle\langle 01| + |11\rangle\langle 11|$. An assertion with such projection cannot be directly implemented because $\mathrm{rank}\, P = 3$ and $\mathrm{rank}\, P \notin \{2, 1\}$.

In this section, we introduce several transformation techniques to overcome these two obstacles. The basic idea is to use the conjunction of projections and auxiliary qubit to convert the target assertion into some new assertions without dimension mismatch. Then some additional unitary transformations are introduced to rotate the basis in the projective measurements. These transformation techniques can be employed to compile the assertions and make a quantum program with projection-based assertions executable on a measurement-restricted real quantum computer.

4.1 Additional Unitary Transformation

We first resolve the limited measurement basis problem without considering the dimension mismatch problem. Suppose the assertion $\mathrm{assert}(\overline{q}; P)$ we hope to implement is over n qubits, that is, $\overline{q} = q_1, q_2, \cdots, q_n$, each of q_i is a single qubit variable. We assume that $\mathrm{rank}\, P = 2^m$ for some integer m with $0 \leq m \leq n$ so there is no dimension mismatch problem.

Proposition 4.1. *For projection P with $\mathrm{rank}\, P = 2^m$, there exists a unitary transformation U_P such that (here $I_{q_i} = I_{\mathcal{H}_{q_i}}$):*

$$U_P P U_P^\dagger = Q_{q_1} \otimes Q_{q_2} \otimes \cdots \otimes Q_{q_n} = \bigotimes_{i=1}^{n} Q_{q_i} \triangleq Q_P,$$

where $Q_{q_i} \in \{|0\rangle_{q_i}\langle 0|, |1\rangle_{q_i}\langle 1|, I_{q_i}\}$ for each $1 \leq i \leq n$. U_P and Q_P can be obtained immediately after we diagonalize the projection P.

We call the pair (U_P, Q_P) an **i**mplementation in the **c**omputational **b**asis (ICB for short) of $\mathrm{assert}(\overline{q}; P)$. ICB is not unique in general. According to this proposition, we have the following procedure to implement $\mathrm{assert}(\overline{q}; P)$:

(1) Apply U_P on $\overline{q}$;
(2) Check Q_P in the following steps: For each $1 \leq i \leq n$, if $Q_{q_i} = |0\rangle_{q_i}\langle 0|$ or $|1\rangle_{q_i}\langle 1|$, then measure q_i in the computational basis to see whether the outcome k is consistent with Q_{q_i}; that is, $Q_{q_i} = |k\rangle_{q_i}\langle k|$. If all outcomes are consistent, go ahead; otherwise, we terminate the program with an error message;
(3) Apply $U_P^\dagger$ on $\overline{q}$.

The transformation for $\mathrm{assert}(\overline{q}; P)$ with ICB (U_P, Q_P) when $\mathrm{rank}\, P = 2^m$ is:

$$\mathrm{assert}(\overline{q}; P) \equiv\ \overline{q} := U_P[\overline{q}];\ \mathrm{assert}(\overline{q}; Q_P);\ \overline{q} := U_P^\dagger[\overline{q}]$$

Since Q_P is now a projection in the computational basis, $\mathrm{assert}(\overline{q}; Q_P)$ can be executed by Definition 3.2 and the projective measurement constructed by Q_P is executable.

Example 4.1. *Given a two-qubit register $\overline{q} = q_1, q_2$, if we want to test whether it is in the Bell state (maximally entangled state) $|\Phi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, we can use the assertion $\mathrm{assert}(\overline{q}; P = |\Phi\rangle\langle\Phi|)$. We apply proposition 4.1 and diagonalize the projection P.*

$$P = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)\frac{1}{\sqrt{2}}(\langle 00| + \langle 11|) = \begin{bmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \end{bmatrix}$$

$$U_P P U_P^\dagger = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & 0 & \frac{1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & 0 & 0 & \frac{-1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \end{bmatrix} \cdot P \cdot \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} & 0 & \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 0 & \frac{-1}{\sqrt{2}} & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = Q_P$$

The generated diagonal matrix Q_P is actually $|0\rangle_{q_1}\langle 0| \otimes |0\rangle_{q_2}\langle 0|$ and the unitary U_P can be implemented with first an CNOT *gate and then a H gate. Therefore, we have:*

$$H[q_1]\mathrm{CNOT}[q_1, q_2] \cdot P \cdot \mathrm{CNOT}[q_1, q_2]H[q_1] = |0\rangle_{q_1}\langle 0| \otimes |0\rangle_{q_2}\langle 0|$$

we can first apply CNOT *gate on $\overline{q}$ and H gate on q_1, then measure q_1 and q_2 in the computational basis. If both outcomes are "0", we apply H on q_1 and* CNOT *on $\overline{q}$ again to recover the state; otherwise, we terminate the program and report that the state is* not *Bell state $|\Phi\rangle$.*

Unitary generation: The generated unitary may not be the exact inverse of the preceding operations. Suppose the desired state is $|\psi\rangle$, the only requirement for the unitary is $U : |\psi\rangle \mapsto |0\rangle$, and the output of U under input states other than $|\psi\rangle$ does not matter. This may allow simpler implementations of U. In general, it is not clear whether the generated U will be simpler or more complex when decomposing the U into basic single- and two-qubit gates. We demonstrate an in-principle unitary generation process but the actual implementation can be further optimized with techniques like tensor network, decision diagram, symbolic execution, etc. The scalability will be determined by the optimization on the matrix calculation and storage. This is left as future work.

4.2 Combining Assertions

The first transformation technique solves the measurement basis issue but does not consider the dimension mismatch issue, which will be addressed by the next two techniques. We first consider an assertion $\mathrm{assert}(\overline{q}; P)$ in which the projection P has $\mathrm{rank}\, P \leq 2^{n-1}$ and $\mathrm{rank}\, P \neq 2^m$ with some integer m. We have the following proposition to decompose this assertion into multiple sub-assertions that do not have dimension mismatch issues.

Proposition 4.2. *For projection P with $\mathrm{rank}\, P \leq 2^{n-1}$, there exist projections $P_1, P_2, \cdots, P_l$ satisfying $\mathrm{rank}\, P_i = 2^{n_i}$ for all $1 \leq i \leq l$ where $n_i \in \mathbb{N}$, such that $P = P_1 \cap P_2 \cap \cdots \cap P_l$.*

Essentially, this way works for our scheme because conjunction can be defined in Birkhoff-von Neumann quantum logic. Theoretically, $l = 2$ is sufficient; but in practice, a larger l may allow us to choose simpler P_i for each $i \leq l$.

Using the above proposition, to implement $\mathrm{assert}(\overline{q}; P)$, we may sequentially apply $\mathrm{assert}(\overline{q}; P_1)$, $\mathrm{assert}(\overline{q}; P_2)$, $\cdots$, $\mathrm{assert}(\overline{q}; P_l)$. Suppose (U_{P_i}, Q_{P_i}) is an ICB of $\mathrm{assert}(\overline{q}; P_i)$ for $1 \leq i \leq l$, we have the following scheme to implement $\mathrm{assert}(\overline{q}; P)$:

(1) Set counter $i = 1$;
(2) If $i = 1$, apply U_{P_1}; else if $i = l$, apply $U_{P_l}^\dagger$ and return; otherwise, apply $U_{P_{i-1}}^\dagger U_{P_i}$;
(3) Check Q_{P_i}; $i := i + 1$; go to step (2).

The transformation for $\mathrm{assert}(\overline{q}; P)$ when $\mathrm{rank}\, P \leq 2^{n-1}$ is:

$$\mathrm{assert}(\overline{q}; P) \equiv \mathrm{assert}(\overline{q}; P_1);\ \mathrm{assert}(\overline{q}; P_2);\ \ldots\ldots;\ \mathrm{assert}(\overline{q}; P_l)$$

where $\mathrm{rank}\, P_i = 2^{n_i}$ and $P = P_1 \cap P_2 \cap \cdots \cap P_l$. There are no dimension mismatch issues for these sub-assertions and they can be further transformed with Proposition 4.1.

Example 4.2. *Given register $\overline{q} = q_1, q_2, q_3$, how to implement $\mathrm{assert}(\overline{q}; P)$ where $P = |00\rangle_{q_1q_2}\langle 00| \otimes I_{q_3} + |111\rangle_{q_1q_2q_3}\langle 111|$? We first observe that $P = P_1 \cap P_2$ where*

$$P_1 = (|00\rangle_{q_1q_2}\langle 00| + |11\rangle_{q_1q_2}\langle 11|) \otimes I_{q_3},$$
$$P_2 = |00\rangle_{q_1q_2}\langle 00| \otimes I_{q_3} + |100\rangle_{q_1q_2q_3}\langle 100| + |111\rangle_{q_1q_2q_3}\langle 111|.$$

with following properties:

$$\mathrm{CNOT}[q_1, q_2] \cdot P_1 \cdot \mathrm{CNOT}[q_1, q_2] = I_{q_1} \otimes |0\rangle_{q_2}\langle 0| \otimes I_{q_3}$$
$$\mathrm{Toffoli}[q_1, q_3, q_2] \cdot P_2 \cdot \mathrm{Toffoli}[q_1, q_3, q_2] = I_{q_1} \otimes |0\rangle_{q_2}\langle 0| \otimes I_{q_3}.$$

Therefore, we can implement $\mathrm{assert}(\overline{q}; P)$ by:

- *Apply $\mathrm{CNOT}[q_1, q_2]$;*
- *Measure q_2 and check if the outcome is "0"; if not, terminate and report the error message;*
- *Apply $\mathrm{CNOT}[q_1, q_2]$ and then $\mathrm{Toffoli}[q_1, q_3, q_2]$;*
- *Measure q_2 and check if the outcome is "0"; if not, terminate and report the error message;*
- *Apply $\mathrm{Toffoli}[q_1, q_3, q_2]$.*

4.3 Auxiliary Qubits

The previous two techniques can transform projections with $\operatorname{rank} P \leq 2^{n-1}$ but those projections with $\operatorname{rank} P > 2^{n-1}$ remain unresolved. This case cannot be handled with the conjunction of a group of sub-assertions directly because logic conjunction can only result in a subspace with fewer dimensions (compared with the original subspaces of the projections in the sub-assertions). The possible subspace of a projection in an n-qubit system has at most 2^{n-1} dimensions since we have to measure at least one qubit. As a result, we cannot use logic conjunction to construct a projection with $\operatorname{rank} P > 2^{n-1}$. The logic disjunction of projections with small ranks can create a subspace of larger size but it is not suitable for assertion design. As discussed at the beginning of Section 3, it is expected that a correct state is not changed during the assertion checking. But if a state ρ at the tested program location is in a space of a large size, applying a projective measurement with a small subspace may destroy the tested state when the tested state is not in the small subspace, leading to inefficient assertion checking.

We propose the third technique, introducing auxiliary qubits, to tackle this problem. Actually, one auxiliary qubit is already sufficient. Suppose we have an n-qubit program with a 2^n-dimensional state space. If we add one additional qubit into this system, the system now has $n+1$ qubits with a 2^{n+1}-dimensional state space. This new qubit is not in the original quantum program so it is not involved in any assertions for the program. A projection P with $2^{n-1} < \operatorname{rank} P \leq 2^n$ can thus be implemented in the new 2^{n+1}-dimensional space using the previous two transformation techniques. One auxiliary qubit is sufficient because the projection P is originally in a 2^n-dimensional space and we always have $\operatorname{rank} P \leq 2^n$.

The transformation for $\operatorname{assert}(\overline{q}; P)$ when $\operatorname{rank} P > 2^{n-1}$ is:

$$\operatorname{assert}(\overline{q}; P) \equiv a := |0\rangle;\ \operatorname{assert}(a, \overline{q}; |0\rangle_a\langle 0| \otimes P)$$

where a is the new auxiliary qubit. Noting that $\operatorname{rank}(|0\rangle_a\langle 0| \otimes P) = \operatorname{rank} P \leq 2^n$.

Example 4.3. *Given register $\overline{q} = q_1, q_2$, we aim to implement $\operatorname{assert}(\overline{q}; P)$ where $P = |0\rangle_{q_1}\langle 0| \otimes I_{q_2} + |11\rangle_{q_1 q_2}\langle 11|$.*

We may have the decomposition $|0\rangle_a\langle 0| \otimes P = P_0 \cap P_1$, where

$$P_0 = |0\rangle_a\langle 0| \otimes I_{\overline{q}},\ P_1 = |00\rangle_{aq_1}\langle 00| \otimes I_{q_2} + |011\rangle_{aq_1q_2}\langle 011| + |100\rangle_{aq_1q_2}\langle 100|,$$

and P_1 can be implemented with one additional unitary transformation:

$$\operatorname{Fredkin}[q_2, a, q_1] \cdot P_1 \cdot \operatorname{Fredkin}[q_2, a, q_1] = I_a \otimes |0\rangle_{q_1}\langle 0| \otimes I_{q_2}.$$

where the Fredkin *gate is defined in Section 2.1.*

Note that P_0 automatically holds since the auxiliary qubit a is already initialized to $|0\rangle$, we only need to execute:

- *Introduce auxiliary qubit a, initialize it to $|0\rangle$;*
- *Apply* $\operatorname{Fredkin}[q_2, a, q_1]$*;*
- *Measure q_1 and check if the outcome is "0"; if not, terminate and report the error message;*
- *Apply* $\operatorname{Fredkin}[q_2, a, q_1]$*; free the auxiliary qubit a.*

4.4 Local Projection: Trading Checking Accuracy for Implementation Efficiency

As shown in the three transformation techniques, we need to manipulate the projection operators and some unitary transformations to implement an assertion. These transformations can be easily automated when n is small or the tested state is not fully entangled (which means we can deal with them part by part directly). For projections over multiple qubits, it is possible that the qubits are highly entangled. Asserting such entangled states accurately requires non-trivial efforts to find the unitary transformations and we need to manipulate operators of size 2^n for an n-qubit system in

the worst case, which makes it hard to fully automate the transformations on a classical computer when n is large. Such scalability issue widely exists in quantum computing research that requires automation on a classical computer, e.g., simulation [Chen et al. 2018], compiler optimization and its verification [Hietala et al. 2019; Shi et al. 2019], formal verification of quantum circuits [Paykin et al. 2017; Rand et al. 2018].

In our runtime projection-based assertion checking, we propose **local projection** technique to mitigate this scalability problem (not fully resolve it) by designing assertions that only manipulate and observe part of a large system without affecting a highly entangled state over multiple qubits. These assertions, which are only applied on a smaller number of qubits, could always be automated easily with simplified implementations but the assertion checking constraints are also relaxed. This approach is inspired by the quantum state tomography via local measurements [Chen et al. 2012; Linden et al. 2002; Xin et al. 2017], a common approach in quantum information science.

We first introduce the notion of partial trace to describe the state (operator) of a subsystem. Let $\overline{q}_1$ and $\overline{q}_2$ be two disjoint registers with corresponding state Hilbert space $\mathcal{H}_{\overline{q}_1}$ and $\mathcal{H}_{\overline{q}_2}$, respectively. The partial trace over $\mathcal{H}_{\overline{q}_1}$ is a mapping $\mathrm{tr}_{\overline{q}_1}(\cdot)$ from operators on $\mathcal{H}_{\overline{q}_1} \otimes \mathcal{H}_{\overline{q}_2}$ to operators in $\mathcal{H}_{\overline{q}_2}$ defined by: $\mathrm{tr}_{\overline{q}_1}(|\phi_1\rangle_{\overline{q}_1}\langle\psi_1| \otimes |\phi_2\rangle_{\overline{q}_2}\langle\psi_2|) = \langle\psi_1|\phi_1\rangle \cdot |\phi_2\rangle_{\overline{q}_2}\langle\psi_2|$ for all $|\phi_1\rangle, |\psi_1\rangle \in \mathcal{H}_{\overline{q}_1}$ and $|\phi_2\rangle, |\psi_2\rangle \in \mathcal{H}_{\overline{q}_2}$ together with linearity. The partial trace $\mathrm{tr}_{\overline{q}_2}(\cdot)$ over $\mathcal{H}_{\overline{q}_2}$ can be defined dually. Then, the local projection is defined as follows:

Definition 4.1 (Local projection). *Given* $\mathrm{assert}(\overline{q}; P)$*, a local projection* $P_{\overline{q}'}$ *over* $\overline{q}' \subseteq \overline{q}$ *is defined as:*

$$P_{\overline{q}'} = \mathrm{supp}\left(\mathrm{tr}_{\overline{q}\setminus\overline{q}'}(P)\right).$$

Proposition 4.3 (**Soundness** of local projection). *For any* $\rho \models P$*, we have* $\rho \models P_{\overline{q}'} \otimes I_{\overline{q}\setminus\overline{q}'}$.

This simplified assertion with $P_{\overline{q}'}$ will lose some checking accuracy because some states not in P may be included in $P_{\overline{q}'}$, allowing false positives. However, by taking the partial trace, we are able to focus on the subsystem of $\overline{q}'$. The implementation of $\mathrm{assert}(\overline{q}'; P_{\overline{q}'})$ can partially test whether the state satisfies P. Moreover, the number of qubits in $\overline{q}'$ is smaller, and we only need to manipulate small-size operators when implementing $\mathrm{assert}(\overline{q}'; P_{\overline{q}'})$. We have the following implementation strategy which is essentially a trade-off between assertion implementation efficiency and checking accuracy:

- Find a sequence of local projection $P_{\overline{q}_1}, P_{\overline{q}_2}, \cdots, P_{\overline{q}_l}$ of $\mathrm{assert}(\overline{q}; P)$;
- Instead of implementing the original $\mathrm{assert}(\overline{q}; P)$, we sequentially apply $\mathrm{assert}(\overline{q}_1; P_{\overline{q}_1})$, $\mathrm{assert}(\overline{q}_2; P_{\overline{q}_2}), \cdots, \mathrm{assert}(\overline{q}_l; P_{\overline{q}_l})$.

Example 4.4. *Given register* $\overline{q} = q_1, q_2, q_3, q_4$*, we want to check if the state is the superposition of the following states:*

$$|\psi_1\rangle = |+\rangle_{q_1}|111\rangle_{q_2q_3q_4}, \quad |\psi_2\rangle = |000\rangle_{q_1q_2q_3}|-\rangle_{q_4}, \quad |\psi_3\rangle = \frac{1}{\sqrt{2}}|0\rangle_{q_1}\left(|00\rangle_{q_2q_3} + |11\rangle_{q_2q_3}\right)|1\rangle_{q_4}.$$

To accomplish this, we may apply the assertion $\mathrm{assert}(\overline{q}; P)$ *with* $P = \mathrm{supp}\left(\sum_{i=1}^{3} |\psi_i\rangle\langle\psi_i|\right)$*. However, projection* P *is highly entangled which prevents efficient implementation. But if we only observe part of the system, we will the following local projections:*

$$\begin{aligned}
P_{q_1q_2} &= \mathrm{tr}_{q_3q_4}(P) = |0\rangle_{q_1}\langle 0| \otimes I_{q_2} + |11\rangle_{q_1q_2}\langle 11|,\\
P_{q_2q_3} &= \mathrm{tr}_{q_1q_4}(P) = |00\rangle_{q_2q_3}\langle 00| + |11\rangle_{q_2q_3}\langle 11|,\\
P_{q_3q_4} &= \mathrm{tr}_{q_1q_2}(P) = |00\rangle_{q_3q_4}\langle 00| + |11\rangle_{q_3q_4}\langle 11|.
\end{aligned}$$

To avoid implementing $\mathrm{assert}(\overline{q}, P)$ *directly, we may use* $\mathrm{assert}(q_1, q_2;\ P_{q_1q_2})$, $\mathrm{assert}(q_2, q_3;\ P_{q_2q_3})$, *and* $\mathrm{assert}(q_3, q_4; P_{q_3q_4})$ *instead. Though these assertions do not fully characterize the required property, their implementation requires only relatively low cost, i.e., each of them only acts on two qubits.*

In the next example, we show that a local projection may detect some bugs but not all of them.

Example 4.5. *Consider the following program with three qubits* p, q, r*:*

$$S \equiv p := H[p]; q := H[q]; r := H[r]; p, q := \mathrm{CZ}[p, q]; p, r := \mathrm{CZ}[p, r]; q := H[q]; r := H[r]$$

The program S *produces a GHZ state* $\frac{|000\rangle_{pqr}+|111\rangle_{pqr}}{\sqrt{2}}$ *if the input state is* $|000\rangle_{pqr}$*. Suppose a full description of GHZ state involving three qubits is somewhat difficult to implement due to complexity; instead, we choose the assertion* $\mathrm{assert}(p, q; P)$ *inserted at the end of the program where projection* P *is the local projection of GHZ state, i.e.,*

$$P = |00\rangle_{pq}\langle 00| + |11\rangle_{pq}\langle 11| = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

which can be implemented by 1). Apply CNOT on p, q*; 2). Measure qubit* q *and check if the measurement output is 0; 3). Apply CNOT on* p, q*. However, it is not difficult to realize that* $\mathrm{assert}(p, q; P)$ *is not a perfect description of GHZ state; for example, the following *bug* program* S_{bug} *where the final unitary transformation* $r := H[r]$ *is missing:*

$$S_{\mathrm{bug}} \equiv p := H[p]; q := H[q]; r := H[r]; p, q := \mathrm{CZ}[p, q]; p, r := \mathrm{CZ}[p, r]; q := H[q]$$

also passes $\mathrm{assert}(p, q; P)$ *with input* $|000\rangle_{pqr}$*. However, the program* S_{bug} *in fact produces:*

$$\frac{|00\rangle_{pq}|+\rangle_r + |11\rangle_{pq}|-\rangle_r}{\sqrt{2}}$$

if the input is $|000\rangle_{pqr}$*. On the other hand, consider another *bug* program* S'_{bug} *where the unitary transformation* $q := H[q]$ *is missing:*

$$S'_{\mathrm{bug}} \equiv p := H[p]; q := H[q]; r := H[r]; p, q := \mathrm{CZ}[p, q]; p, r := \mathrm{CZ}[p, r]; r := H[r]$$

Then it can be shown that if the input state is $|000\rangle_{pqr}$, $\mathrm{assert}(p, q; P)$ *is not passed and we are able to conclude that* S'_{bug} *is not the desired program using Theorem 3.1.*

4.5 Summary

To the best of our knowledge, the three transformations constitute the first working flow to implement an arbitrary projective measurement on measurement-restricted quantum computers. A complete flow to make an assertion $\mathrm{assert}(\overline{q}; P)$ (on n qubits) executable is summarized as follows:

(1) If $\mathrm{rank}\, P > 2^{n-1}$, initialize one auxiliary qubit a, let $n := n+1$ and $P := |0\rangle_a\langle 0| \otimes P$ (Section 4.3);
(2) If $\mathrm{rank}\, P \notin \{2^{n-1}, 2^{n-2}, \cdots, 1\}$, find a group of sub-assertions (Section 4.2);
(3) Apply unitary transformations to implement the assertion or sub-assertions (Section4.1).

The three transformations cover all possible cases for projections with different ranks and basis. Therefore, all projection-based assertions can finally be executed on a quantum computer. The local projection technique can be applied when an assertion is hard to be implemented (automatically). Whether to use local projection is optional.

5 OVERALL COMPARISON

In this section, we will have an overall comparison among Proq and two other quantum program assertions in terms of assertion coverage (i.e., the expressive power of the predicates, the assertion locations) and debugging overhead (i.e., the number of executions, additional gates, measurements).

Baseline: We use the statistical assertions (Stat) [Huang and Martonosi 2019b] and the QEC-inspired assertions (QECA) [Liu et al. 2020] as the baseline assertion schemes. To the best of our knowledge, they are the only published quantum program assertions till now. Stat employs a classical statistical test on the measurement results to check if a state satisfies a predicate. QECA introduces auxiliary qubits to indirectly measure the tested state.

5.1 Coverage Analysis

Assertion predicates: Proq employs projections which are able to represent a wide variety of predicates. However, both Stat and QECA only support three types of assertions: classical assertion, superposition assertion, and entanglement assertion. The expressive power difference has been summarized in Figure 1. For Stat, all these three types of assertions can be considered as $\mathrm{rank}\, P = 1$ special cases in Proq. The corresponding projections are

$P = |t\rangle \langle t|$, t ranges over all n-bit strings for classical assertion (suppose n qubits are asserted)

$P = |{+}{+}{+}\ldots\rangle \langle {+}{+}{+}\ldots|$ for superposition assertion

$P = (|00\ldots0\rangle + |11\ldots1\rangle)(\langle 00\ldots0| + \langle 11\ldots1|)$ for entanglement assertion

Stat's language does not support other types of states. QECA supports arbitrary 1-qubit states (these states can naturally cover the classical assertion and superposition assertion in Stat), some special 2-qubit entanglement states, and some special 3-qubit entangle states. These states can be considered as some $\mathrm{rank}\, P = 1, 2, 4$ special cases in Proq, respectively. So all QECA assertions are covered in Proq. Moreover, the implementations of QECA assertions are all designed manually without a systematic assertion implementation generation so they cannot be extended to more cases directly. The expressive power of the assertions in Proq, which can support many more complicated cases as introduced in Section 3 and 4, is much more than that of the baseline schemes.

Assertion locations: Thanks to the expressive power of the predicates in Proq, projection-based assertions can be injected at more locations with complex intermediate states in a program. The baseline schemes can only inject assertions at those locations with states that can be checked with the very limited types of assertions. If the baseline schemes insert assertions at locations with other types of states, their assertions will always return negative results since the predicates in their assertions are not correct. Therefore, the number of potential assertion injection locations of Proq is much larger than that of the baseline schemes.

5.2 Overhead Analysis

It is not easy to directly perform a fair overhead comparison between Proq and the baseline because Proq supports many more types of predicates as explained above. We first discuss the impact of this difference in assertion coverage in practical debugging.

Assertion coverage impact: Proq support assertions that cannot be implemented in Stat and QECA. These assertions will help locate the bug more quickly. When inserting assertions in a tested program, Proq assertions can always be injected closer to a potential bug because Proq allows more assertion injection locations. The potential bug location can then be narrowed down to a smaller program segment, which makes it easier for the programmers to manually search for the bug after an error message is reported.

Then we remove the assertion coverage difference by assuming all the assertions are within the three types of assertions supported in all assertion schemes.

Assertion checking overhead: We mainly discuss two aspects of the assertion checking overhead, 1) the number of assertion checking program executions and 2) the numbers of additional unitary transformations (quantum gates) and measurements to implement each of the assertions.

(1) **Compare with Stat:** Stat's approach is quite different from Proq. It only injects measurements to directly measure the tested states without any additional transformations.
(a) **number of executions:** The classical assertion, the first supported assertion type in Stat, is equivalent to the corresponding one in Proq. The tested state remains unchanged if it is the expected state. However, when checking for superposition states and entanglement states, the number of assertion checking program executions will be large because 1) Stat requires a large number of samples for each assertion to reconstruct an amplitude distribution over multiple basis states, and 2) the measurements will always affect the tested states so that only one assertion can be checked per execution. It is not yet clear how many executions are required since the statistical properties of checking Stat assertions are not well studied. The original Stat paper [Huang and Martonosi 2019b] claims to apply chi-square test and contingency table analysis (with no details about the testing process) on the measurement results collection of each assertion but it does not provide the numbers of required executions to achieve an acceptable confidence level for different assertions over different numbers of qubits, which makes it hard to directly compare the checking overhead (no publicly available code). We believe the number of executions will be large at least when the tested state is in a superposition state over multiple computational basis states. For example, the superposition assertion, which checks for the state $|{+}{+}{+}\ldots\rangle$ in an n-qubit system, requires $k \gg 2^n$ testing executions to observe a uniform distribution over all 2^n basis states.
(b) **number of gates and measurements:** For an assertion (any type) in Stat, it only requires n measurements on n qubits in assertion checking but it may need to be executed many times as explained above. For the corresponding assertions in Proq, a classical assertion requires n measurements (the same with Stat, e.g., Assertion A_0 in Figure 3). A superposition assertion requires additionally $2n$ H gates (e.g., Assertion A_1 in Figure 3). An entanglement assertion requires additionally $2(n-1)$ CNOT gates and 2 H gates (e.g., Assertion A_2 in Figure 3). Proq only needs few additional gates (linear to the number of qubits) for the commonly supported assertions.

(2) **Compare with QECA:** All QECA assertions are equivalent to their corresponding Proq assertions. Therefore, QECA has the same checking efficiency and supports multi-assertion per execution if we only consider those QECA-supported assertions. The statistical properties (Theorem 3.1 and 3.2) we prove can also be directly applied to QECA. So **the number of the assertion checking executions is the same** for QECA and Proq. The difference between QECA and Proq is that the actual assertion implementation in terms of quantum gates and measurements. The **implementation cost of Proq is lower** than that of QECA because QECA always need to couple the auxiliary qubits with existing qubits. We will have concrete data of the assertion implementation cost comparison between Proq and QECA later in a case study in Section 6.1.

6 CASE STUDIES: RUNTIME ASSERTIONS FOR REALISTIC QUANTUM ALGORITHMS

In this section, we perform case studies by applying projection-based assertions on two famous sophisticated quantum algorithms, the Shor's algorithm [Shor 1999] and the HHL algorithm [Harrow

et al. 2009]. For Shor's algorithm, we focus on a concrete example of its quantum order finding subroutine. The assertions are simple and can be supported by the baselines, which allows us to compare the resource consumption between Proq and the baseline and show that Proq could generate low overhead runtime assertions. For HHL algorithm, instead of just asserting a concrete circuit implementation, we will show that Proq could have non-trivial assertions that cannot be supported by the baselines. In these non-trivial assertions, we will illustrate how the proposed techniques, i.e., combining assertions, auxiliary qubits, local projection, can be applied in implementing the projections. Numerical simulation confirms that Proq assertions can work correctly.

6.1 Shor's Algorithm

Shor's algorithm was proposed to factor a large integer [Shor 1999]. Given an integer N, Shor's algorithm can find its non-trivial factors within $O(poly(log(N)))$ time. In this paper, we focus on its quantum order finding subroutine and omit the classical part which is assumed to be correct.

$p := |0\rangle^{\otimes n}$;
while $M[p] = 1$ do
  $p := |0\rangle^{\otimes n}$; $q := |0\rangle^{\otimes n}$; $\mathrm{assert}(p, q; A_0)$; $p := H^{\otimes n}[p]$; $\mathrm{assert}(p, q; A_1)$;
  $p, q := U_f[p, q]$; $\mathrm{assert}(p, q; A_2)$; $p := \mathrm{QFT}^{-1}[p]$; $\mathrm{assert}(p, q; A_3)$;
od

Fig. 2. Shor's algorithm program with assertions. The projections A_0, A_1, A_2, A_3 are defined in Section 6.1.2.

6.1.1 Shor's Algorithm Program. Figure 2 shows the program of the quantum subroutine in Shor's algorithm with the injected assertions in the quantum **while**-language. Briefly, it leverages Quantum Fourier Transform (QFT) to find the period of the function $f(x) = a^x \bmod N$ where a is a random number selected by a preceding classical subroutine. The transformation U_f, the measurement M, and the result set R are defined as follows:

$$U_f : \ |x\rangle_p |0\rangle_q \mapsto |x\rangle_p |a^x \bmod N\rangle_q, M = \left\{M_0 = \sum_{r\in R} |r\rangle\langle r|, M_1 = I - M_0\right\},$$

$$R = \{r \mid \gcd(a^{\frac{r}{2}} + 1, N) \text{ or } \gcd(a^{\frac{r}{2}} - 1, N) \text{ is a nontrivial factor of N}\}$$

For the measurement, the set R consists of the expected values that can be accepted by the follow-up classical subroutine. For a comprehensive introduction, please refer to [Nielsen and Chuang 2010].

6.1.2 Assertions for a Concrete Example. The circuit implementation we select for the subroutine is for factoring $N = 15$ with the random number $a = 11$ [Vandersypen et al. 2001]. Based on our understanding of Shor's algorithm, we have four assertions, A_0, A_1, A_2, and A_3, as shown in Figure 2. Figure 3 shows the final assertion-injected circuit with 5 qubits. The circuit blocks labeled with assert are for the four assertions with four projections defined as follows:

$$A_0 = |00000\rangle_{0,1,2,3,4}\langle 00000|; \ A_1 = |{+}{+}{+}\rangle_{0,1,2}\langle {+}{+}{+}| \otimes |00\rangle_{3,4}\langle 00|;$$
$$A_2 = |{+}{+}\rangle_{0,1}\langle {+}{+}| \otimes (|000\rangle + |111\rangle)_{2,3,4}(\langle 000| + \langle 111|);$$
$$A_3 = (|000\rangle + |001\rangle)_{0,1,2}(\langle 000| + \langle 001|) \otimes (|00\rangle + |11\rangle)_{3,4}(\langle 00| + \langle 11|).$$

We detail the implementation of the assertion circuit blocks in the upper half of Figure 3. For each assertion, we list its projection, the additional unitary transformations, with the complete implementation circuit diagram. For A_1, A_2, and A_3, since the qubits not fully entangled, we only assert part of the qubits without affecting the results. The unitary transformations are decomposed into CNOT gates and single-qubit gates, which is the same with QECA for a fair comparison.

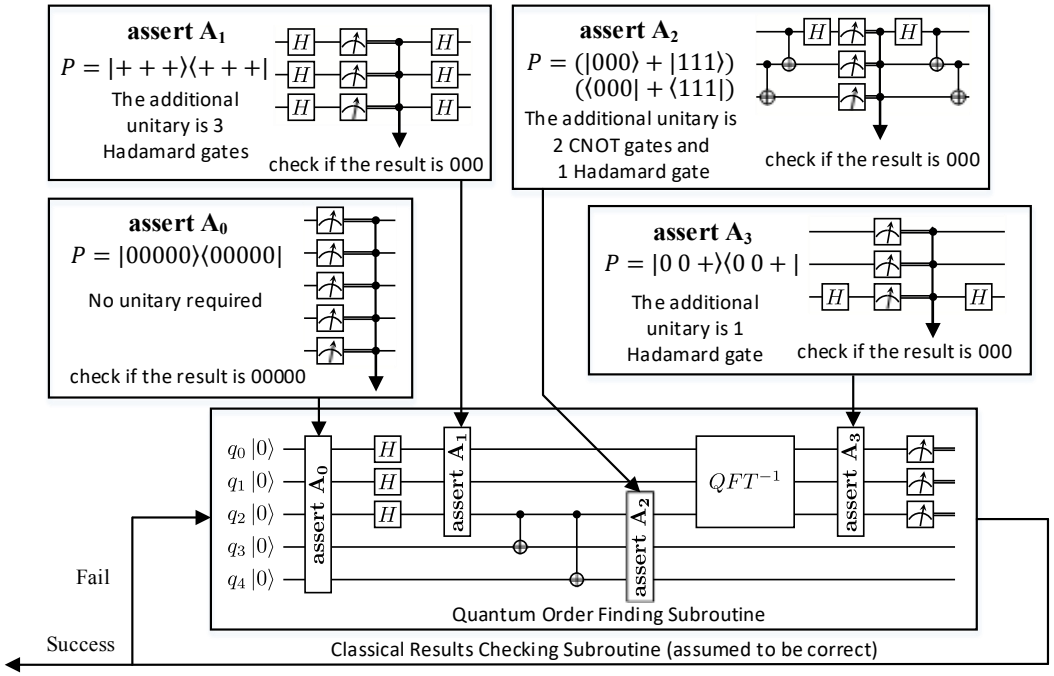

Fig. 3. Assertion-injected circuit implementation for Shor's algorithm with $N = 15$ and $a = 11$

6.1.3 Assertion Comparison. Similar to Section 5, we first compare the coverage of assertions for this realistic algorithm and then detail the implementation cost in terms of the number of additional gates, measurements, and auxiliary qubits.

Assertion coverage: All four assertions are supported in Stat and Proq. For QECA, A_0, A_1, and A_3 are covered but A_2 is not yet supported even if it is an entanglement state. The reason is that the QECA assertion only supports 3-qubit entanglement states with $rankP = 4$ but A_2 is a 3-qubit entanglement state with $rankA_2 = 1$.

We compare the circuit cost when implementing the assertions between Proq and QECA. Stat is not included because we have already discussed the implementation difference in Section 5.2 and it is not clear how many executions are required for Stat.

Table 1 shows the implementation cost of the three assertions supported by both Proq and QECA. In particular, we compare the number of H gates, CNOT gates, measurements, and auxiliary qubits. It can be observed that Proq uses no CNOT gates and auxiliary qubits for the three considered assertions, while QECA always needs to use additional CNOT gates and auxiliary qubits. This reason is that QECA always measures auxiliary qubits to indirectly probe the qubit information. So that additional CNOT gates are always required to couple the auxiliary qubits with existing qubits. This design significantly increases the implementation cost when comparing with Proq.

Table 1. Detailed assertion implementation cost comparison between Proq and QECA [Liu et al. 2020]

	A_0		A_1		A_3	
# of	Proq	QECA	Proq	QECA	Proq	QECA
H	0	0	6	6	2	2
CNOT	**0**	**5**	**0**	**6**	**0**	**4**
Measure	5	5	3	3	3	3
Aux. Qbit	**0**	**1**	**0**	**1**	**0**	**1**

To summarize, we demonstrate the complete assertion-injected circuit for a quantum program of Shor's algorithm and the implementation details of the assertions. We compare the implementation cost between Proq and QECA to show that Proq has lower cost for the limited assertions that are supported by both assertion schemes.

6.2 HHL Algorithm

In the first example of Shor's algorithm, we focus the assertion implementation on a concrete circuit example and compare against other assertions due to the simplicity of the intermediate states. In the next HHL algorithm example, we will have non-trivial assertions that are not supported in the baselines and demonstrate how to apply the techniques introduced in Section 4.

The HHL algorithm was proposed for solving linear systems of equations [Harrow et al. 2009]. Given a matrix A and a vector $\vec{b}$, the algorithm produces a quantum state $|x\rangle$ which is corresponding to the solution $\vec{x}$ such that $A\vec{x} = \vec{b}$. It is well-known that the algorithm offers up to an exponential speedup over the fastest classical algorithm if A is sparse and has a low condition number κ.

$p := |0\rangle^{\otimes n};\ q := |0\rangle^{\otimes m};\ r := |0\rangle;$
while $M[r] = 1$ do
  $\mathrm{assert}(p, r; P);$
  $q := |0\rangle^{\otimes m};\ q := U_b[q];\ p := H^{\otimes n}[p]; p, q := U_f[p, q];\ p := \mathrm{QFT}^{-1}[p];\ \mathrm{assert}(p; S);$
  $p, r := U_c[p, r];\ p := \mathrm{QFT}[p]; p, q := U_f^{\dagger}[p, q];\ p := H^{\otimes n}[p];\ \mathrm{assert}(p, q, r; R);$
od
$\mathrm{assert}(q; Q);$

Fig. 4. HHL algorithm program with assertions

6.2.1 HHL Program. The HHL algorithm has been formulated with the quantum **while**-language in [Zhou et al. 2019] and we adopt the assumptions and symbols there. Briefly speaking, A is a Hermitian and full-rank matrix with dimension $N = 2^m$, which has the diagonal decomposition $A = \sum_{j=1}^{N} \lambda_j |u_j\rangle\langle u_j|$ with corresponding eigenvalues λ_j and eigenvectors $|u_j\rangle$. We assume for all j, $\delta_j = \frac{\lambda_j t_0}{2\pi} \in \mathbb{N}^+$ and set $T = 2^n = \lceil \max_j \delta_j \rceil$, where t_0 is a time parameter to perform unitary transformation U_f. Moreover, the input vector $\vec{b}$ is presumed to be unit and corresponding to state $|b\rangle$ with the linear combination $|b\rangle = \sum_{j=1}^{N} \beta_j |u_j\rangle$. It is straightforward to find the solution state $|x\rangle = c \sum_{j=1}^{N} \frac{\beta_j}{\lambda_j} |u_j\rangle$ where c is for normalization.

The HHL program has three registers p, q, r which are $n, m, 1$-qubit systems and used as the control system, state system, and indicator of while loop, respectively. For detailed definitions of U_b, U_f, QFT, and the measurement M, please refer to [Harrow et al. 2009; Zhou et al. 2019].

6.2.2 Debugging Scheme for HHL Program. We introduce the debugging scheme for the HHL program shown in Figure 4. The projections P, Q, S, R are defined as follows:

$$P = |0\rangle_p\langle 0| \otimes |0\rangle_r\langle 0|;\ Q = |x\rangle_q\langle x|;\ S = \mathrm{supp}\left(\sum_{j=1}^{N} |\delta_j\rangle_p\langle \delta_j|\right)$$
$$R = |0\rangle_p\langle 0| \otimes (|x\rangle_q\langle x| \otimes |1\rangle_r\langle 1| + I_q \otimes |0\rangle_r\langle 0|).$$

Projection R is across all qubits while P is focused on register p, r and Q is focused on the output register q. These projections can be implemented using the techniques introduced in Section 4; more precisely:

(1) Implementation of $\mathrm{assert}(p, r; P)$:
measure register p and r directly to see if the outcomes are all "0";
(2) Implementation of $\mathrm{assert}(q; Q)$:
apply U_x on q; (additional unitary transformation in Section 4.1)
measure register q and check if the outcome is "0";
apply $U_x^\dagger$ on q;
(3) Implementation of $\mathrm{assert}(p, q, r; R)$:
measure register p directly to see if the outcome is "0";
introduce an auxiliary qubit a, initialize it to $|0\rangle$; (auxiliary qubit in Section 4.3)
apply U_x on q and U_R on r, q, a;
measure register a and check if the outcome is "0"; (combining assertions in Section 4.2)
apply $U_R^\dagger$ on r, q, a and $U_x^\dagger$ on q;

where U_x is defined by $U_x|x\rangle = |0\rangle$ and U_R is defined by

$$U_R|1\rangle_r\langle 1| \otimes |i\rangle_q\langle i| \otimes |k\rangle_a\langle k| = |1\rangle_r\langle 1| \otimes |i\rangle_q\langle i| \otimes |k \oplus 1\rangle_a\langle k \oplus 1|$$

for $i \geq 1$ and $k = 0, 1$ and unchanged otherwise.

We need to pay more attention to $\mathrm{assert}(p; S)$. The most accurate predicate here is

$$S' = \sum_{j,j'=1}^{N} \beta_j \overline{\beta}_{j'} |\delta_j\rangle_p\langle\delta_{j'}| \otimes |u_j\rangle_q\langle u_{j'}| \otimes |0\rangle_r\langle 0|$$

which is a highly entangled projection over register p and q. As discussed in Section 4.4, in order to avoid the hardness of implementing S', we introduce $S = \mathrm{supp}(\mathrm{tr}_{q,r}(S'))$ which is the local projection of S' over p. Though $\mathrm{assert}(p; S)$ is strictly weaker than original $\mathrm{assert}(p, q, r; S')$, it can be efficiently implemented and partially test the state.

6.2.3 Numerical Simulation Results. For illustration, we choose $m = n = 2$ as an example. Then the matrix A is 4×4 matrix and b is 4×1 vector. We first randomly generate four orthonormal vectors for $|u_j\rangle$ and then select δ_j to be either 1 or 3. Such configuration will demonstrate the applicability of all four techniques in Section 4. Finally, A and b are generated as follows.

$$A = \begin{bmatrix} 1.951 & -0.863 & 0.332 & -0.377 \\ -0.863 & 2.239 & -0.011 & -0.444 \\ 0.332 & -0.011 & 1.301 & -0.634 \\ -0.377 & -0.444 & -0.634 & 2.509 \end{bmatrix}, b = \begin{bmatrix} -0.486 \\ -0.345 \\ -0.494 \\ -0.633 \end{bmatrix}$$

Assertion coverage: We have four assertions, labeled P, Q, R, and S, for the HHL program. Only P is for a classical state and supported by the Stat and QECA. Q, R, and S are more complex and not supported by the baseline assertions.

Figure 5 shows the amplitude distribution of the states during the execution of the four assertions and each block corresponds to one assertion. Since our experiments are performed in simulation, we can directly obtain the state vector $|\psi\rangle$. The X-axis represents those basis states of which the amplitudes are not zero. The Y-axis is the probability of the measurement outcome. Each histogram represents the probability distribution across different computational basis states. This probability is be calculated by $\|\langle\psi|x\rangle\|^2$, where $|x\rangle$ is the corresponding basis state. The texts over the histograms represent the program locations where we record each of the states. For example, in '**Assertion Q**' block, we show that the state vector has non-zero amplitudes on multiple basis states. But after applying the unitary transformation Assertion Q, the state vector only has non-zero amplitudes on one basis state.

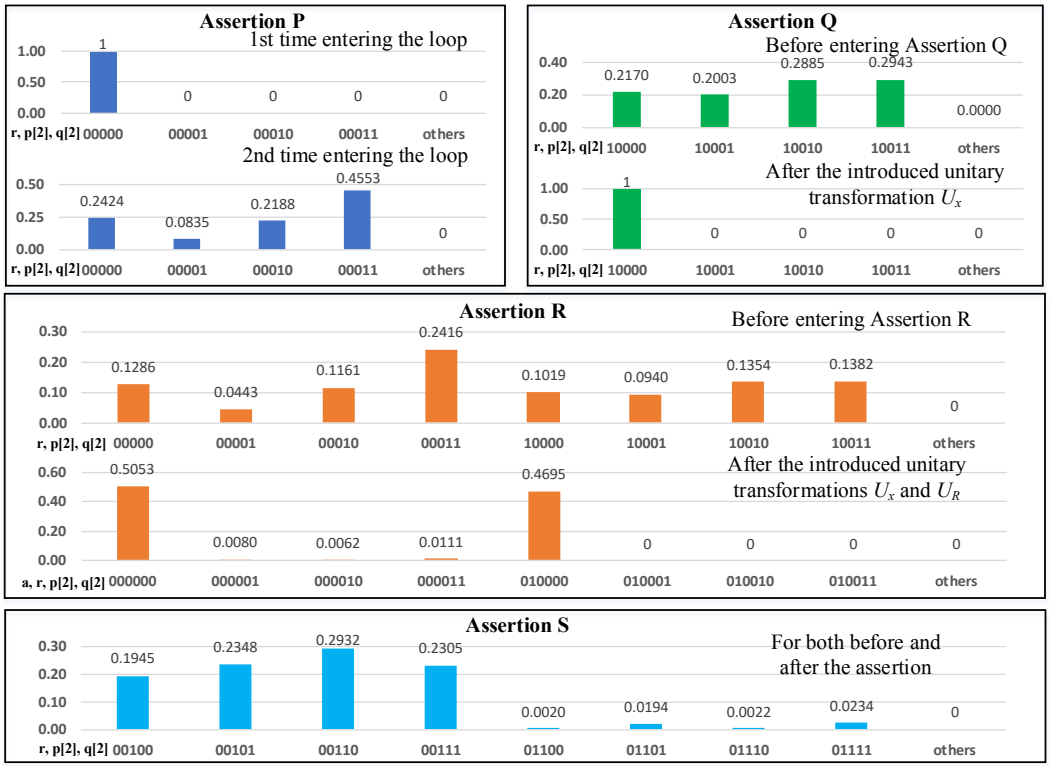

Fig. 5. Numerical simulation results for the states around the assertions in HHL algorithm

Assertion P is at the beginning of the loop body. The predicate is $P = |000\rangle_{r,p}\langle 000|$, which means that the quantum registers r and p should always be in state $|0\rangle$ and $|00\rangle$, respectively, at the beginning of the loop body. Figure 5 shows that when the program enters the loop D at the first and second time, the assertion is satisfied and the quantum registers r and p are 0.

Assertion Q is at the end of the program. Figure 5 shows that there are non-zero amplitudes at 4 possible measurement outcomes at the assertion location. But after the applied unitary transformation, the only possible outcome is 10000. Such an assertion is hard for Stat and QECA to describe but it is easy to define this assertion using projection in Proq.

Assertion R is at the end of the loop body. Figure 5 confirms that the basis states with non-zero amplitudes are in the subspace defined by the projection in assertion R. Its projection implementation involves the techniques of combining assertions and using auxiliary qubits. Such complex predicates cannot be defined in Stat and QECA while Proq can implement and check it.

Assertion S is in the middle of the loop body. At this place the state is highly entangled as mentioned above and directly implementing this projection will be expensive. We employ the local projection technique in Section 4.4. Since δ_js are selected to be either 1 or 3, the projection S becomes $|01\rangle_p\langle 01| + |11\rangle_p\langle 11|$. This simple form of local projection that can be easily implemented. Figure 5 confirms that the tested highly entangled state is not affected in this local projective measurement.

To summarize, we design four assertions for the program of HHL algorithm. Among them, only P can be defined in Stat and QECA. The remaining three assertions, which cannot be defined in Stat or QECA, demonstrate that Proq assertions can better test and debug realistic quantum algorithms.

7 DISCUSSION

Program testing and debugging have been investigated for a long time because it reflects the practical application requirements for reliable software. Compared with its counterpart in classical computing, quantum program testing and debugging are still at a very early stage. Even the basic testing and debugging approaches (e.g., assertions) are not yet available or well-developed for quantum programs. This paper made efforts towards practical quantum program runtime testing and debugging through studying how to design and implement effective and efficient quantum program assertions. Specifically, we select projections as predicates in our assertions because of the logical expressive power and efficient runtime checking property. We prove that quantum program testing with projection-based assertion is statistically effective. Several techniques are proposed to implement the projection under machine constraints. To the best of our knowledge, this is the first runtime assertion scheme for quantum program testing and debugging with such flexible predicates, efficient checking, and formal effectiveness guarantees. The proposed assertion technique would benefit future quantum program development, testing, and debugging.

Although we have demonstrated the feasibility and advantages of the proposed assertion scheme, several future research directions can be explored as with any initial research.

Projection implementation optimization: We have shown that our assertion-based debugging scheme can be implemented with several techniques in Section 3 and demonstrated concrete examples in Section 6. However, further optimization of the projection implementation is not yet well studied. One assertion can be split into several sub-assertions, but different sub-assertion selections would have different implementation overhead. We showed that one auxiliary qubit is enough but employing more auxiliary qubits may yield fewer sub-assertions. For the circuit implementation of an assertion, the decomposition of the assertion-introduced unitary transformations can be optimized for several possible objectives, e.g., gate count, circuit depth. A systematic approach to generate optimized assertion implementations is thus important for more efficient assertion-based quantum program debugging in the future.

More efficient checking: Assertions for a complicated highly entangled state may require significant effort for its precise implementation. However, the goal of assertions is to check if a tested state satisfies the predicates rather than to prove the correctness of a program. It is possible to trade-in checking accuracy for simplified assertion implementation by relaxing the constraints in the predicates. Local projection can be a solution to approximate a complex projective measurement as we discussed in Section 4.4 and demonstrated in one of the assertions for the HHL algorithm in Section 6. However, the degree of predicate relaxation and its effect on the robustness of the assertions in realistic erroneous program debugging need to be studied. Other possible directions, like non-demolition measurement [Braginsky et al. 1980], are also worth exploring.

8 RELATED WORK

This paper explores runtime assertion schemes for testing and debugging a quantum program on a quantum computer. In particular, the efficiency and effectiveness of our assertions come from the application of projection operators. In this section, we first introduce other existing runtime quantum program testing schemes, which are the closest related work, and then briefly discuss other quantum programming research involving projection operators.

8.1 Quantum Program Assertions

Recently, two types of assertions have been proposed for debugging on quantum computers. Huang and Martonosi proposed quantum program assertions based on statistical tests on classical observations [Huang and Martonosi 2019b]. For each assertion, the program executes from the

beginning to the place of the injected assertion followed by measurements. This process is repeated many times to extract the statistical information about the state. The advantage of this work is that, for the first time, assertion is used to reveal bugs in realistic quantum programs and help discover several bug patterns. But in this debugging scheme, each time only one assertion can be tested due to the destructive measurements. Therefore, the statistical assertion scheme is very time consuming. Proq circumvents this issue by choosing to use projective assertions.

Liu et al. further improved the assertion scheme by proposing dynamic assertion circuits inspired by quantum error correction [Liu et al. 2020]. They introduce ancilla qubits and indirectly collect the information of the qubits of interest. The success rate can also be improved since some unexpected states can be detected and corrected in the noisy scenarios. However, their approach requires manually designed transformation circuits and cannot be directly extended to more general cases. Their transformation circuits rely on ancilla qubits, which will increase the implementation overhead as discussed in Section 6.1.

Moreover, both of these assertion schemes can only inspect very few types of states that can be considered as some special cases of our proposed projection-based assertions, leading to limited applicability. In summary, our assertion and debugging schemes outperform these two existing assertion schemes mentioned above in terms of expressive power, flexibility, and efficiency.

8.2 Quantum Programming Language Research with Projections

Projection operators have been used in logic systems and static analysis for quantum programs. All projections in (the closed subspaces of) a Hilbert space form an orthomodular lattice [Kalmbach 1983], which is the foundation of the first Birkhoff-von Neumann quantum logic [Birkhoff and Von Neumann 1936]. After that, projections were employed to reason about [Brunet and Jorrand 2004] or develop a predicate transformer semantics [Ying et al. 2010] of quantum programs. Recently, projections were also used in other quantum logics for verification purposes [Unruh 2019; Yu 2019; Zhou et al. 2019]. Orthogonal to these prior works, this paper proposes to use projection-based predicates in assertion, targeting runtime testing and debugging rather than logic or static analysis.

9 CONCLUSION

The demand for bug-free quantum programs calls for efficient and effective debugging scheme on quantum computers. This paper enables assertion-based quantum program debugging by proposing Proq, a projection-based runtime assertion scheme. In Proq, predicates in the **assert** primitives are projection operators, which can significantly increase the expressive power and lower the assertion checking overhead compared with existing quantum assertion schemes. We study the theoretical foundations of quantum program testing with projection-based assertions to rigorously prove its effectiveness and efficiency. We also propose several transformations to make the projection-based assertions executable on measurement-restricted quantum computers. The superiority of Proq is demonstrated by its applications to inject and implement assertions for two well-known sophisticated quantum algorithms.

ACKNOWLEDGMENTS

We thank all anonymous reviewers for their valuable comments. This work was supported in part by National Science Foundation of US (Grant Nos. 1730309 and 1925717). N. Yu was supported by Australian Research Council (Grant No. DE180100156). M. Ying was supported in part by the National Key R&D Program of China (Grant No. 2018YFA0306701), the Australian Research Council (Grant Nos. DP160101652 and DP180100691), the National Natural Science Foundation of China (Grant No. 61832015). We are grateful to the Max Planck Institute for Software Systems for hosting L. Zhou.

REFERENCES

Ali Javadi Abhari, Arvin Faruque, Mohammad Javad Dousti, Lukas Svec, Oana Catu, Amlan Chakrabati, Chen-Fu Chiang, Seth Vanderwilt, John Black, Fred Chong, Margaret Martonosi, Martin Suchara, Ken Brown, Massoud Pedram, and Todd Brun. [n.d.]. *2012. Scaffold: Quantum Programming Language.* Technical Report. Technical Report TR-934-12. Princeton University.

Héctor Abraham, Ismail Yunus Akhalwaya, Gadi Aleksandrowicz, Thomas Alexander, Gadi Alexandrowics, Eli Arbel, Abraham Asfaw, Carlos Azaustre, Panagiotis Barkoutsos, George Barron, Luciano Bello, Yael Ben-Haim, Daniel Bevenius, Lev S. Bishop, Samuel Bosch, David Bucher, CZ, Fran Cabrera, Padraic Calpin, Lauren Capelluto, Jorge Carballo, Ginés Carrascal, Adrian Chen, Chun-Fu Chen, Richard Chen, Jerry M. Chow, Christian Claus, Christian Clauss, Abigail J. Cross, Andrew W. Cross, Juan Cruz-Benito, Cryoris, Chris Culver, Antonio D. Córcoles-Gonzales, Sean Dague, Matthieu Dartiailh, Abdón Rodríguez Davila, Delton Ding, Eugene Dumitrescu, Karel Dumon, Ivan Duran, Pieter Eendebak, Daniel Egger, Mark Everitt, Paco Martín Fernández, Albert Frisch, Andreas Fuhrer, IAN GOULD, Julien Gacon, Gadi, Borja Godoy Gago, Jay M. Gambetta, Luis Garcia, Shelly Garion, Gawel-Kus, Juan Gomez-Mosquera, Salvador de la Puente González, Donny Greenberg, John A. Gunnels, Isabel Haide, Ikko Hamamura, Vojtech Havlicek, Joe Hellmers, Łukasz Herok, Hiroshi Horii, Connor Howington, Shaohan Hu, Wei Hu, Haruki Imai, Takashi Imamichi, Raban Iten, Toshinari Itoko, Ali Javadi-Abhari, Jessica, Kiran Johns, Naoki Kanazawa, Anton Karazeev, Paul Kassebaum, Arseny Kovyrshin, Vivek Krishnan, Kevin Krsulich, Gawel Kus, Ryan LaRose, Raphaël Lambert, Joe Latone, Scott Lawrence, Dennis Liu, Peng Liu, Panagiotis Barkoutsos ZRL Mac, Yunho Maeng, Aleksei Malyshev, Jakub Marecek, Manoel Marques, Dolph Mathews, Atsushi Matsuo, Douglas T. McClure, Cameron McGarry, David McKay, Srujan Meesala, Antonio Mezzacapo, Rohit Midha, Zlatko Minev, Michael Duane Mooring, Renier Morales, Niall Moran, Prakash Murali, Jan Müggenburg, David Nadlinger, Giacomo Nannicini, Paul Nation, Yehuda Naveh, Nick-Singstock, Pradeep Niroula, Hassi Norlen, Lee James O'Riordan, Pauline Ollitrault, Steven Oud, Dan Padilha, Hanhee Paik, Simone Perriello, Anna Phan, Marco Pistoia, Alejandro Pozas-iKerstjens, Viktor Prutyanov, Jesús Pérez, Quintiii, Rudy Raymond, Rafael Martín-Cuevas Redondo, Max Reuter, Diego M. Rodríguez, Mingi Ryu, Martin Sandberg, Ninad Sathaye, Bruno Schmitt, Chris Schnabel, Travis L. Scholten, Eddie Schoute, Ismael Faro Sertage, Nathan Shammah, Yunong Shi, Adenilton Silva, Yukio Siraichi, Seyon Sivarajah, John A. Smolin, Mathias Soeken, Dominik Steenken, Matt Stypulkoski, Hitomi Takahashi, Charles Taylor, Pete Taylour, Soolu Thomas, Mathieu Tillet, Maddy Tod, Enrique de la Torre, Kenso Trabing, Matthew Treinish, TrishaPe, Wes Turner, Yotam Vaknin, Carmen Recio Valcarce, Francois Varchon, Desiree Vogt-Lee, Christophe Vuillot, James Weaver, Rafal Wieczorek, Jonathan A. Wildstrom, Robert Wille, Erick Winston, Jack J. Woehr, Stefan Woerner, Ryan Woo, Christopher J. Wood, Ryan Wood, Stephen Wood, James Wootton, Daniyar Yeralin, Jessie Yu, Laura Zdanski, and Zoufalc. 2019. Qiskit: An Open-source Framework for Quantum Computing. https://doi.org/10.5281/zenodo.2562110

Garrett Birkhoff and John Von Neumann. 1936. The logic of quantum mechanics. *Annals of mathematics* (1936), 823–843.

Vladimir B Braginsky, Yuri I Vorontsov, and Kip S Thorne. 1980. Quantum nondemolition measurements. *Science* 209, 4456 (1980), 547–557.

Olivier Brunet and Philippe Jorrand. 2004. Dynamic quantum logic for quantum programs. *International Journal of Quantum Information* 2, 01 (2004), 45–54.

Jianxin Chen, Zhengfeng Ji, Bei Zeng, and D. L. Zhou. 2012. From ground states to local Hamiltonians. *Phys. Rev. A* 86 (Aug 2012), 022339. Issue 2. https://doi.org/10.1103/PhysRevA.86.022339

Jianxin Chen, Fang Zhang, Cupjin Huang, Michael Newman, and Yaoyun Shi. 2018. Classical simulation of intermediate-size quantum circuits. *arXiv preprint arXiv:1805.01450* (2018).

Google. 2018. Announcing Cirq: An Open Source Framework for NISQ Algorithms. https://ai.googleblog.com/2018/07/announcing-cirq-open-source-framework.html.

Alexander S Green, Peter LeFanu Lumsdaine, Neil J Ross, Peter Selinger, and Benoît Valiron. 2013. Quipper: a scalable quantum programming language. In *ACM SIGPLAN Notices*, Vol. 48. ACM, 333–342.

Lov K Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. ACM, 212–219.

Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. 2009. Quantum algorithm for linear systems of equations. *Physical review letters* 103, 15 (2009), 150502.

Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. 2019. A Verified Optimizer for Quantum Circuits. *arXiv preprint arXiv:1912.02250* (2019).

Yipeng Huang and Margaret Martonosi. 2019a. QDB: From Quantum Algorithms Towards Correct Quantum Programs. In *9th Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Yipeng Huang and Margaret Martonosi. 2019b. Statistical assertions for validating patterns and finding bugs in quantum programs. In *Proceedings of the 46th International Symposium on Computer Architecture*. ACM, 541–553.

IBM. 2019. Gate and operation specification for quantum circuits. https://github.com/Qiskit/openqasm.

Ali JavadiAbhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T Chong, and Margaret Martonosi. 2015. ScaffCC: Scalable compilation and analysis of quantum programs. *Parallel Comput.* 45 (2015), 2–17.

Gudrun Kalmbach. 1983. *Orthomodular lattices*. Vol. 18. Academic Pr.

Yangjia Li and Mingsheng Ying. 2014. Debugging quantum processes using monitoring measurements. *Phys. Rev. A* 89 (Apr 2014), 042338. Issue 4. https://doi.org/10.1103/PhysRevA.89.042338

Noah Linden, Sandu Popescu, and William Wootters. 2002. Almost Every Pure State of Three Qubits Is Completely Determined by Its Two-Particle Reduced Density Matrices. *Phys. Rev. Lett.* 89 (Oct 2002), 207901. Issue 20. https://doi.org/10.1103/PhysRevLett.89.207901

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*. 1017–1030.

Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. 2014. Quantum principal component analysis. *Nature Physics* 10, 9 (2014), 631–633.

Michael A Nielsen and Isaac L Chuang. 2010. Quantum Computation and Quantum Information. *Quantum Computation and Quantum Information, by Michael A. Nielsen, Isaac L. Chuang, Cambridge, UK: Cambridge University Press, 2010* (2010).

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* (Paris, France) *(POPL 2017)*. ACM, New York, NY, USA, 846–858. https://doi.org/10.1145/3009837.3009894

Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O'brien. 2014. A variational eigenvalue solver on a photonic quantum processor. *Nature communications* 5 (2014), 4213.

Robert Rand, Jennifer Paykin, and Steve Zdancewic. 2018. QWIRE practice: Formal verification of quantum circuits in Coq. *arXiv preprint arXiv:1803.00699* (2018).

Rigetti. 2019. A Python library for quantum programming using Quil. https://github.com/rigetti/pyquil.

Rigetti Forest team. 2019. Forest SDK. https://www.rigetti.com/forest.

Yunong Shi, Xupeng Li, Runzhou Tao, Ali Javadi-Abhari, Andrew W Cross, Frederic T Chong, and Ronghui Gu. 2019. Contract-based verification of a realistic quantum compiler. *arXiv preprint arXiv:1908.08963* (2019).

Peter W Shor. 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review* 41, 2 (1999), 303–332.

Krysta M 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 domain-specific language. *arXiv preprint arXiv:1803.00652* (2018).

Dominique Unruh. 2019. Quantum relational hoare logic. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 33.

Lieven MK Vandersypen, Matthias Steffen, Gregory Breyta, Costantino S Yannoni, Mark H Sherwood, and Isaac L Chuang. 2001. Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance. *Nature* 414, 6866 (2001), 883.

A. Winter. 1999. Coding theorem and strong converse for quantum channels. *IEEE Transactions on Information Theory* 45, 7 (Nov 1999), 2481–2485. https://doi.org/10.1109/18.796385

William K Wootters and Wojciech H Zurek. 1982. A single quantum cannot be cloned. *Nature* 299, 5886 (1982), 802.

Tao Xin, Dawei Lu, Joel Klassen, Nengkun Yu, Zhengfeng Ji, Jianxin Chen, Xian Ma, Guilu Long, Bei Zeng, and Raymond Laflamme. 2017. Quantum State Tomography via Reduced Density Matrices. *Phys. Rev. Lett.* 118 (Jan 2017), 020401. Issue 2. https://doi.org/10.1103/PhysRevLett.118.020401

Mingsheng Ying. 2011. Floyd–hoare logic for quantum programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 33, 6 (2011), 19.

Mingsheng Ying. 2016. *Foundations of Quantum Programming*. Morgan Kaufmann.

Mingsheng Ying, Runyao Duan, Yuan Feng, and Zhengfeng Ji. 2010. Predicate transformer semantics of quantum programs. *Semantic Techniques in Quantum Computation* 8 (2010), 311–360.

Nengkun Yu. 2019. Quantum Temporal Logic. arXiv:1908.00158 [cs.LO]

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*. ACM, 1149–1162.