CoqQ: Foundational Verification of Quantum Programs

LI ZHOU, Max Planck Institute for Security and Privacy (MPI-SP), Germany and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
GILLES BARTHE, Max Planck Institute for Security and Privacy (MPI-SP), Germany and IMDEA Software Institute, Spain
PIERRE-YVES STRUB, Meta, France
JUNYI LIU, State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China and University of Chinese Academy of Sciences, China
MINGSHENG YING, State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China and Tsinghua University, China

CoqQ is a framework for reasoning about quantum programs in the Coq proof assistant. Its main components are: a deeply embedded quantum programming language, in which classic quantum algorithms are easily expressed, and an expressive program logic for proving properties of programs. CoqQ is foundational: the program logic is formally proved sound with respect to a denotational semantics based on state-of-art mathematical libraries (MathComp and MathComp Analysis). CoqQ is also practical: assertions can use Dirac expressions, which eases concise specifications, and proofs can exploit local and parallel reasoning, which minimizes verification effort. We illustrate the applicability of CoqQ with many examples from the literature.

CCS Concepts: • **Theory of computation → Interactive proof systems**; **Quantum computation theory**; **Programming logic**.

Additional Key Words and Phrases: Quantum Programs, Program Logics, Proof Assistants, Mathematical Libraries

ACM Reference Format:
Li Zhou, Gilles Barthe, Pierre-Yves Strub, Junyi Liu, and Mingsheng Ying. 2023. CoqQ: Foundational Verification of Quantum Programs. *Proc. ACM Program. Lang.* 7, POPL, Article 29 (January 2023), 33 pages. https://doi.org/10.1145/3571222

1 INTRODUCTION

Quantum programming languages hold the promise of making the computational power of quantum computers readily accessible to software developers. As such, they are a key element in large-scale efforts to deploy quantum computing. Examples of industrially supported quantum programming languages include IBM's Qiskit [Aleksandrowicz et al. 2019], Google's Cirq [The Cirq Developers 2018], and Microsoft's Q# [Svore et al. 2018]. In spite of the benefits of quantum programming

Authors' addresses: Li Zhou, Max Planck Institute for Security and Privacy (MPI-SP), Bochum, Germany and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China, li.zhou@mpi-sp.org, zhouli@ios.ac.cn; Gilles Barthe, Max Planck Institute for Security and Privacy (MPI-SP), Bochum, Germany and IMDEA Software Institute, Madrid, Spain, gjbarthe@gmail.com; Pierre-Yves Strub, Meta, France, strubpy@meta.com; Junyi Liu, State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China and University of Chinese Academy of Sciences, Beijing, China, liujy@ios.ac.cn; Mingsheng Ying, State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China and Tsinghua University, Beijing, China, yingms@ios.ac.cn.

This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2023 Copyright held by the owner/author(s).
2475-1421/2023/1-ART29
https://doi.org/10.1145/3571222

languages, writing correct quantum programs remains error-prone, not the least because quantum computation often contradicts human intuition.

(Deductive) program verification provides a principled approach for reasoning compositionally about programs. One fundamental strength of deductive program verification is its ability to reason about rich specifications, and thus to prove that programs are correct. Deductive program verification is traditionally based on program logics. Informally, these program logics feature syntax-directed proof rules, which can be used (backwards) to decompose proof goals into simpler goals. Ultimately the simplest goals can be established in isolation using (e.g., first-order predicate) logic. Program verification and program logics are used extensively by the software industry to validate large-scale classic developments. Similarly, there exist several program logics for reasoning about quantum programs (see Section 9). These logics obey the same principles as their counterparts for classical programs. Yet, they do not achieve similar usability and scalability.

Program logics for quantum programs face two main challenges in comparison to program logics for classical programs. First, quantum states and assertions have a richer structure than their classical counterparts. In particular, quantum states have the structure of a Hilbert space and quantum assertions are modelled as Hermitian operators [D'Hondt and Panangaden 2006]. As a consequence, proving entailment of assertions, as required for instance by the rule of consequence, often involves complex calculations. Second, the effect of atomic instructions on quantum states is inherently non-local, because these states may be entangled. As a consequence, proof rules for quantum programs do not have natural support for local reasoning, such as the classical frame rule in Hoare logic. This makes reasoning very cumbersome and error-prone.

A common means to ensure that proofs of program correctness adhere to the program logic is to provide mechanized support for the program logic. Mechanized support is beneficial in many ways: it can help bookkeeping proof obligations and discharge boilerplate obligations automatically. However, when program logics are themselves very complex, there is the risk that the implementation of the program logic is itself flawed. One appealing approach to address these concerns is to develop verified program verifiers, i.e. program verifiers which are themselves proved correct w.r.t. the semantics of programs. A verified program verifier typically consists of several ingredients:

- an embedding of the programming language. One often opts for a deep embedding, in which the syntax of programs is modelled as an inductive type;
- a semantics of the programming language. The semantics is built from first principles and defines the behaviour of programs;
- a formalization of the program logic. Typically, each rule of the program logic is formalized as a lemma showing the soundness of the rule w.r.t. the semantics.

As a consequence, a program verified using the formalized program logic can be deemed correct from first principles. There exist many verified verifiers for classical programming languages, including the Verified Software Toolchain (VST) [Appel 2011] for C programs—which in addition integrates the CompCert verified compiler [Leroy 2009]—and RustBelt developed in Iris for Rust programs [Jung et al. 2018a,b].

Contributions. In this paper, we present CoqQ, a verified program verifier for **qwhile**, a core imperative language with classical control flow and quantum data. As its name suggests, CoqQ is formalized in the Coq proof assistant, which has been used extensively to formalize mathematics and program semantics. The main components of CoqQ are:

- a formalization of the **qwhile** programming language. The crux of our formalization is a denotational semantics built from first principles. Specifically, we extend the algebraic hierarchy of MathComp [Mahboubi and Tassi 2021], an extensive library of formalized

mathematics, with a formalization of Hilbert spaces. Such a formalization of Hilbert spaces is essential for the generality of the approach and for enabling local reasoning, as we discuss below. Note that our formalization is restricted to finite-dimensional Hilbert spaces; we inherit this limitation from MathComp's formalization of vector spaces, which is also restricted to finite dimensions. However, this restriction is not a problem for verifying the existing quantum algorithms;

- a formalization of labelled Dirac notation. Dirac notation, a.k.a. bra - ket notation, is used ubiquitously to model quantum states. One major benefit of Dirac notation is to support equational reasoning about quantum systems. However, it does not reflect (and hence does not exploit) the structure of quantum states. To overcome this limitation, quantum physicists, in particular those working on many-body quantum physics, commonly use labels to tag quantum expressions with the (sub)systems in which they live or operate. The benefit of the resulting labelled Dirac notation is that it avoids the need of writing the matrix representation of a large state, an observable or a Hamiltonian of a many-body system; instead, it allows to define the above objects as a linear combination of the tensor products of local quantities with labels (e.g. subscripts) to indicate the involved subsystems.
  CoqQ provides a formalization of labelled Dirac notation; to our best knowledge, no such formalization has been developed before. The key element of the formalization is an interpretation of expressions written using labelled Dirac notation using abstract tensor products of Hilbert spaces. More specifically, given a finite set of symbols L and the abstract Hilbert space $\mathcal{H}_x$ for each $x \in L$ (the state Hilbert space of each quantum variable), we define the Hilbert space $\mathcal{H}_S \triangleq \bigotimes_{x \in S} \mathcal{H}_x$ corresponding to any subset $S \subseteq L$. We use this tensor product construction to interpret labelled Dirac notation. One technical issue is that a complex expression may be interpreted in a different Hilbert space than its immediate subexpressions. Our formalization addresses this issue by leveraging canonical structures and big operators, two classic tools already used extensively by the MathComp libraries.
  We leverage our formalization to validate identities that are commonly used when calculating using labelled Dirac notation, e.g. the following commutativity property:
  $$|\phi\rangle_{S_1}|\psi\rangle_{S_2} = |\psi\rangle_{S_2}|\phi\rangle_{S_1} \qquad \text{if} \qquad S_1 \cap S_2 = \emptyset$$
- a verified Hoare logic. Statements of the logic are of the form $\{A_{S_1}\}C\{B_{S_2}\}$, where C is a quantum program, A and B are linear operators over the subsystems defined by S_1 and S_2 respectively. Following a standard practice in verified program verifiers, every proof rule is formalized as a lemma that is stated relative to the denotational semantics of its corresponding construct. The program logic is a mild adaptation of [Ying 2011, 2019; Ying et al. 2018, 2022] that allows all linear operators as predicates rather than only quantum predicates (Hermitian operators with eigenvalues between 0 and 1 [D'Hondt and Panangaden 2006]), together with a new rule (R.Inner) that facilitate the use of forward reasoning. One specificity of our program logic is that it allows to express program correctness as state transformation. Specifically, our formalization supports judgments of the form:
  $$\vDash \{\!\!\{|u\rangle_{S_u}\}\!\!\} C \{\!\!\{|v\rangle_{S_v}\}\!\!\}$$
  stating that C transforms the input state $|u\rangle_{S_u}$ to the output state $|v\rangle_{S_v}$, assuming that $\||u\rangle_{S_u}\| = \||v\rangle_{S_v}\| = 1$. This kind of judgment provides a *human-readable* statement that looks akin to statements found in textbook presentations of quantum algorithms.

In order to illustrate the benefits of CoqQ, we conduct a representative set of case studies, including the HHL algorithm for solving linear equations [Harrow et al. 2009], Grover's search algorithm [Grover 1996], quantum phase estimation (QPE) and the hidden subgroup problem (HSP) algorithm

Table 1. Comparison with other mechanizations of quantum programming languages. The column (State) indicates if the formalization supports a general notion of state, i.e., (partial) density operators in arbitrary (typed) Hilbert space. The column (Logic) indicates if the formalization includes a program logic. The column (Found.) means the formalization is built from first principles. ‡ indicates that qrhl-tool supports relational verification. Libraries are discussed in Section 9.

Tool	Programming model	State	Logic	Found.	Proof Assistant	Libraries
QWIRE	Circuit	○	○	●	Coq	Std. Lib.
SQIR	Circuit	○	○	●	Coq	Std. Lib.
QHLProver	High-level	○	●	●	Isabelle	JNF, DL
qrhl-tool	High-level	●	●‡	○	Isabelle	JNF, CBO
QBRICKS	Circuit	○	●	●	Why3, SMT	Std. Lib.
IMD	Mathematics	○	○	●	Isabelle	JNF
CoqQ	High-level	●	●	●	Coq	MathComp

[Kitaev 1996; Lomont 2004], together with the circuit implementation of quantum Fourier transformation (QFT) and Bravyi-Gosset-Konig's algorithm for hidden linear function (HLF) problem [Bravyi et al. 2018]. Several of these examples have been verified before; one notable exception is the HSP algorithm, which requires the full generality of **qwhile** and cannot be expressed (let alone be verified) easily in other formalizations.

Related Work. There is a large body of work in the design, implementation, and verification of quantum programs. However, CoqQ is to our best knowledge the first formally verified program verifier for a high-level quantum programming language that operates over a general notion of state. We elaborate below, by contrasting CoqQ with other tools that support verification of quantum programs within proof assistants—a more extensive and detailed comparison is found in Section 9. The comparison is summarized pictorially in Table 1.

Our work is most closely related to formalizations of **qwhile**. There are two such formalizations. The first one is QHLProver [Liu et al. 2019], which is used for proving correctness of quantum programs based on quantum Hoare logic. The formalization is based on a representation of quantum states as matrices (using the Jordan Normal Form library from Isabelle [Thiemann and Yamada 2016]), rather than on a general theory of Hilbert spaces. The second one is qrhl [Unruh 2019b, 2020], which is used for relational verification of quantum programs. The formalization of qrhl is not (yet) foundational. Rather, the proof rules of the logic are modelled as axioms.

However, the currently prevailing line of work uses proof assistants to reason about circuit-based quantum programs that operate over concrete representations of quantum states. There are several efforts in this direction, including QWIRE [Paykin et al. 2017; Rand et al. 2017], SQIR [Hietala et al. 2021a,b], QBRICKS [Chareton et al. 2021] and IMD (Isabelle Marries Dirac) [Bordg et al. 2021]. These efforts are rather different from ours, and elide several of the key challenges addressed by CoqQ.

As usual, the tools cannot be ordered by comparing the number of ● in the table. In particular, there are evident trade-offs between high-level programming languages and circuits. Approaches based on the former can typically ease the specification and verification of high-level algorithms, whereas approaches based on the latter remain closer to implementations, and blend more nicely with verified compilers. We discuss these trade-offs further in Section 9.

2 MOTIVATING EXAMPLE: HIDDEN SUBGROUP PROBLEM

Let G be a finite group and Y be a finite set. The Hidden Subgroup Problem (HSP) [Kitaev 1996] is to compute, given oracle access to a function $f : G \to Y$, its hidden subgroup structure. More precisely, assuming that there exists a subgroup H of G such that for every $g_1, g_2 \in G$, $f(g_1) = f(g_2)$ if and only if $g_2 - g_1 \in H$; the Hidden Subgroup Problem (HSP) is to compute a subset $Z \subseteq G$ such that its generated subgroup $\langle Z \rangle$ satisfies $\langle Z \rangle = H$. The problem arises in many settings, including integer factorization, discrete logarithm, and graph theory.

The existence of polynomial-time quantum algorithms for solving HSP over arbitrary finite groups remains an open problem. However, there are polynomial-time quantum algorithms for solving HSP over finite abelian groups. The core of these algorithms is a quantum procedure that samples uniformly over the orthogonal subgroup $H^\perp$ of H. The subgroup is defined as $H^\perp = \{g \in G \mid \forall h \in H.\ \chi_g(h) = 1\}$, where $\chi_g : G \to \mathbb{C}^*$ is a so-called character (defined formally in Section 8.1; definition is irrelevant for this overview)—recall that $\mathbb{C}^*$ is the set of non-zero complex numbers. Given a procedure for sampling uniformly from $H^\perp$, one can solve the Hidden Subgroup Problem by generating sufficiently many samples, and by applying a classical algorithm to convert a generating set for $H^\perp$ into a generating set for H. We omit the details here and refer the reader to [Lomont 2004; Nielsen and Chuang 2002] for further details.

```
for 0 ≤ i < k do x_i := |0⟩;
y := |y_0⟩ ;
for 0 ≤ i < k do x_i := QFT[x_i];
[x̄, y] := U[f][x̄, y];
for 0 ≤ i < k do x_i := QFT[x_i].
```

Fig. 1. HSP algorithm

The code *HSP* of the sampling algorithm appears in Figure 1. The code operates over two quantum registers: the register $\overline{\mathsf{x}}$ ranges over the cyclic group decomposition $\mathbb{Z}_{p_0} \times \ldots \times \mathbb{Z}_{p_{k-1}}$ of G and the register y ranges over the finite set Y. We let x_i denote the ith projection of $\overline{\mathsf{x}}$. We note that this ability to declare rich types for variables simplifies the writing of the algorithm—and ultimately eases verification. We briefly comment on the code:

- the first two lines initialize the registers $\overline{\mathsf{x}}$ and y to default values;
- the second **for** loop applies QFT (Quantum Fourier Transform) on every x_i;
- the next line applies the unitary transformation $U[f]$ that provides quantum access to f;
- the third **for** loop applies QFT on every x_i.

Correctness of *HSP* is stated informally as follows: *HSP* samples uniformly from the so-called orthogonal subgroup $H^\perp$ of H—the formal definition of $H^\perp$ is deferred to Section 8.1. However, assertions in quantum Hoare logic are Hermitian operators that model observables of the quantum system. In order to capture the correctness of *HSP* formally, we consider the assertions (parameterized by $g \in G$) $\{|g\rangle_{\overline{\mathsf{x}}}\langle g|\}$, which represents the probability of observing g when measuring $\overline{\mathsf{x}}$ on the output state. Correctness is captured by the following statements (for simplicity, we omit proof modes that will be discussed later):

$$\forall g \in H^\perp,\ \models \left\{\frac{1}{|H^\perp|}\right\}\ HSP\ \ \{|g\rangle_{\overline{\mathsf{x}}}\langle g|\} \tag{1}$$

$$\forall g \notin H^\perp,\ \models \{0\}\ \ HSP\ \ \{|g\rangle_{\overline{\mathsf{x}}}\langle g|\}\,. \tag{2}$$

Taken together, these statements entail that *HSP* outputs an element of $H^\perp$ uniformly at random (see Section 8.1 for details). The derivation of these triples in quantum Hoare logic proceeds structurally, following the order of execution:

- we use local and parallel reasoning to reason about the initialization of the registers;

- we use local and parallel reasoning, and some auxiliary proof about QFT, to reason about the application of QFT. Moreover, we use basic facts from group theory to transform the assertion into an equivalent one;
- we use the unitary rule to reason about the oracle call. Moreover, we use basic facts from group theory to transform the assertion into an equivalent one;
- we use local and parallel reasoning, and some auxiliary proof about QFT, to reason about the application of QFT. Moreover, we use basic facts from group theory to transform the assertion into an equivalent one;
- last, we apply a new rule, called (R.Inner), to establish the desired Hoare triple. At a high-level, the rule helps to combine backward and forward reasoning, and helps to write formal proofs that follow informal proofs closely.

Note that each instruction combines an application of the proof rule, proof obligations to justify local/parallel reasoning, and domain-specific reasoning (here group theory) to recast intermediate assertions into a suitable form. This makes the proof intricate, and a good example for mechanization in a verified program verifier. We provide additional details in Section 8.1.

3 PRELIMINARIES

Quantum computing is built upon quantum mechanics (the finite-dimensional instance of) which might be characterized by linear algebra. We will briefly review some basic definitions of linear algebra and then give a quick introduction to how quantum mechanics are formalized using concepts in linear algebra.

3.1 Abstract Linear Algebra

We assume that readers are familiar with basic (abstract) linear algebra. We write $\mathbb{C}$ for the set of complex numbers, $\mathbf{i}$ for the imaginary of $\mathbb{C}$ and $\overline{c}$ for the conjugate of $c \in \mathbb{C}$, U, V for linear spaces and $\mathbf{u}, \mathbf{v}, \mathbf{w}$ for vectors in a linear space.

We start with the definition of tensor products.

Definition 3.1 (Tensor product). *The tensor product space of U and V might be defined*[1] *as $U \otimes V \triangleq \mathrm{span}\{(\mathbf{u}_i, \mathbf{v}_j)\}$ where $\{\mathbf{u}_i\}$ and $\{\mathbf{v}_j\}$ are bases of U and V respectively. For any $\mathbf{u} = \sum_i a_i \mathbf{u}_i \in U$ and $\mathbf{v} = \sum_j b_j \mathbf{v}_j \in V$, their tensor product is defined as:*

$$\mathbf{u} \otimes \mathbf{v} \triangleq \sum_i \sum_j a_i b_j (\mathbf{u}_i, \mathbf{v}_j) \in U \otimes V.$$

Next, we consider linear maps. Let $\mathcal{L}(U; V)$ denote the set of all linear maps from U to V, and $\mathcal{L}(U) \triangleq \mathcal{L}(U; U)$ for all linear maps on U. $\mathcal{L}(U; V)$ forms a linear space with addition $f + g : \mathbf{u} \mapsto f(\mathbf{u}) + g(\mathbf{u})$ and scalar multiplication $af : \mathbf{u} \mapsto af(\mathbf{u})$. Note that identity map $I : \mathbf{u} \mapsto \mathbf{u}$ is a linear map and thus belongs to $\mathcal{L}(U)$. Given two linear maps $f \in \mathcal{L}(U; V)$ and $g \in \mathcal{L}(W, U)$, let $f \circ g : \mathbf{w} \mapsto f(g(\mathbf{w}))$ be the composition of f and g which is again a linear map $f \circ g \in \mathcal{L}(W, V)$; composition is associative, i.e., $f \circ (g \circ h) = (f \circ g) \circ h$. We can also define tensor products of sets of linear maps: since $\mathcal{L}(U_1, V_1)$ and $\mathcal{L}(U_2, V_2)$ are both linear spaces, we can define the tensor product space $\mathcal{L}(U_1, V_1) \otimes \mathcal{L}(U_2, V_2) \simeq \mathcal{L}(U_1 \otimes U_2; V_1 \otimes V_2)$ as well as the tensor product of linear maps, i.e., $f \otimes g : \mathbf{u} \otimes \mathbf{v} \mapsto f(\mathbf{u}) \otimes g(\mathbf{v}) \in \mathcal{L}(U_1 \otimes U_2; V_1 \otimes V_2)$.

We now turn to Hilbert spaces. They play an essential role in quantum mechanics.

[1]There are alternative ways to define the tensor product up to an isomorphism.

Definition 3.2 (Finite-dimensional Hilbert space over $\mathbb{C}$). *A finite-dimensional Hilbert space $\mathcal{H}$ over $\mathbb{C}$ (Hilbert space for short) is a finite-dimensional linear space over $\mathbb{C}$ equipped with an inner product $\langle\cdot,\cdot\rangle$ mapping each pair of vectors to $\mathbb{C}$ such that*[2]*:*

(1) (conjugate symmetric) $\langle \mathbf{u},\mathbf{v}\rangle = \overline{\langle \mathbf{v},\mathbf{u}\rangle}$ for all $\mathbf{u},\mathbf{v}\in\mathcal{H}$;
(2) (linear on the second argument) $\langle \mathbf{u}, a\mathbf{v}+\mathbf{w}\rangle = a\langle \mathbf{u},\mathbf{v}\rangle + \langle \mathbf{u},\mathbf{w}\rangle$ for all $a\in\mathbb{C}$ and $\mathbf{u},\mathbf{v},\mathbf{w}\in\mathcal{H}$;
(3) (positive definite) $\langle \mathbf{u},\mathbf{u}\rangle \geq 0$ for all $\mathbf{u}\in\mathcal{H}$ and "=" holds if and only if $\mathbf{u}=0$.

Hilbert spaces are linear spaces and thus inherit all their definitions and results; e.g., $\mathcal{L}(\mathcal{H})$ is the set of linear maps on $\mathcal{H}$. We often call $f\in\mathcal{L}(\mathcal{H}_1;\mathcal{H}_2)$ a (linear) operator. We can further define: *(induced) norm*: $\|\mathbf{u}\| \triangleq \sqrt{\langle \mathbf{u},\mathbf{u}\rangle}$. We call $\mathbf{u}$ a unit vector if $\|\mathbf{u}\|=1$; *orthonormal basis (ONB)*: a basis $\{\mathbf{u}_i\}$ such that $\|\mathbf{u}_i\|=1$ and $\langle \mathbf{u}_i,\mathbf{u}_j\rangle = 0$ if $i\neq j$; *outer product*: $[\mathbf{u},\mathbf{v}] : \mathbf{w}\mapsto \langle \mathbf{v},\mathbf{w}\rangle\mathbf{u} \in \mathcal{L}(\mathcal{H}_2;\mathcal{H}_1)$ for $\mathbf{u}\in\mathcal{H}_1$ and $\mathbf{v}\in\mathcal{H}_2$; *adjoint* of $A\in\mathcal{L}(\mathcal{H}_1,\mathcal{H}_2)$: the unique operator $A^\dagger\in\mathcal{L}(\mathcal{H}_2,\mathcal{H}_1)$ such that $\forall\,\mathbf{u},\mathbf{v}\in\mathcal{H},\ \langle \mathbf{u}, A(\mathbf{v})\rangle = \langle A^\dagger(\mathbf{u}),\mathbf{v}\rangle$; *trace* of $A\in\mathcal{L}(\mathcal{H})$: $\mathrm{tr}A \triangleq \sum_i \langle \mathbf{u}_i, A(\mathbf{u}_i)\rangle$ where $\{\mathbf{u}_i\}$ is an ONB of $\mathcal{H}$. It is often convenient to view the *partial trace* of an operator in $\mathcal{L}(\mathcal{H}_1\otimes\mathcal{H}_2)$ as a linear map, using tensor products; that is, we define the linear maps $\mathrm{tr}_1 \in \mathcal{L}(\mathcal{L}(\mathcal{H}_1\otimes\mathcal{H}_2);\mathcal{L}(\mathcal{H}_2))$ as $\mathrm{tr}_1 : f\otimes g\mapsto \mathrm{tr}(g)f$ and $\mathrm{tr}_2\in\mathcal{L}(\mathcal{L}(\mathcal{H}_1\otimes\mathcal{H}_2);\mathcal{L}(\mathcal{H}_1))$ as $\mathrm{tr}_2 : f\otimes g\mapsto \mathrm{tr}(f)g$.

There are many important subclasses of linear operators. Suppose $A\in\mathcal{L}(\mathcal{H})$, we define:

- Hermitian: $A^\dagger = A$;
- positive-semidefinite (or simply *positive*) (denoted by $\mathcal{L}^+(\mathcal{H})$): $\forall \mathbf{u}\in\mathcal{H}, \langle \mathbf{u}, A(\mathbf{u})\rangle \geq 0$;
- density operator (denoted by $\mathcal{D}^1(\mathcal{H})$): A is positive and $\mathrm{tr}A = 1$;
- partial density operator (denoted by $\mathcal{D}(\mathcal{H})$): A is positive and $\mathrm{tr}A\leq 1$;
- quantum predicate (or effect; denoted by $\mathcal{O}(\mathcal{H})$): A and $I-A$ are both positive;
- unitary operator (denoted by $\mathcal{U}(\mathcal{H})$: $A^\dagger\circ A = A\circ A^\dagger = I$;
- projection (denoted by $\mathcal{P}(\mathcal{H})$: A is positive and $A\circ A = A$.

The positivity property can be used to define the Löwner order $A\sqsubseteq B$ order on linear maps: for any $A,B\in\mathcal{L}(\mathcal{H})$, $A\sqsubseteq B$ iff $B-A$ is positive. Note that $(\mathcal{O}(\mathcal{H}),\sqsubseteq)$ forms a complete partial order (CPO).

One important class of linear operators are the so-called super-operators. A super-operator from $\mathcal{H}_1$ to $\mathcal{H}_2$ is an element of $\mathcal{SO}(\mathcal{H}_1;\mathcal{H}_2)\triangleq \mathcal{L}(\mathcal{L}(\mathcal{H}_1);\mathcal{L}(\mathcal{H}_2))$. By convention, we write $\mathcal{I}$ for the identity super-operator (which in fact is I on $\mathcal{L}(\mathcal{H})$). Similarly, we write $\mathcal{SO}(\mathcal{H})\triangleq\mathcal{L}(\mathcal{L}(\mathcal{H}))$ for the set of super-operators on $\mathcal{H}$. We remark that $\mathcal{SO}(\cdot;\cdot)$ is again a linear space, and thus we can define their tensor product. For a super-operator $\mathcal{E}$, we focus on the following properties:

- positive: $\forall A\in\mathcal{L}^+(\mathcal{H}_1), \mathcal{E}(A)\in\mathcal{L}^+(\mathcal{H}_1)$;
- completely-positive (denoted by $\mathcal{CP}(\mathcal{H}_1;\mathcal{H}_2)$): $\mathcal{E}\otimes\mathcal{I}$ is positive for identity super-operators $\mathcal{I}\in\mathcal{SO}(\mathcal{H}')$ on any $\mathcal{H}'$;
- trace-preserving: $\forall A\in\mathcal{L}^+(\mathcal{H}_1)$, $\mathrm{tr}(\mathcal{E}(A)) = \mathrm{tr}(A)$;
- trace-nonincreasing: $\forall A\in\mathcal{L}^+(\mathcal{H}_1)$, $\mathrm{tr}(\mathcal{E}(A))\leq\mathrm{tr}(A)$;
- quantum operation (denoted by $\mathcal{QO}(\mathcal{H}_1;\mathcal{H}_2)$) : completely-positive and trace-nonincreasing;

Linear algebraic structure. Given a Hilbert space $\mathcal{H}$, then all $\mathcal{H}$, $\mathcal{L}(\mathcal{H})$ and $\mathcal{SO}(\mathcal{H})$ have linear algebraic structure, and thus we can freely use addition + and scalar multiplication and apply the theory of linear space. However, the subsets such as quantum predicates $\mathcal{O}(\mathcal{H})$ and quantum operations $\mathcal{QO}(\mathcal{H})$ do not have such structure, i.e., they are not closed under addition or scalar multiplication; we need to be careful, for example, additional proofs are required to ensure $O_1+O_2\in\mathcal{O}(\mathcal{H})$ even if $O_1,O_2\in\mathcal{O}(\mathcal{H})$.

[2]Finite-dimensional linear space is complete w.r.t. any vector norm, and thus we omit the condition of completeness.

3.2 Introduction to Quantum Mechanics

Let us briefly introduce the fundamental concepts of quantum mechanics and their corresponding mathematical objects in linear algebra.

State Space. The state space of any isolated physical system is a Hilbert space $\mathcal{H}$. The state of a system is completely described by a unit vector $\mathbf{u} \in \mathcal{H}$. To model the probabilistic ensembles of quantum states, *(partial) density operators* are introduced; in detail, if the system is in one of the $\{\mathbf{u}_i\}$ with probability p_i ($\sum_i p_i = 1$), then the system is fully characterized by $\rho \triangleq \sum_i p_i[\mathbf{u}_i, \mathbf{u}_i] \in \mathcal{D}^1(\mathcal{H})$ the summation of the outer product of $\mathbf{u}_i$ with weight p_i.

Evolution. The evolution of a closed quantum system is described by a unitary operator[3]. That is, if the initial state is $\mathbf{u}$ and the evolution is described by U, then the final state is $U(\mathbf{u})$. More generally, the evolution of an open quantum system which might interact with the environment, is modelled as a quantum operation $\mathcal{E} \in \mathcal{QO}(\mathcal{H})$, i.e., a complete-positive trace-nonincreasing superoperator on $\mathcal{H}$; if the initial state is $\rho \in \mathcal{D}(\mathcal{H})$, then the final state is $\mathcal{E}(\rho)$.

Quantum Measurement. A quantum measurement on a system with state space $\mathcal{H}$ is described by a collection of operators $\{M_m\}_{m\in J}$ ($M_m \in \mathcal{L}(\mathcal{H})$ and J is the index set) such that $\sum_{m\in J} M_m^\dagger \circ M_m = I$, where J is the set of all possible measurement outcomes and I the identity operator. It is physically interpreted as : if the state is $\mathbf{u}$ immediately before the measurement then the probability that result m occurs is $p(m) = \|M_m(\mathbf{u})\|$ and the post-measurement state is $\frac{M_m(\mathbf{u})}{\sqrt{p(m)}}$. If the state is described by density operator ρ, then $p(m) = \mathrm{tr}(M_m^\dagger \circ M_m \circ \rho)$ and the post-measurement state is $\frac{M_m \circ \rho \circ M_m^\dagger}{\mathrm{tr}(M_m^\dagger \circ M_m \circ \rho)}$. Performing quantum measurement is the only way to extract classical information (i.e., outcome) from a quantum system.

Quantum Predicates. Quantum predicates are defined as physical observables $O \in \mathcal{O}(\mathcal{H})$. The motivation comes from [D'Hondt and Panangaden 2006]. Informally, we can build for every observable O a projective measurement $\mathcal{M} = \{P_\lambda\}$ where λ ranges over the eigenvalues of O and $P_\lambda \in \mathcal{P}(\mathcal{H})$ such that $O = \sum_\lambda P_\lambda$. Then suppose that the system is in state $\rho \in \mathcal{D}^1(\mathcal{H})$; after performing the measurement $\mathcal{M}$, we have probability $\mathrm{tr}(P_\lambda \circ \rho)$ to obtain as outcome λ. Moreover, note that *expectation*, i.e., average value of outcome, is computed as $\sum_\lambda \mathrm{tr}(P_\lambda \circ \rho)\lambda = \mathrm{tr}(O \circ \rho)$. Thus, $\mathrm{tr}(O \circ \rho)$ is the expectation of O in state ρ. This expectation might be interpreted as the degree to which the quantum state ρ satisfies quantum predicate O.

Composite System. The state space of a composite quantum system is the tensor product of the state spaces of the component physical systems. Specifically, if system $i \in \{1, 2, \cdots, n\}$ is prepared in $\mathbf{u}_i \in \mathcal{H}_i$, then the joint state of the total system is $\bigotimes_{i=1}^n \mathbf{u}_i = \mathbf{u}_1 \otimes \mathbf{u}_2 \otimes \cdots \otimes \mathbf{u}_n \in \bigotimes_{i=1}^n \mathcal{H}_i$.

Given a density operator $\rho \in \mathcal{D}(\mathcal{H}_A \otimes \mathcal{H}_B)$ of a composite system AB, we define $\rho|_A \triangleq \mathrm{tr}_B(\rho) \in \mathcal{D}(\mathcal{H}_A)$ and $\rho|_B \triangleq \mathrm{tr}_A(\rho) \in \mathcal{D}(\mathcal{H}_B)$ as the reduced density operators of each subsystem A and B. Physically, the reduced density operator fully describes the state of a subsystem if all other subsystems are discarded or ignored.

Entanglement. This is one of the most important features of the quantum world. Mathematically, it says that there exists state $\mathbf{w} \in \mathcal{H}_A \otimes \mathcal{H}_B$ (a state of the composite system AB) such that, there does not exist $\mathbf{u} \in \mathcal{H}_A, \mathbf{v} \in \mathcal{H}_B$ satisfies $\mathbf{w} = \mathbf{u} \otimes \mathbf{v}$. It suggests that it is impossible to prepare $\mathbf{w}$ by preparing two independent states of A and B respectively; and what is more, the local operation in one of the systems will influence the other, for example, the outcomes of performing measurements on A and B separately are correlated.

[3]Such transformation is fully determined by Schrödinger equation of the system.

4 LABELLED DIRAC NOTATION

Dirac notation (a.k.a. bra-ket notation) is a widely used notation for representing quantum states. One of the main appeals of the notation is to simplify calculations on quantum states. Informally, the crux of the Dirac notation is to provide a notation that can describe all mathematical entities and operations used in quantum mechanics and to interpret these ingredients uniformly as linear operators. By this means, the Dirac notation allows to calculate over descriptions of quantum systems using basic rewriting rules of linear algebra such as linearity, associativity and commutativity.

The standard Dirac notation combines the following ingredients:

(1) ket $|\cdot\rangle$ represents a state, i.e., $|u\rangle \triangleq \mathbf{u} \in \mathcal{H}$. Since $\mathbb{C}$ itself is a Hilbert space with $\langle a, b\rangle \triangleq \overline{a}b$, states can also be viewed as elements of $\mathcal{L}(\mathbb{C};\mathcal{H})$;
(2) bra $\langle\cdot|$ represents a co-state, i.e, a linear map defined by $\langle u| \triangleq \mathbf{v} \mapsto \langle \mathbf{u}, \mathbf{v}\rangle \in \mathcal{L}(\mathcal{H};\mathbb{C})$;
(3) inner product $\langle\cdot|\cdot\rangle$, i.e., $\langle u|v\rangle \triangleq \langle \mathbf{u}, \mathbf{v}\rangle \in \mathbb{C}$; inner products can also be viewed as linear maps over $\mathbb{C}$, and more specifically as the composition of the bra and the ket viewed as linear maps, i.e. $\langle u|v\rangle \in \mathcal{L}(\mathbb{C})$ with $\langle u|v\rangle = \langle u| \circ |v\rangle$;
(4) outer product $|\cdot\rangle\langle\cdot|$, i.e., $|u\rangle\langle v| \triangleq [\mathbf{u}, \mathbf{v}] \in \mathcal{L}(\mathcal{H}_1;\mathcal{H}_2)$ if $\mathbf{u} \in \mathcal{H}_2$ and $\mathbf{v} \in \mathcal{H}_1$; this definition of outer product is also consistent with composition $|u\rangle\langle v| = |u\rangle \circ \langle v|$;
(5) tensor product $|\cdot\rangle|\cdot\rangle$ and $\langle\cdot|\langle\cdot|$, i.e., $|u\rangle|v\rangle \triangleq \mathbf{u} \otimes \mathbf{v} \in \mathcal{H}_1 \otimes \mathcal{H}_2$ and $\langle u|\langle v| \triangleq \langle u| \otimes \langle v| \in \mathcal{L}(\mathcal{H}_1 \otimes \mathcal{H}_2;\mathbb{C})$ if $\mathbf{u} \in \mathcal{H}_1$ and $\mathbf{v} \in \mathcal{H}_2$.

Example 4.1. *The Dirac notation eases equational reasoning, as shown by the following example (the linear map view of expressions is shown in the second line).*

$$\begin{array}{lccccc} & (|\phi\rangle\langle\psi|)(|\alpha\rangle\langle\beta|) & \xlongequal{assoc.} & |\phi\rangle(\langle\psi|\alpha\rangle)\langle\beta| & \xlongequal{scalar} & (\langle\psi|\alpha\rangle)(|\phi\rangle\langle\beta|) \\ \textit{view:} & (|\phi\rangle \circ \langle\psi|) \circ (|\alpha\rangle \circ \langle\beta|) & & |\phi\rangle \circ (\langle\psi| \circ |\alpha\rangle) \circ \langle\beta| & & (\langle\psi|\alpha\rangle)(|\phi\rangle \circ \langle\beta|) \end{array}$$

Labelled Dirac notation is an enhancement of Dirac notation that uses subscripts to identify the subsystems where a ket/bra/etc lies or operates; for example, $|\phi\rangle_S$, ${}_S\langle\psi|$, ${}_S\langle\phi|\psi\rangle_S$ and $|\phi\rangle_S\langle\psi|$ range over states, co-states, inner products and outer products on subsystem S, where S is a set of labels drawn from some fixed global set L. This notation has two benefits: first, it makes it possible to describe subsystems of a quantum system, without the need of lifting the subsystem to the global state by means of tensor products. This makes the description of quantum subsystems much more concise and easier to manipulate in calculations. Second, it exposes identities that are commonly used in the calculation. One such identity is commutativity of tensor product, i.e. $|\phi\rangle_S|\psi\rangle_{S'} = |\psi\rangle_{S'}|\phi\rangle_S$ provided S and S' are disjoint.

Expressions based on labelled Dirac notation admit a local interpretation. Specifically, we can define in addition to the global state space $\mathcal{H}_S = \bigotimes_{p \in S} \mathcal{H}_p$ for every $S \subseteq L$; see Figure 2. Then one can interpret expressions based on labelled Dirac notations as linear maps between two local state spaces. This interpretation uses lifting to extend operations from some subspace to a cylindrical extension of that space.

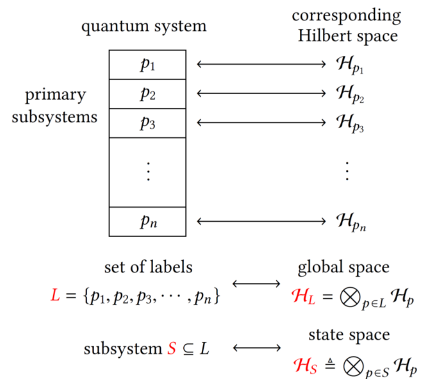

Fig. 2. Global vs local state space

Definition 4.1 (Cylindrical Extension). *Given two subsets $S \subseteq T \subseteq L$ and a linear operator $A \in \mathcal{L}(\mathcal{H}_S)$, we define the cylindrical extension of A_S in T as $cl_T(A) \triangleq A_S \otimes I_{T\setminus S}$. If $T = L$, we simply write $cl(A)$ for $cl_L(A)$.*

To illustrate the use of cylindrical extensions, suppose that we have two disjoint subsystems S_1 and S_2 with initial

state $|\phi\rangle_{S_1}|\psi\rangle_{S_2}$, and we apply the unitary transformation U_{S_1} to S_1. To model the result of this computation, one can first lift U_{S_1} to $cl_{S_1\cup S_2}(U_{S_1}) = U_{S_1} \otimes I_{S_2}$ (operator on $\mathcal{H}_{S_1\cup S_2}$) and then apply the resulting map to the initial state:

$$(U_{S_1} \otimes I_{S_2})(|\phi\rangle_{S_1}|\psi\rangle_{S_2}) = (U_{S_1}|\phi\rangle_{S_1})(I_{S_2}|\psi\rangle_{S_2}) = (U_{S_1}|\phi\rangle_{S_1})|\psi\rangle_{S_2}.$$

However, since we have already labelled all subsystems in the formula, such a lifting step can be automatically identified via the context, and thus, we might simply write:

$$U_{S_1}(|\phi\rangle_{S_1}|\psi\rangle_{S_2}) = (U_{S_1}|\phi\rangle_{S_1})|\psi\rangle_{S_2} \tag{3}$$

This notation captures the intuition that we apply U_{S_1} locally to $|\phi\rangle_{S_1}$ while keeping $|\psi\rangle_{S_2}$ unchanged. Our interpretation of labelled Dirac notation obeys the principle of "*localizing objects as much as possible*".

Example 4.2. *Suppose S and T are two disjoint subsystems with orthonormal basis $\{|v_i\rangle_S\}_{i\in J}$ and $\{|u_i\rangle_T\}_{i\in J}$ respectively. We define the injection of a function $A : J \times J \to \mathbb{C}$ to a linear operator on S (or, T) by $A[S] \triangleq A(i,j)|v_i\rangle_S\langle v_i|$ (or, $A[T] \triangleq A(i,j)|u_i\rangle_T\langle u_i|$). Let $|\Phi\rangle$ be the maximally entangled state on S and T, i.e., $|\Phi\rangle = \sum_{i\in J} |v_i\rangle_S|u_i\rangle_T$. Then, for any A, applying $A[S]$ on $|\Phi\rangle$ yields the same states if we perform $A^T[T]$ on $|\Phi\rangle$ where $A^T(i,j) = A(j,i)$. This can be proved using labelled Dirac notation:*

$$\begin{aligned}
A[S]|\Phi\rangle &= \sum_{mn} A(m,n)|v_m\rangle_S\langle v_n| \sum_i |v_i\rangle_S|u_i\rangle_T = \sum_{mni} A(m,n)_S\langle v_n|v_i\rangle_S|v_m\rangle_S|u_i\rangle_T \\
&= \sum_{mi} A(m,i)|v_m\rangle_S|u_i\rangle_T = \sum_{mij} A(j,i)_T\langle u_j|u_m\rangle_T|v_m\rangle_S|u_i\rangle_T \\
&= \sum_{mij} A(j,i)|u_i\rangle_T\langle u_j|\big(|u_m\rangle_T|v_m\rangle_S\big) = \sum_{ij} A(j,i)|u_i\rangle_T\langle u_j| \sum_m |v_m\rangle_S|u_m\rangle_T \\
&= A^T[T]|\Phi\rangle
\end{aligned}$$

This derivation showcases the benefits of labelled Dirac notation: with the help of blue labels, we can quickly identify which (co-)states can be composed and rearranged (using associativity or scalar property) without worrying about the order of tensor product (such as in the third line). Furthermore, note that in the first line, we do not need to write out the lifting details ($A[S]$ should be lifted to $A[S] \otimes I_T$). Instead we directly identify that $|v_m\rangle_S\langle v_n|$ will act on $|v_i\rangle_S$ while leaving $|u_i\rangle_T$ unchanged.

5 SYNTAX AND SEMANTICS OF QWHILE

In this section, we present the syntax and denotational semantics of **qwhile**, a core language that follows the classical control and quantum data paradigm.

5.1 Syntax

We first define an abstract syntax of **qwhile** programs. The specificity of this syntax is that it does not use variables. Later, we show how to instantiate the language and its semantics to programs with named variables.

Definition 5.1 (Abstract syntax). *Quantum programs are generated by the following syntax:*

$$\begin{aligned}
C ::= \mathbf{abort} \mid \mathbf{skip} \mid C_1; C_2 \mid \mathbf{init}\ \rho_S \mid \mathbf{apply}\ U_S \mid \\
\mathbf{if}\ (\square m \cdot \mathcal{M}_S = m \to C_m)\ \mathbf{fi} \mid \mathbf{while}\ \mathcal{M}'_S = b\ \mathbf{do}\ C\ \mathbf{od}
\end{aligned} \tag{4}$$

where b is a boolean value, S ranges over subsets of a set L of labels, ρ_S, U_S, $\mathcal{M}_S$ and $\mathcal{M}'_S$ range respectively over density operators, unitary operators, quantum measurements on $\mathcal{H}_S$, and two-value measurements (i.e., $\mathcal{M}'_S = \{M_0, M_1\}$).

The initialization **init** ρ_S resets the subsystem S to quantum state ρ_S. The unitary transformation **apply** U_S apply U_S on subsystem S. The **if**-statement **if** $(\square m \cdot \mathcal{M}_S = m \to C_m)$ **fi** first performs quantum measurement $\mathcal{M}_S = \{M_m\}$ on subsystem S and then executes the subprogram C_m according to the outcome m. For the **while**-statement **while** $\mathcal{M}'_S = b$ **do** C **od**, if the outcome of two-value measurement $\mathcal{M}'_S$ on subsystem S is $\neg b$ then the program terminates; otherwise, the program executes subprogram C and then repeats the loop again.

For convenience, we further define the syntactic sugar for sequential programs:

$$\textbf{for } i \leftarrow J \textbf{ do } C_i \equiv C_{i_0}; C_{i_1}; \cdots; C_{i_n} \tag{5}$$

where $J = \{i_0, i_1, \cdots, i_n\}$ is a sequence of indexes. We simply write **for** $i < n$ **do** C_i if i ranges over $0, 1 \cdots, n-1$.

Remark 5.1 (Metaprogramming). *We mainly focus on the quantum variables and commands. As a consequence, we will use metaprogramming to introduce classical variables/parameters; this may be regarded as parameterized quantum circuits (such as QFT and HLF) and programs (such as HSP and HHL).*

5.2 Semantics

The denotational semantics of programs is defined inductively on their structure. The semantics of loops is based on the (syntactic) notion of approximation.

Definition 5.2 (Syntactic Approximation of While; c.f. [Ying 2011]). *For a given quantum loop* **while** $\mathcal{M}_S = b$ **do** C **od**, *we inductively define its k-th syntactic approximation*

$$\textbf{while}^{(k)}\ \mathcal{M}_S = b \textbf{ do } C \textbf{ od}$$

for any integer $k \geq 0$ *as follows:*

$$\begin{cases} \textbf{while}^{(0)}\ \mathcal{M}_S = b \textbf{ do } C \textbf{ od} & \equiv \textbf{abort} \\ \textbf{while}^{(k+1)}\ \mathcal{M}_S = b \textbf{ do } C \textbf{ od} & \equiv \textbf{if } \mathcal{M}_S = \neg b \ \to \ \textbf{skip} \\ & \qquad \square \qquad\ b \ \to \ C; \textbf{while}^{(k)}\ \mathcal{M}_S = b \textbf{ do } C \textbf{ od fi} \end{cases}$$

The semantics characterizes the evolution of quantum systems according to the high-level description of the commands given in Section 5.1.

Definition 5.3 (Semantics). *The semantics* $[\![C]\!]$ *of a quantum program* C *is a super-operator on* $\mathcal{H}_L$ *(that is, linear map from* $\mathcal{L}(\mathcal{H}_L)$ *to* $\mathcal{L}(\mathcal{H}_L)$*) inductively defined as follows:*

(1) $[\![\textbf{abort}]\!] \triangleq 0$; *i.e.,* $[\![\textbf{abort}]\!] : \rho \mapsto 0$;
(2) $[\![\textbf{skip}]\!] \triangleq \mathcal{I}_L$; *i.e.,* $[\![\textbf{skip}]\!] : \rho \mapsto \rho$;
(3) $[\![C_1; C_2]\!] \triangleq [\![C_1]\!] \circ [\![C_2]\!]$; *i.e.,* $[\![C_1; C_2]\!] : \rho \mapsto [\![C_1]\!]([\![C_2]\!](\rho))$;
(4) $[\![\textbf{init } \rho_S]\!] : \rho \mapsto \mathrm{tr}_S(\rho) \otimes \rho_S$;
(5) $[\![\textbf{apply } U_S]\!] : \rho \mapsto U_S \rho U_S^\dagger$;
(6) $[\![\textbf{if } (\square m \cdot \mathcal{M}_S = m \to C_m) \textbf{ fi}]\!] : \rho \mapsto \sum_m [\![C_m]\!](M_m \rho M_m^\dagger)$;
(7) $[\![\textbf{while } \mathcal{M}_S = b \textbf{ do } C \textbf{ od}]\!] \triangleq \lim_{k\to\infty} [\![\textbf{while}^{(k)}\ \mathcal{M}_S = b \textbf{ do } C \textbf{ od}]\!]$.

Note that we use labelled Dirac notation for defining the semantics of unitary transformations; in detail, $U_S \rho U_S^\dagger \triangleq cl(U_S) \circ \rho \circ cl(U_S^\dagger)$. *We also use labelled Dirac notation* $M_m \rho M_m^\dagger$ *for defining the semantics of conditionals and loops.*

The command **abort** maps any state ρ to 0, i.e. as a program that never terminates (e.g. as the program **while** true **do skip od**); **skip** has no effect on the state and corresponds to the identity map; **init** ρ_S resets the subsystem S to state ρ_S, while leaving the rest part unchanged (recall that the subsystem is described by the partial trace); **apply** U_S says that the whole system

evolves following the unitary transformation $cl(U_S)$. For the **if** statement, the state evolves to $[\![C_m]\!]\left(\frac{M_m\rho M_m^\dagger}{\mathrm{tr}(M_m^\dagger M_m\rho)}\right)$ with probability $\mathrm{tr}(M_m^\dagger M_m\rho)$ (where $\frac{M_m\rho M_m^\dagger}{\mathrm{tr}(M_m^\dagger M_m\rho)}$ is the post-measurement state before executing C_m); this is a standard quantum ensemble, and thus the final state can be described by $\sum_m p(m)[\![C_m]\!]\left(\frac{M_m\rho M_m^\dagger}{\mathrm{tr}(M_m^\dagger M_m\rho)}\right) = \sum_m [\![C_m]\!](M_m\rho M_m^\dagger)$ by linearity. Finally, the semantics of **while** statement is defined as the limit of the semantics of its syntactic approximations.

Note that the limit in (7) always exists according to monotone convergence theorems on ordered Hilbert spaces—where the order is Löwner order. Note that our semantics contrasts with classic domain-theoretical semantics which interprets loops as a least fixed point over the complete partial order (CPO) of quantum operations (recall that a quantum operation is completely-positive and trace-nonincreasing super-operator). However, one can show that $[\![C]\!]$ is a quantum operation and that our semantics coincides with the domain-theoretic semantics. The proof of equivalence is based on the following lemma, which states that the supremum in the CPO of quantum operations is consistent with the topological limits of super-operators:

$$\forall \text{ non-decreasing sequence } \mathcal{E}_i \in QO(\mathcal{H}),\ \bigsqcup_i \mathcal{E}_i = \lim_{i\to\infty} \mathcal{E}_i.$$

This choice of defining the semantics of programs as super-operators rather than quantum operations is motivated by usability considerations. First, super-operators enjoy linear algebraic properties that are not verified by quantum operations—see Section 3.1. This makes our current formalization easier to develop, and more importantly easier to use; see Remark 6.1 for a similar explanation. Moreover, such choice is *safe* in the sense that the semantics of commands are quantum operations that have been proven in CoqQ.

A key aspect of our semantics is that it naturally induces a local semantics over the subsystem defined by its variables. This is captured by the following lemma, which is an adaptation of [Ying 2016, Proposition 3.3.5] to our setting.

Lemma 5.1 (Localization of semantics). *Suppose C is a quantum program, and let $set(C)$ be the union of all subsystems mentioned in C. There uniquely exists a local super-operator $[\![C]\!]_l$: $\mathcal{L}(\mathcal{H}_{set(C)}) \to \mathcal{L}(\mathcal{H}_{set(C)})$ such that $[\![C]\!] = [\![C]\!]_l \otimes \mathcal{I}_{\overline{set(C)}}$ where $\mathcal{I}_{\overline{set(C)}}$ is the identity super-operator on subsystem $\overline{set(C)} \triangleq L\backslash set(C)$. We call $[\![C]\!]_l$ the local semantics of C.*

This localization property plays an essential role in the proof of the soundness of Hoare logic. It states that the program C will influence at most those subsystems entangled with $\mathrm{set}(C)$, while leaving those uncorrelated subsystems unchanged.

5.3 Concrete Syntax

Although the abstract syntax suffices to develop the meta-theory of **qwhile**, the formalization of textbook algorithms and their correctness proofs is greatly simplified by switching to a concrete syntax with support for typed variables. In order to maximize convenience, the introduction of variables must satisfy two desiderata: first, variables should be given arbitrary types, rather than being limited to represent qubits; second, variables should be closed under Cartesian products and projections, so that it is sufficient to use the following (selected) syntax:

$$\mathsf{x} := |\mathsf{t}\rangle \;\big|\; \mathsf{x} := U \;\big|\; \textbf{if } \mathrm{meas}[\mathsf{x}] = \mathsf{t} \to C_{\mathsf{t}} \textbf{ fi} \;\big|\; \textbf{while } \mathrm{meas}[\mathsf{x}] = b \textbf{ do } C \textbf{ od}$$

for initialization, unitary transformation, if and while statements using computational basis measurement as guards, respectively.

Instantiating the semantics of abstract programs to concrete programs requires several steps:

- defining the Hilbert space associated with a (finite) set with its elements as the computational basis;
- modelling quantum variables. A quantum variable $\mathtt{x}$ of type $\mathtt{T}$ associates its symbolic name and a storage location (quantum subsystem $\mathrm{set}(\mathtt{x})$), together with the mapping between the associated Hilbert space of $\mathtt{T}$ and state space of its quantum subsystem. We write $|u\rangle_{\mathtt{x}} \in \mathcal{H}_{\mathrm{set}(\mathtt{x})}$ and $A[\mathtt{x}] \in \mathcal{L}(\mathcal{H}_{\mathrm{set}(\mathtt{x})})$ for the injection of $\mathbf{u} \in \mathcal{H}_{\mathtt{T}}$ and $A \in \mathcal{L}(\mathcal{H}_{\mathtt{T}})$ respectively.
- modelling composition of quantum variables. Suppose two quantum variables $\mathtt{x}_1$ of type $\mathtt{T}_1$ and $\mathtt{x}_2$ of type $\mathtt{T}_2$ with disjoint domains (i.e., $\mathrm{set}(\mathtt{x}_1) \cap \mathrm{set}(\mathtt{x}_2) = \emptyset$), we manipulate the pair $\overline{\mathtt{x}} \triangleq [\mathtt{x}_1,\mathtt{x}_2]$ as a quantum variable of type $\mathtt{T}_1 * \mathtt{T}_2$ which consistent with actions on its components.

6 PROGRAM LOGIC

CoqQ implements a program logic that is proved sound with respect to the denotational semantics of **qwhile** programs.

6.1 Judgments

Correctness of **qwhile** programs is expressed using Hoare triples. A (mild) novelty of our approach is that we allow assertions to be arbitrary linear operators rather than quantum predicates (effects). This simplifies the proof rules since linear operators have better algebraic properties than operators. Of course, our notion of validity coincides with the usual notion of validity when the pre- and postconditions are quantum predicates.

Definition 6.1 (Valid judgment). *A Hoare triple is a judgment of the form $\{A_{S_1}\}C\{B_{S_2}\}$ with $S_1, S_2 \subseteq L$ and $A \in \mathcal{L}(\mathcal{H}_{S_1}),\ B \in \mathcal{L}(\mathcal{H}_{S_2})$. The validity of a Hoare triple is defined as follows (recall Definition 4.1 for cylindrical extension):*

- *total correctness: $\models_{\mathrm{t}} \{A_{S_1}\}C\{B_{S_2}\}$ if for all $\rho \in \mathcal{D}(\mathcal{H}_L)$,*

$$\mathrm{tr}[cl(A_{S_1}) \circ \rho] \le \mathrm{tr}[cl(B_{S_2}) \circ [\![C]\!](\rho)]; \tag{6}$$

- *partial correctness: $\models_{\mathrm{p}} \{A_{S_1}\}C\{B_{S_2}\}$ if for all $\rho \in \mathcal{D}(\mathcal{H}_L)$,*

$$\mathrm{tr}[cl(A_{S_1}) \circ \rho] \le \mathrm{tr}[cl(B_{S_2}) \circ [\![C]\!](\rho)] + [\mathrm{tr}(\rho) - \mathrm{tr}([\![C]\!](\rho))]. \tag{7}$$

Recall that $\mathrm{tr}[cl(A_S) \circ \rho] = \mathrm{tr}(A_S \circ \rho|_S)$ is the expectation of A_S in local state of ρ in subsystem S. Therefore, $\models_{\mathrm{t}} \{A_{S_1}\}C\{B_{S_2}\}$ says that the expectation of A_{S_1} in local input of subsystem S_1 is smaller than the expectation of B_{S_2} in local output of subsystem S_2. The validity of partial correctness is stated similarly, but the probability of non-termination $\mathrm{tr}(\rho) - \mathrm{tr}([\![C]\!](\rho))$ is added to the RHS of the inequality.

In some circumstances, it is desirable to establish a stronger notion of correctness, where the inequalities in Eqn. (6) or (7) are replaced by =. We use the superscript "s" (for saturated) to indicate that the equality holds, e.g., $\models_{\mathrm{t}}^{\mathrm{s}}$ for saturated total correctness. We remark that deriving saturation brings stronger results, i.e., the equivalence (rather than inequality) of pre- and post-expectation, as used in the HSP algorithm, while requiring no extra effort since most rules apply to both saturated and non-saturated cases.

Finally, it is often convenient to use a special form of judgments $\{AA^\dagger\}C\{BB^\dagger\}$, for which we introduce the following syntactic sugar:

$$\models \{\!|A|\!\}C\{\!|B|\!\}.$$

This notation is particularly useful when pre- and postconditions are expressed as states in labelled Dirac notation. In this case, $\models \{\!|\,|u\rangle_{S_u}|\!\}C\{\!|\,|v\rangle_{S_v}|\!\}$ holds whenever the program C transforms the input state $|u\rangle_{S_u}$ into the output state $|v\rangle_{S_v}$—under the proviso that $\||u\rangle_{S_u}\| = \||v\rangle_{S_v}\| = 1$. This

provides a *human-readable* statement that looks like textbook statements of program correctness. This readability extends not only to statements but also to proofs (see Fig. 7 for an example). In particular, using labelled Dirac notation as pre- and postconditions allows us to write, interpret, derive and calculate assertions locally, using the equational theory attached to the notation.

Remark 6.1. *Using linear operators as assertions benefits the usability of CoqQ (denoted by +) but suffers from certain drawbacks compared to quantum predicates and projections (denoted by −):*

+ *It is expressive. Judgments in [Ying 2011, 2019; Ying et al. 2018, 2022] are special cases of our judgments. By establishing Theorem 3.2 and 3.3 in [Zhou et al. 2019], our judgment can easily convert to projection judgments adopted in [Unruh 2019a; Zhou et al. 2019] and vice versa.*
+ *It saves us a lot of proof obligations in practice. For instance, when reasoning about summation or scalar multiplications, we do not have to prove any side condition, whereas a formalization based on quantum predicates/projections would require that all intermediate assertions are quantum predicates/projections.*
− *We need to check that pre- and postconditions are Hermitian before rephrasing the valid judgment to a readable/physical interpretation. Generally, the value $tr(A \circ \rho)$ in Eqn. (6)/(7) is meaningless if A is not Hermitian (A stands for some predicate and ρ the state).*
− *It requires some extra side conditions compared to quantum predicates/projections in a few cases such as rules (Ax.Inv) and (Frame.P) in Fig. 3.*
− *We cannot benefit from the advantages of using quantum logic (i.e., the logic that uses projections as atomic propositions) as assertion logic (see [Zhou et al. 2019] for a summary of such advantages).*

6.2 Inference Rules

Our proof system provides a rich set of inference rules for reasoning about valid Hoare triples. Proof rules are divided into several categories: syntax-directed rules, which are specific to one program construct, including custom rules for **for** loops; structural rules, which can be interspersed with syntax-directed rules to ease/enable reasoning; state-based rules, which are based on the representation of assertions as states in labelled Dirac notation. Rules from the first and second categories are inspired by existing literature. However, rules from the third category, notably the rule (R.Inner), seem new. Fig. 3 presents a selection of forward rules for partial correctness based on the concrete syntax.

Syntax-directed rules. The rules (Ax.Sk), (Ax.InF), (Ax.UTF), (R.SC), (R.IF) and (R.LP.P) are used to reason about the core constructs of the **qwhile** language. In addition, the rules (R.PC.T), (Ax.UTFP) and (Ax.InFP) are used to reason about **for** loops. programs. These rules have the side condition of "disjointness" – assume that the programs (predicates) are about different quantum subsystems.

Structural rules. Structural rules are not tied to a specific language construct and can be interspersed with other rules to simplify correctness proofs. Rule (R.CC.P) simplifies correctness proofs by enabling linear combinations of pre- and postconditions. Rule (Ax.Inv) says that any predicate disjoint from the footprint of the program is preserved. (R.EI) allows us to lift predicates to a larger space and can be regarded as using "auxiliary variables". Finally, (Frame.P) allows to focus on the part of the predicates that are related to the program while keeping the rest part unchanged.

State-based rules. The rules (Ax.UTF') and (Ax.InF') are state-based variants of the rules (Ax.UTF) and (Ax.InF). Besides the human-readability, these rules reduce the calculations on labelled Dirac notation by half; for example, compare to the rule (Ax.UTF); in this case, the postcondition is $(U[\mathrm{x}]|v\rangle_S)\underline{({}_S\langle v|U[\mathrm{x}]^\dagger)}$; note that we do not need to calculate the second part (labelled by the underline) since it is just the adjoint of the first part (without underline). Most of the rules have

$$\text{(Ax.Sk)}\ \models_{\mathrm{p}} \{A\}\mathbf{Skip}\{A\} \qquad \text{(Ax.UTF)}\ \models_{\mathrm{p}} \{A\}\mathsf{x} := U[\mathsf{x}]\{U[\mathsf{x}]AU[\mathsf{x}]^{\dagger}\}$$

$$\text{(Ax.InF)}\ \frac{S \cap \mathrm{set}(x) = \emptyset}{\models_{\mathrm{p}} \{A_S\}\mathsf{x} := |\mathsf{t}\rangle\{A_S \otimes |\mathsf{t}\rangle_{\mathsf{x}}\langle \mathsf{t}|\}} \qquad \text{(R.SC)}\ \frac{\models_{\mathrm{p}} \{A\}S_1\{B\} \quad \models_{\mathrm{p}} \{B\}S_2\{C\}}{\models_{\mathrm{p}} \{A\}S_1; S_2\{C\}}$$

$$\text{(R.IF)}\ \frac{\models_{\mathrm{p}} \{A_{\mathsf{t}}\}C_{\mathsf{t}}\{B\} \text{ for all } \mathsf{t}}{\models_{\mathrm{p}} \left\{ \sum_{\mathsf{t}:\mathsf{T}} |\mathsf{t}\rangle_{\mathsf{x}}\langle \mathsf{t}|A_m|\mathsf{t}\rangle_{\mathsf{x}}\langle \mathsf{t}| \right\}\mathbf{if}\ \mathrm{meas}[\mathsf{x}] = \mathsf{t} \to C_{\mathsf{t}}\ \mathbf{fi}\{B\}}$$

$$\text{(R.LP.P)}\ \frac{R := |b\rangle_{\mathsf{x}}\langle b|A|b\rangle_{\mathsf{x}}\langle b| + |\neg b\rangle_{\mathsf{x}}\langle \neg b|B|\neg b\rangle_{\mathsf{x}}\langle \neg b| \quad \models_{\mathrm{p}} \{A\}C\{R\} \quad A \sqsubseteq I \quad B \sqsubseteq I}{\models_{\mathrm{p}} \{R\}\mathbf{while}\ \mathrm{meas}[\mathsf{x}] = b\ \mathbf{do}\ C\ \mathbf{od}\{B\}}$$

$$\text{(R.PC.P)}\ \frac{\begin{array}{c}\forall i, \models_{\mathrm{p}} \{A_{i,S_{A_i}}\}P_i\{B_{i,S_{B_i}}\} \quad \forall i, 0 \sqsubseteq A_{i,S_{A_i}} \sqsubseteq I_{S_{A_i}} \quad \forall i, 0 \sqsubseteq B_{i,S_{B_i}} \sqsubseteq I_{S_{B_i}} \\ \forall i \neq j, (\mathrm{set}(C_i) \cup S_{A_i} \cup S_{B_i}) \cap (\mathrm{set}(C_j) \cup S_{A_j} \cup S_{B_j}) = \emptyset\end{array}}{\models_{\mathrm{p}} \{\bigotimes_i A_{i,S_{A_i}}\}\mathbf{for}\ i\ \mathbf{do}\ C_i\{\bigotimes_i B_{i,S_{B_i}}\}}$$

$$\text{(Ax.UTFP)}\ \frac{\forall i \neq j, \mathrm{set}(\mathsf{x}_i) \cap \mathrm{set}(\mathsf{x}_j) = \emptyset}{\models_{\mathrm{p}} \{A\}\mathbf{for}\ i\ \mathbf{do}\ \mathsf{x}_i := U_i[\mathsf{x}_i]\{(\bigotimes_i U_i[\mathsf{x}_i])A(\bigotimes_i U_i[\mathsf{x}_i])^{\dagger}\}}$$

$$\text{(Ax.InFP)}\ \frac{\forall i \neq j, \mathrm{set}(\mathsf{x}_i) \cap \mathrm{set}(\mathsf{x}_j) = \emptyset}{\models_{\mathrm{p}} \{1\}\mathbf{for}\ i\ \mathbf{do}\ \mathsf{x}_i := |\mathsf{t}_i\rangle\{\bigotimes_i |\mathsf{t}_i\rangle_{\mathsf{x}_i}\langle \mathsf{t}_i|\}}$$

$$\text{(R.Or)}\ \frac{A \sqsubseteq A' \quad \models_{\mathrm{p}} \{A'\}C\{B'\} \quad B' \sqsubseteq B}{\models_{\mathrm{p}} \{A\}C\{B\}} \qquad \text{(R.CC.P)}\ \frac{\forall i, \models_{\mathrm{p}} \{A_i\}C\{B_i\} \quad \forall i, 0 \leq \lambda_i \quad \sum_i \lambda_i \leq 1}{\models_{\mathrm{p}} \{\sum_i \lambda_i A_i\}C\{\sum_i \lambda_i B_i\}}$$

$$\text{(Ax.Inv)}\ \frac{A_S \sqsubseteq I_S \quad S \cap \mathrm{set}(C) = \emptyset}{\models_{\mathrm{p}} \{A_S\}C\{A_S\}} \qquad \text{(R.El)}\ \frac{\models_{\mathrm{p}} \{A_{S_A}\}C\{B\} \quad S_A \cap S = \emptyset}{\models_{\mathrm{p}} \{A_{S_A} \otimes I_S\}C\{B\}}$$

$$\text{(Frame.P)}\ \frac{\models_{\mathrm{p}} \{A_{S_A}\}C\{B_{S_B}\} \quad 0 \sqsubseteq R_S \sqsubseteq I_S \quad (\mathrm{set}(C) \cup S_A \cup S_B) \cap S = \emptyset}{\models_{\mathrm{p}} \{A_{S_A} \otimes R_S\}C\{B_{S_B} \otimes R_S\}}$$

$$\text{(R.Inner)}\ \frac{\models_{\mathrm{t}}^{\mathrm{s}} \{1\}C\{|v\rangle_{S_v}\langle v|\} \quad \||v\rangle_{S_v}\| \leq 1 \quad S_u \subseteq S_v}{\models_{\mathrm{t}}^{\mathrm{s}} \{\|_{S_u}\langle u|v\rangle_{S_v}\|^2\}C\{|u\rangle_{S_u}\langle u|\}}$$

$$\text{(Ax.UTF}')\ \models_{\mathrm{p}} \{\!\{|v\rangle_S\}\!\}\mathsf{x} := U[\mathsf{x}]\{\!\{U[\mathsf{x}]|v\rangle_S\}\!\} \qquad \text{(Ax.InF}')\ \frac{S \cap \mathrm{set}(x) = \emptyset}{\models_{\mathrm{p}} \{\!\{|v\rangle_S\}\!\}\mathsf{x} := |\mathsf{t}\rangle\{\!\{|v\rangle_S \otimes |\mathsf{t}\rangle_{\mathsf{x}}\}\!\}}$$

Fig. 3. Selected inference rules provided in CoqQ. In (R.Inner), $\models_{\mathrm{t}}^{\mathrm{s}}$ stands for the saturated total correctness, i.e., the inequality in Eqn. (6) is replaced by "=".

such variants (which we label their name by a single quote); we do not display them here due to space limitations.

The new rule (R.Inner) provides a way to link forward and backward reasoning, i.e., it derives the precondition for a given postcondition from a judgment derived by forward reasoning. Forward reasoning is more suitable for programs with fixed inputs (or starting with initialization), and is usually intuitive and relatively simple; backward reasoning is suitable when the postcondition is given, but with a possibly much more involved computation. HSP algorithm is such an example; see Section 8.1.

6.3 Soundness and Weakest Precondition

All inference rules are formalized as lemmas and proved from first principles. Formally, we claim:

Theorem 6.1 (Soundness). *All the inference rules displayed in Section 6.2 are sound with respect to the judgment defined in Definition 6.1.*

In addition to the soundness proof, we show that for all quantum predicates in $O(\mathcal{H}_L)$, the weakest (liberal) precondition is well-defined, i.e., and satisfies $\models \{A\}C\{B\}$ if and only if $A \sqsubseteq wp.C.B$ for total correctness (or $wlp.C.B$ for partial correctness).

7 IMPLEMENTATION

All the aforementioned concepts have been formalized in the Coq [The Coq Development Team 2022] proof assistant. Our formalization relies on the MathComp [Mahboubi and Tassi 2021] library for basic data structures such as finite sets, ordered fields, linear algebra, and vector spaces, and on the MathComp Analysis [The MathComp Analysis Development Team 2022] library, an ongoing attempt at providing a library for classical analysis in Coq that is compatible with the mathematical objects introduced in the MathComp library.

Following an established approach popularized by MathComp and MathComp Analysis, we leverage the power of canonical structures to ease formalization and reasoning about mathematical objects. The main benefits of canonical structures include notation overloading, transparent navigation along the mathematical hierarchy, e.g. using the coercion mechanism supported by Coq, and several forms of proof automation. We refer the reader to [Gonthier et al. 2013] for a more general discussion.

7.1 Mathematical Libraries

Finite Dimensional Hilbert Spaces. MathComp library has an extensive theory about finite dimensional linear spaces and linear algebra. However, it lacked a formalization of Hilbert spaces, which are essential for modelling quantum states. Hence, we enriched the MathComp hierarchy with a new type `hermitianType R`, for `R` a real domain, that extends the MathComp defined type `vectType R[i]` of finite dimensional linear spaces over the complex closure of `R`. This new structure comes with a hermitian inner product [< . , . >], i.e. a sesquilinear form over `R[i]`. We also formalized the core theory of Hermitian spaces. Notably, we defined and proved correct the Gram–Schmidt process that allows the orthonormalization of a set of vectors w.r.t. an inner product. Last, our library comes with a sub-type `chsType R`, of `hermitianType R`, that denotes Hermitian spaces that come with an orthonormal canonical basis. The latter is useful to convert between abstract vectors and their concrete matrix forms, and hence to compute inner products by matrix multiplication.

Tensor Product & Hilbert Spaces. Our library comes with a definition of the tensor product of linear spaces. However, instead of defining the tensor product of *two* linear spaces only, we define the tensor product directly over a finitely indexed family of linear spaces. Our formal definition of tensor products relies on the basis-based definition of tensor products: if $\{E_i\}_i$ is a finite family of linear spaces over a field k with respective basis $\{e_{i,j}\}_j$, then $\bigotimes_i E_i$ is a vector space with the formal basis $\{e_{1,j_1} \otimes e_{2,j_2} \otimes \cdots \otimes e_{n,j_n}\}_{j_1,j_2,\ldots,j_n}$.

The primary motivation to define tensor products over finitely indexed families is to deal with nested tensor products. Consider for instance the spaces $(E \otimes F) \otimes G$ and $E \otimes (F \otimes G)$. These two spaces are isomorphic—more generally, for a fixed field k and up to linear spaces isomorphism, the tensor product operator forms a commutative monoid over k-linear spaces. Using binary products would introduce the need to handle this isomorphism explicitly. Using finitely indexed families minimizes the need to deal with such isomorphisms.

Our library also comes with a notion of multi-linear map over a finitely indexed family of linear spaces $\{E_i\}_i$ and a formal proof of the universal property of tensor products: any multi-linear map over $\bigtimes_i E_i$ can be canonically lifted to a linear map over $\bigotimes_i E_i$. Finally, we show that the tensor product of Hilbert spaces is itself a Hilbert space.

Vectors and Linear Operators. Recall that for a finite set of symbols L and a family of linear spaces $\{\mathcal{H}_x\}_{x\in\mathsf{L}}$, the Hilbert space of any subsystem $\mathsf{S} \subseteq \mathsf{L}$ is defined as H_S$\triangleq \bigotimes_{x\in\mathsf{S}} \mathcal{H}_x$[4]. Moreover, recall that vectors and linear operators are defined as type constructors taking their respective domain and codomain as parameters. E.g., for $\mathsf{S},\mathsf{T} \subseteq \mathsf{L}$, the type of vectors in the subsystem S is H_S, while the type of linear maps from S to T is F_(S,T). An interesting fact is that $\mathcal{H}_\emptyset$ is isomorphic to $\mathbb{C}$, which makes the linear map view of states and co-states in the subsystem S consistent with F_(∅,S) and F_(S,∅) respectively. We further define the tensor product for vectors and operators. For instance, if f_1 : F_($\mathsf{S}_1,\mathsf{T}_1$) and f_2 : F_($\mathsf{S}_2,\mathsf{T}_2$) are two operators, their tensor product $\mathsf{f}\otimes\mathsf{g}$ is defined as the type F_($\mathsf{S}_1\cup\mathsf{S}_2,\mathsf{T}_1\cup\mathsf{T}_2$). In principle, a labelled Dirac notation is interpreted as a linear map and can be modelled by the above definitions. However, these definitions are dependent on the domains & co-domains of the underlying objects. Unfortunately, it is well-known that equational reasoning about dependently typed objects is unwieldy, due to the common need to use type casts. For example, in our concrete setting, $\mathsf{f}_1\otimes\mathsf{f}_2$ and $\mathsf{f}_2\otimes\mathsf{f}_1$ have the respective incompatible types F_($\mathsf{S}_1\cup\mathsf{S}_2,\mathsf{T}_1\cup\mathsf{T}_2$) and F_($\mathsf{S}_2\cup\mathsf{S}_1,\mathsf{T}_2\cup\mathsf{T}_1$), and thus a type cast is needed to coerce their type and make them comparable.

Labelled Dirac Notation. To avoid the type cast, our library comes up with a definition of labelled Dirac notation as a non-dependent type which is carefully designed to have linear algebraic structure. Roughly speaking, a Dirac expression (QE in CoqQ) is a variant of mapping ∀ S T ⊆ L, F_(S,T); suppose e : QE, then for any $\mathsf{S},\mathsf{T} \subseteq \mathsf{L}$, e S T : F_(S,T) returns the linear map from subsystem S to T. We encode an arbitrary linear map f :F_(S,T) to a Dirac expression ⌈f⌉ defined as follows:

$$\mathsf{f} : \mathsf{F_(S,T)} \;\; \underset{\text{decode}}{\overset{\text{encode}}{\rightleftharpoons}} \;\; \lceil \mathsf{f} \rceil : \mathsf{QE} \triangleq \begin{array}{c} \\ \emptyset \\ \vdots \\ \mathsf{S} \\ \vdots \\ \mathsf{L} \end{array} \begin{array}{c} \begin{array}{ccccc} \emptyset & \cdots & \mathsf{T} & \cdots & \mathsf{L} \end{array} \\ \begin{bmatrix} 0 & \cdots & 0 & \cdots & 0 \\ & \ddots & \vdots & \ddots & \\ 0 & \cdots & \mathsf{f} & \cdots & 0 \\ & \ddots & \vdots & \ddots & \\ 0 & \cdots & 0 & \cdots & 0 \end{bmatrix} \end{array}.$$

That is, ⌈f⌉ S T = f and for any S', T', ⌈f⌉ S T = 0 if S'≠S or T'≠T. We say a Dirac expression e is well-formed with domain S and codomain T if there exists f :F_(S,T) such that e = ⌈f⌉; there is an one-to-one correspondence between non-zero linear maps and non-zero well-formed Dirac expressions. We can define the corresponding unary and binary operators for (well-formed) Dirac expressions which are consistent to the linear maps; for example, $\lceil \mathsf{f}_1\otimes\mathsf{f}_2\rceil = \lceil\mathsf{f}_1\rceil\otimes\lceil\mathsf{f}_2\rceil$ for any linear map $\mathsf{f}_1,\mathsf{f}_2$ (on arbitrary domain and codomain). We can show that $\lceil\mathsf{f}_1\rceil\otimes\lceil\mathsf{f}_2\rceil = \lceil\mathsf{f}_2\rceil\otimes\lceil\mathsf{f}_1\rceil$; we do not need to worry about type cast anymore.

Interestingly, we use the *canonical structure* to trace the (co)domain of a labelled Dirac notation. We provide the canonical instance for each unary/binary operation/big operator of Dirac expression, and thus Coq will automatically infer its (co)domain from its structure. For instance, we define the sub-type `Structure` {wf S, T} for the well-formed Dirac expression with domain S and codomain T; Coq will coerce $\lceil\mathsf{f}_1\rceil\otimes\lceil\mathsf{f}_2\rceil$ to type {wf $\mathsf{S}_1\cup\mathsf{S}_2,\mathsf{T}_1\cup\mathsf{T}_2$} if needed, providing f_i : F_($\mathsf{S}_i,\mathsf{T}_i$) for $i = 1, 2$.

Note that our discussion of labelled Dirac notation has so far been confined to unary and binary operators. However, an important feature of our formalization is that it allows to use big operators for tensor products, i.e. $\bigotimes_i$. This can be done since tensor products have a commutative monoid structure with 1 (i.e., identity function of type F_(∅, ∅)) as the identity element. The use of big operators in labelled Dirac notation is crucial for several of our examples.

[4]In CoqQ, we follow the convention of MathComp that add a quote before the type notation, e.g., 'H_S, 'F_(S,T) and 'QE below.

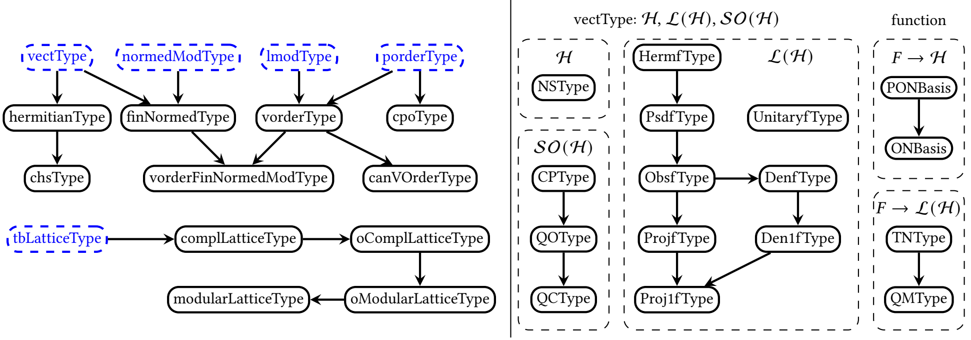

Fig. 4. Hierarchy graph (left) and structures for the quantum framework (right). The types inside the dashed box (coloured blue) are types in MathComp and MathComp Analysis.

Concepts for Quantum Framework. As instances of `vectType`, Hilbert space $\mathcal{H}$, linear maps $\mathcal{L}(\mathcal{H})$ and super-operator $\mathcal{SO}(\mathcal{H})$ inherits the linear algebraic structure and form the basis of our quantum framework. Besides, we formalize most of the fundamental concepts in quantum mechanics using `Structure`, including normalized state (`NSType`), completely-positive maps (`CPType`), quantum operation (`QOType`), quantum channel (`QCType`), (partial) orthonormal basis (`PONBasis` and `ONBasis`), trace-nonincreasing maps (`TNType`), quantum measurement (`QMType/TPType`), and a series of subsets of linear operators – hermitian, positive-semidefinite, quantum predicate (observable), (partial) density operator, unitary, (rank-1) projection; see Fig. 4. A state/operator/super-operator/function with proof of some property can be declared as a canonical instance of the corresponding definition and then inherits the properties of that definition.

Other Results. We also prove several results that are not covered by MathComp Analysis. These results include the Bolzano-Weierstrass theorem on Euclidean spaces, the monotone convergence theorem for finite-dimensional linear spaces. The latter is directly applied to define the semantics of **qwhile** programs. Finally, we develop a library for matrix norm and subspace theory (represented by projection and declared as a canonical instance of orthomodular lattice).

7.2 Formalization of qwhile and Program Logic

The abstract syntax **qwhile** language is deeply embedded in Coq, i.e., implemented as an inductive type `cmd`. The concrete syntax is implemented as a shallow embedding.

Hilbert Space Associated with a (Finite) Set. The semantics of programs relies on the canonical construction of a Hilbert space from a finite set, and on operators that lift standard set-theoretical constructions, including Cartesian products and finite products (modelled as dependent functions over finite sets), to Hilbert spaces. We briefly explain the two kinds of constructions.

The Hilbert space $\mathcal{H}_{\mathtt{T}}$ associated with a finite type $\mathtt{T}$ is the $|\mathtt{T}|$-dimensional Hilbert space with computational basis $\{\widetilde{\mathtt{t}}\}_{\mathtt{t}\in\mathtt{T}}$ (i.e., $\langle\widetilde{\mathtt{t}_1},\widetilde{\mathtt{t}_2}\rangle = 1$ if $\mathtt{t}_1 = \mathtt{t}_2$ and 0 otherwise; we use tilde to denote that $\widetilde{\mathtt{t}} \in \mathcal{H}_{\mathtt{T}}$ is a state). Any state $\mathbf{u} \in \mathcal{H}_{\mathtt{T}}$ can be written as $\mathbf{u} = \sum_{\mathtt{t}\in\mathtt{T}}\langle\widetilde{\mathtt{t}},\mathbf{u}\rangle\widetilde{\mathtt{t}}$ and every linear operator A on $\mathcal{H}_{\mathtt{T}}$ can be decomposed as $A = \sum_{\mathtt{t}_1,\mathtt{t}_2\in\mathtt{T}}\langle\widetilde{\mathtt{t}_2}, A(\widetilde{\mathtt{t}_1})\rangle[\widetilde{\mathtt{t}_2},\widetilde{\mathtt{t}_1}]$.

Recall that the product type of $\mathtt{T}_1$ and $\mathtt{T}_2$ is denoted by $\mathtt{T}_1 * \mathtt{T}_2$. If $\mathtt{T}_1$ and $\mathtt{T}_2$ are finite types then so is $\mathtt{T}_1 * \mathtt{T}_2$ and furthermore $\mathcal{H}_{\mathtt{T}_1*\mathtt{T}_2}$ is isomorphic to $\mathcal{H}_{\mathtt{T}_1} \otimes \mathcal{H}_{\mathtt{T}_2}$. Based on this isomorphism, we can define the tensor product $\widetilde{\mathtt{t}_1} \otimes \widetilde{\mathtt{t}_2} \triangleq \widetilde{(\mathtt{t}_1,\mathtt{t}_2)} \in \mathcal{H}_{\mathtt{T}_1*\mathtt{T}_2}$ and linearly extend to all states, i.e., for all $\mathbf{u}_1 \in \mathcal{H}_{\mathtt{T}_1}$ and $\mathbf{u}_2 \in \mathcal{H}_{\mathtt{T}_2}$, define: $\mathbf{u}_1 \otimes \mathbf{u}_2 = \sum_{\mathtt{t}_1,\mathtt{t}_2\in\mathtt{T}}\langle\widetilde{\mathtt{t}_1},\mathbf{u}_1\rangle\langle\widetilde{\mathtt{t}_2},\mathbf{u}_2\rangle\widetilde{(\mathtt{t}_1,\mathtt{t}_2)} \in \mathcal{H}_{\mathtt{T}_1*\mathtt{T}_2}$. Similarly we can

define the tensor product of operators $A_1 \otimes A_2 \in \mathcal{L}(\mathcal{H}_{\mathsf{T}_1 * \mathsf{T}_2})$ by $A_1 \otimes A_2 : \mathbf{u}_1 \otimes \mathbf{u}_2 \mapsto A_1(\mathbf{u}_1) \otimes A_2(\mathbf{u}_2)$ for all $\mathbf{u}_1 \in \mathcal{H}_{\mathsf{T}_1}$ and $\mathbf{u}_2 \in \mathcal{H}_{\mathsf{T}_2}$ and then linearly extend to all states.

Typed Quantum Variables. Recall from Section 5.3 that a quantum variable x of type T, declared as `vars x : T`, consists of a subset $S \subseteq L$ denoted by set(x) (we call it the domain of x), and an encoding function $v_{\mathsf{x}} : \mathsf{T} \to \mathcal{H}_{\mathrm{set}(\mathsf{x})}$ such that $\{|v_{\mathsf{x}}(\mathsf{t})\rangle_{\mathrm{set}(\mathsf{x})}\}_{\mathsf{t} \in \mathsf{T}}$ forms an orthonormal basis of $\mathcal{H}_{\mathrm{set}(\mathsf{x})}$. We can refer to Fig. 5 for a picture. Intuitively, set(x) indicates the subsystem that x refers to, and v_{x} tells us what the computational basis of this system is. We simply write $|\mathsf{t}\rangle_{\mathsf{x}} := |v_{\mathsf{x}}(\mathsf{t})\rangle_{\mathrm{set}(\mathsf{x})}$, which can be also realized as: inject $\widetilde{\mathsf{t}} \in \mathcal{H}_{\mathsf{T}}$ to the subsystem set(x) yields the state $|\mathsf{t}\rangle_{\mathsf{x}} \in \mathcal{H}_{\mathrm{set}(\mathsf{x})}$. More generally, we can inject any typed state or linear operator to x as follows:

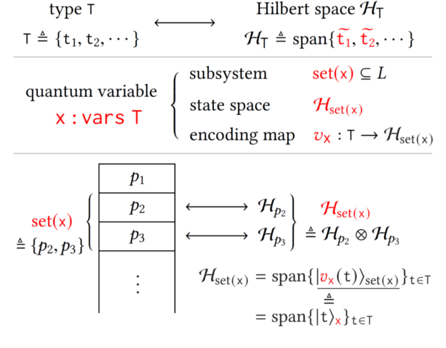

Fig. 5. Typed quantum variable

$$\mathbf{u} \in \mathcal{H}_{\mathsf{T}} \xrightarrow{\texttt{vars x : T}} |u\rangle_{\mathsf{x}} \triangleq \sum_{\mathsf{t} \in \mathsf{T}} \langle \widetilde{\mathsf{t}}, \mathbf{u} \rangle |\mathsf{t}\rangle_{\mathsf{x}} \in \mathcal{H}_{\mathrm{set}(\mathsf{x})}$$

$$A \in \mathcal{L}(\mathcal{H}_{\mathsf{T}}) \xrightarrow{\texttt{vars x : T}} A[\mathsf{x}] \triangleq \sum_{\mathsf{t}_1, \mathsf{t}_2 \in \mathsf{T}} \langle \widetilde{\mathsf{t}_2}, A(\widetilde{\mathsf{t}_1}) \rangle |\mathsf{t}_2\rangle_{\mathsf{x}} \langle \mathsf{t}_1| \in \mathcal{L}(\mathcal{H}_{\mathrm{set}(\mathsf{x})}).$$

Composition of Quantum Variables. CoqQ commonly uses the composition (e.g. Cartesian products) of quantum variables to perform unitary transformations and/or measurements simultaneously on several variables. Below we briefly explain how the composition is implemented. Suppose two quantum variables `vars` $\mathsf{x}_1 : \mathsf{T}_1$ and `vars` $\mathsf{x}_2 : \mathsf{T}_2$ have disjoint domains, i.e., $\mathrm{set}(\mathsf{x}_1) \cap \mathrm{set}(\mathsf{x}_2) = \emptyset$. We can view the pair $[\mathsf{x}_1, \mathsf{x}_2]$ as a quantum variable of type $\mathsf{T}_1 * \mathsf{T}_2$ with $\mathrm{set}([\mathsf{x}_1, \mathsf{x}_2]) = \mathrm{set}(\mathsf{x}_1) \cup \mathrm{set}(\mathsf{x}_2)$ by selecting:

$$v_{[\mathsf{x}_1,\mathsf{x}_2]}(\mathsf{t}_1, \mathsf{t}_2) := |\mathsf{t}_1\rangle_{\mathsf{x}_1} |\mathsf{t}_2\rangle_{\mathsf{x}_2} \in \mathcal{H}_{\mathrm{set}(\mathsf{x}_1) \cup \mathrm{set}(\mathsf{x}_2)}; \quad \text{i.e.,} \quad |(\mathsf{t}_1, \mathsf{t}_2)\rangle_{[\mathsf{x}_1,\mathsf{x}_2]} = |\mathsf{t}_1\rangle_{\mathsf{x}_1} |\mathsf{t}_2\rangle_{\mathsf{x}_2}.$$

Such choice makes the typed tensor product consistent with the composition of quantum variables; for example, suppose $|\phi_i\rangle \in \mathcal{H}_{\mathsf{T}_i}$ and $A_i \in \mathcal{L}(\mathcal{H}_{\mathsf{T}_i})$ for $i = 1, 2$, we have

$$|\phi_1 \otimes \phi_2\rangle_{[\mathsf{x}_1,\mathsf{x}_2]} = |\phi_1\rangle_{\mathsf{x}_1} |\phi_2\rangle_{\mathsf{x}_2}; \quad (A_1 \otimes A_2)[\mathsf{x}_1, \mathsf{x}_2] = A_1[\mathsf{x}_1] \otimes A_2[\mathsf{x}_2].$$

CoqQ provides the composition of other Π types, including tuple and (dependent) finite functions.

Concrete Syntax and Type Checking. The concrete syntax (see Section 7.2) is implemented as a *shallow embedding* in CoqQ with typing rules shown in Fig. 6. Note that the typing rules are directly checked by Coq, i.e., we do not need to define the well-formedness of quantum programs. For example, suppose `x : vars T` and `U` : $\mathcal{L}(\mathcal{H}_{\mathsf{T}})$ and `U` is declared as a canonical instance of `UnitaryfType`, `[ut x := U]` is then a valid command which applies unitary `U` to variable `x`.

Parameterized Judgments. To avoid unnecessary duplication of inference rules, judgments are organized in a single definition, with two boolean parameters pt ∈ {p, t} and st ∈ {s, n} which indicates the total (t) or partial (p) correctness and whether the judgment is saturated (s) or not (n) (saturated means that the inequality in Eqn. (6) or (7) is replaced by "="). Most rules are formalized with parameter pt and/or st, saying that they are sound for both partial/total correctness and/or saturated or not.

Support for For Loops. The proof rules for **for** loops rule make full use of big operator library from MathComp. These rules offer one-step derivation for **for** loops and provide concise proofs

$$\frac{\Gamma \vdash \mathtt{x : vars\ T_1} \quad \Gamma \vdash \mathtt{y : vars\ T_2} \quad \mathrm{set}(\mathtt{x}) \cap \mathrm{set}(\mathtt{y}) = \emptyset}{\Gamma \vdash \mathtt{[x,y] : vars\ (T_1{*}T_2)}}\text{(Var-Pair)}$$

$$\frac{\Gamma \vdash \mathtt{x : vars\ T} \quad \Gamma \vdash \mathtt{t : T}}{\Gamma \vdash \mathtt{x} := |\mathtt{t}\rangle : \mathtt{cmd}}\text{(Init)} \qquad \frac{\Gamma \vdash \mathtt{x : vars\ T} \quad \Gamma \vdash U : \mathcal{U}(\mathcal{H}_{\mathtt{T}})}{\Gamma \vdash \mathtt{x} := U[\mathtt{x}] : \mathtt{cmd}}\text{(UT)}$$

$$\frac{\Gamma \vdash \mathtt{x : vars\ T} \quad \Gamma \vdash C : \mathtt{T} \to \mathtt{cmd}}{\Gamma \vdash \textbf{if}\ \mathrm{meas}[\mathtt{x}] = \mathtt{t} \to C_{\mathtt{t}}\ \textbf{fi} : \mathtt{cmd}}\text{(Cond)} \qquad \frac{\Gamma \vdash \mathtt{x : vars}\ \mathbb{B} \quad \Gamma \vdash \mathtt{b} : \mathbb{B} \quad \Gamma \vdash C : \mathtt{cmd}}{\Gamma \vdash \textbf{while}\ \mathrm{meas}[\mathtt{x}] = b\ \textbf{do}\ C\ \textbf{od} : \mathtt{cmd}}\text{(While)}$$

Fig. 6. Typing rule of the concrete syntax, where T are assumed to be inhabited finite type.

for several circuit-building programs such as HLF algorithm, without the need for mathematical induction which is hard to use in practice if the index is a dependent type.

Utility for Data Type. CoqQ provides rich basic constructs for typed states and unitary operators for both qubits and abstract types, including common 1/2-qubit gates, multiplexer, quantum Fourier bases/transformation, (phase) oracle (i.e., quantum access to a classical function) etc. CoqQ also allows us to build unitary operators from partial information; for example, we use `VUnitary` to build the unitary H_n which maps the default state to uniform superposition state (without knowing about the transformation for other states).

7.3 Statistics

Our Coq/SSReflect development is over 33,000 lines of code, evenly split between definitions and proofs. Our new mathematical libraries form the overwhelming majority of the development with 22000 lines of code in total. The other main parts of the development are the labelled Dirac notation, the qwhile language together with its utility, Hoare logic, and case studies, which respectively account for about 2800, 5000, 1800 and 1300 lines of code. A detailed view of the code statistics is given in Table 2.

Table 2. Code Metric.

	Spec.	**Proof**	**Com.**	**Related files**
Complete partial order	188	90	11	cpo
Tactic for finite set	899	724	166	setdec
Matrix norm / topology	2744	3703	233	mxpred mxnorm mxtopology
Tensor & Hilbert spaces	1017	1345	128	xvector hermitian prodvect tensor
Linear maps / Super-operators	3685	3590	237	lfundef quantum
Subspace theory	1466	1066	80	orthomodular hspace
Labelled Dirac notation	1562	1237	87	dirac
Hilbert spaces over finite type	1181	1466	125	inhabited qtype
QWhile / quantum variables	1245	790	176	qwhile
Quantum Hoare logic	917	919	87	qhl
Case studies	534	758	50	example
Others	329	360	3	mcextra
Total	15767	16048	1383	

8 CASE STUDIES

We use CoqQ to prove the correctness of several well-known quantum algorithms from the literature; the code of the examples is given in Figure 9. Due to space limitations, we develop two examples in more detail, and only provide a short summary for the other examples.

8.1 HSP Algorithm

Fig. 7 outlines the main proof steps of the HSP algorithm. The proof is based on the assumption that the group G is abelian, and uses three main facts from group theory:

- finite Abelian group G is isomorphic to a Cartesian product (or direct sum, if one is familiar with group representation theory) of cyclic groups [Serre 1977]. Without loss of generality our formalization assumes that $G \triangleq \Pi_{0\le i<k}\mathbb{Z}_{p_i}$;
- for every finite abelian group G and subgroup H, G is isomorphic to $H \times G/H$, where G/H denote the quotient group. Elements of G/H are subsets of G called cosets, i.e. $J \in G/H$ iff there exists $g \in G$ such that $J = \{g + h \mid h \in H\}$. Any coset J has the same cardinality (number of elements) as H, i.e., $|J| = |H|$, and thus is always non-empty. We can arbitrarily choose an element from J, denoted by (repr J), and $J = \{g + (\text{repr } J) \mid g \in H\}$. Cosets are disjoint and the union of all cosets forms G, which leads to
$$\sum_{g\in G} F(g) = \sum_{J:\text{coset of } H}\sum_{g\in H} F(g + (\text{repr } J)) \tag{8}$$
for arbitrary function $F : G \to T$ if T is an additive abelian type.
- The character function $\chi_g(h) \triangleq \prod_{m=0}^{k-1} e^{2\pi i g_m h_m/p_m}$ of G where $g, h \in G$, satisfies:
$$\chi_g(h) = \chi_h(g), \quad \chi_g(h_1 + h_2) = \chi_g(h_1)\chi_g(h_2), \quad \sum_{h\in H}\chi_g(h) = \begin{cases} |H| & \forall h \in H. \chi_g(h) = 1 \\ 0 & \text{otherwise} \end{cases}. \tag{9}$$

The proof interleaves applications of proof rules for program constructs, and calculations on labelled Dirac notation. The latter are denoted by $\Leftrightarrow$ in the figure, and are annotated with group theory when the calculation relies on the aforementioned facts from group theory. The final step of the proof applies the rule (R.Inner) to derive Eqn. (1) and (2) from the current post-condition. The use of the rule (R.Inner) is very convenient here. Indeed, backward reasoning of HSP is relatively difficult since such derivation ignores the algebraic structure of the states during the execution that started from $|0\rangle_{\overline{x}}|y_0\rangle_y$. All proofs in the literature are forward, i.e., they show that the output is in a certain state.

Proof outline and interpretation of judgments. We label the inference rules in blue and mark the main lemmas we used from group theory in red in Fig. 7. We first derive the correct formula via forward reasoning:
$$\models^s_t \{\!|1|\!\} HSP \left\{\!\!\left| \frac{1}{|H^\perp|}\sum_{g\in H^\perp}|g\rangle_{\overline{x}}\left[\sum_{J:\text{coset of } H}\chi_g(\text{repr } J)|t_0 + f(\text{repr } J)\rangle_y\right]\right|\!\!\right\}.$$
and then finish the proof of Eqn. (1) and (2), or more specifically,
$$\forall\, g \in H^\perp,\ \models^s_t \left\{\frac{1}{|H^\perp|}\right\}\ HSP\ \{|g\rangle_{\overline{x}}\langle g|\}, \qquad \forall\, g \notin H^\perp,\ \models^s_t \{0\}\ HSP\ \{|g\rangle_{\overline{x}}\langle g|\}.$$
by employing rule (R.Inner). The superscript "s" means that both correctness formulas are saturated, i.e., the pre- and post-expectation are the same; the preconditions are scalars and thus interpreted as probability; the expectation of postcondition $|g\rangle_{\overline{x}}\langle g|$ is the probability that we obtain g if we measure $\overline{x}$ on the computational basis. In summary, these two Hoare triples exactly tell us that:

Extra definitions/lemmas

$U[f] \triangleq \sum_{g\in G}\sum_{t\in X} |(g, t+f(g))\rangle\langle(g,t)|$

$F_G \triangleq \frac{1}{\sqrt{|G|}}\sum_{g,h\in G}\chi_g(h)|g\rangle\langle h|$

$\bigotimes_{i=0}^{k-1} \mathtt{QFT}[\mathtt{x}_i] = F_G[\overline{\mathtt{x}}]$

$\{\!|1|\!\}$

- **for** $0\le i<k$ **do** $\mathtt{x}_i$:= $|0\rangle$; (Ax.InFP′)

$\{\!\!\{\bigotimes_{i=0}^{k-1}|0\rangle_{\mathtt{x}_i}\}\!\!\} \Longleftrightarrow \{\!|\,|0\rangle_{\overline{\mathtt{x}}}|\!\}$

- y := $|\mathtt{t}_0\rangle$; (Ax.InF′)

$\{\!|\,|0\rangle_{\overline{\mathtt{x}}}\otimes|\mathtt{y}_0\rangle_{\mathtt{y}}|\!\}$

- **for** $0\le i<k$ **do** $\mathtt{x}_i$:= QFT[$\mathtt{x}_i$]; (Ax.UTFP′)

$\{\!\!\{\left(\bigotimes_{i=0}^{k-1}\mathtt{QFT}[\mathtt{x}_i]\right)|0\rangle_{\overline{\mathtt{x}}}\otimes|\mathtt{y}_0\rangle_{\mathtt{y}}\}\!\!\} \Longleftrightarrow \{\!|F_G[\mathtt{x}]|0\rangle_{\overline{\mathtt{x}}}\otimes|\mathtt{y}_0\rangle_{\mathtt{y}}|\!\} \Longleftrightarrow$

$\{\!\!\{\frac{1}{\sqrt{|G|}}\sum_g|g\rangle_{\overline{\mathtt{x}}}\otimes|\mathtt{y}_0\rangle_{\mathtt{y}}\}\!\!\} \Longleftrightarrow \{\!\!\{\frac{1}{\sqrt{|G|}}\sum_g|(g,\mathtt{y}_0)\rangle_{[\overline{\mathtt{x}},\mathtt{y}]}\}\!\!\}$

- [$\overline{\mathtt{x}}$, y] := $U[f]$[$\overline{\mathtt{x}}$, y]; (Ax.UTF′)

$\{\!\!\{U[f][\overline{\mathtt{x}},\mathtt{y}]\frac{1}{\sqrt{|G|}}\sum_g|(g,\mathtt{y}_0)\rangle_{[\overline{\mathtt{x}},\mathtt{y}]}\}\!\!\} \Longleftrightarrow \{\!\!\{\frac{1}{\sqrt{|G|}}\sum_g|(g,\mathtt{y}_0+f(g))\rangle_{[\overline{\mathtt{x}},\mathtt{y}]}\}\!\!\} \overset{\text{group}}{\underset{\text{theory}}{\Longleftrightarrow}}$

$$\left\{\!\!\left\{\frac{1}{\sqrt{|G|}}\sum_{J:\text{coset of } H}\sum_{g\in H}|g+(\text{repr } J)\rangle_{\overline{\mathtt{x}}}|\mathtt{y}_0+f(\text{repr } J)\rangle_{\mathtt{y}}\right\}\!\!\right\}$$

- **for** $0\le i<k$ **do** $\mathtt{x}_i$:= QFT[$\mathtt{x}_i$]. (Ax.UTFP′)

$$\left\{\!\!\left\{\left(\bigotimes_{i=0}^{k-1}\mathtt{QFT}[\mathtt{x}_i]\right)\frac{1}{\sqrt{|G|}}\sum_{J:\text{coset of } H}\sum_{g\in H}|g+(\text{repr } J)\rangle_{\overline{\mathtt{x}}}|\mathtt{y}_0+f(\text{repr } J)\rangle_{\mathtt{y}}\right\}\!\!\right\} \overset{\text{group}}{\underset{\text{theory}}{\Longleftrightarrow}}$$

$$\left\{\!\!\left\{\frac{1}{|H^\perp|}\sum_{g\in H^\perp}|g\rangle_{\overline{\mathtt{x}}}\left[\sum_{J:\text{coset of } H}\chi_g(\text{repr } J)|\mathtt{y}_0+f(\text{repr } J)\rangle_{\mathtt{y}}\right]\right\}\!\!\right\}$$

(R.Inner) $\forall g\in H^\perp,\ \models^{\mathrm{s}}_{\mathrm{t}} \left\{\frac{1}{|H^\perp|}\right\}\ HSP\ \{|g\rangle_{\overline{\mathtt{x}}}\langle g|\}$

(R.Inner) $\forall g\notin H^\perp,\ \models^{\mathrm{s}}_{\mathrm{t}} \{0\}\ HSP\ \{|g\rangle_{\overline{\mathtt{x}}}\langle g|\}$

Fig. 7. Proof outline for HSP algorithm. The predicate inside the white curly brackets stands for $\{\!|A|\!\}\triangleq\{AA^\dagger\}$. All left-right arrows ⇔ are the rewrites of the predicates. CoqQ provides the built-in function `Oracle f` to construct $U[f]$ directly.

after executing the quantum part of the HSP algorithm and measuring the register $\overline{\mathtt{x}}$, the probability of obtaining outcome g is $\frac{1}{|H^\perp|}$ if $g\in H^\perp$ and 0 if $g\notin H^\perp$, as we desired.

Statistics. The full formalization is about 390 lines of code. It consists of 50+ lines for proving Eqn. (8) for general finite abelian groups, 60 lines for proving Eqn. (9), 10+ lines to set up variables, hypotheses and HSP algorithm and 250 lines for proving the correctness formulas Eqn. (1) and (2).

8.2 HHL Algorithm

The Harrow-Hassidim-Lloyd or HHL algorithm [Harrow et al. 2009] is a well-known quantum algorithm for solving linear systems of equations, i.e., finding a vector $\mathbf{x}$ such that $A\mathbf{x}=\mathbf{b}$ for

given matrix A and vector $\mathbf{b}$. [Zhou et al. 2019] gives a pen-and-paper proof of the algorithm using quantum Hoare logic. We use CoqQ to formalize the proof.

For simplicity, we follow the same assumptions of [Zhou et al. 2019]: assume A is a full-rank Hermitian operator in $\mathcal{H}_{\mathbb{Z}_m}$ (i.e., A is of m-dimensional matrix) with diagonal decomposition $A = \sum_{j<m} \lambda_j |u_j\rangle\langle u_j|$ where $\{|u_j\rangle\}$ is an orthonormal basis of $\mathcal{H}_{\mathbb{Z}_m}$; assume $|b\rangle \in \mathcal{H}_{\mathbb{Z}_m}$ is an normalized state (i.e., $\| |b\rangle \| = 1$); and further assume that there exists $t_0 \in \mathbb{R}$, for all $0 \le j \le m$, $\lambda_j t_0$ is multiple of 2π, i.e., $\delta_j = \frac{\lambda_j t_0}{2\pi} \in \{1, 2, \cdots, n-1\}$, which makes the algorithm exact.

The algorithm operates on three quantum variables: 1. `q : vars` $\mathbb{Z}_m$ which is used to store the input data $|b\rangle$ – achieved by applying a unitary operator U_b which transforms $|0\rangle$ into $|b\rangle$; 2. `p : vars` $\mathbb{Z}_n$ which acts as the control system and is used to store the value of δ_j; and 3. `r : vars` $\mathbb{B}$ which is used as the guard of while loop – indicating if the subroutine succeeds – done by employing a controlled unitary $U_c : \mathcal{L}(\mathcal{H}_{\mathbb{Z}_n * \mathbb{B}})$ with some suitable parameter C such that:

$$U_c|(0,0)\rangle = |(0,0)\rangle; \quad U_c|(i,0)\rangle = |i\rangle\left(\sqrt{1 - \frac{C^2}{i^2}}|0\rangle + \frac{C}{i}|1\rangle\right) \; \forall i = 1, 2, \cdots, n-1.$$

Writing the algorithm uses several built-in functions of CoqQ. For example, U_b and U_c are constructed by the built-in functions `PUnitary` and `VUnitary` which provide a unitary from the partial information (e.g., we only provide $|0\rangle \mapsto |b\rangle$ for U_b). The key transformation of HHL is the multiplexer of the function $f : k \mapsto e^{\mathrm{i}Akt_0/n}$; that is, $U_f \triangleq \texttt{Multiplexer}(f) = \sum_{k<n} |k\rangle\langle k| \otimes e^{\mathrm{i}Akt_0/n}$.

The HHL program together with a proof outline is displayed in Fig. 8. It is easy to see that the solution for the linear equation $A|x\rangle = |b\rangle$ is $|x\rangle = c\sum_{j=1}^{N} \frac{\beta_j}{\lambda_j}|u_j\rangle$ with $\beta_j \triangleq \langle j|b\rangle$ up to some unimportant scale factor where c is only used to normalize $|x\rangle$. The correctness of the HHL algorithm can be formulated as follows:

$$\models_{\mathrm{p}} \{\!|1|\!\} HHL \{\!| |x\rangle_{\mathtt{q}} |\!\}. \tag{10}$$

Informally, whenever the program terminates, the variable `q` is in state $|x\rangle$.

The proof uses subspace theory based on projection representation, simplifying the proof of Löwner order property. Roughly speaking, a state $|v\rangle$ with norm $\| |v\rangle \| \le 1$ lies in some subspace V (convertible to the projection) must implies $|v\rangle\langle v| \sqsubseteq V$.

Statistics. The code in total is about 280 lines, including 90 lines for setting up the parameters and simple properties of these parameters, 15 lines to define the HHL program and 170 lines for proving the Eqn. (10).

8.3 Parallel Hadamard

Parallel Hadamard is a circuit that converts the initial state of a quantum circuit to a uniform superposition state. Parallel Hadamard is a key step to leverage the power of quantum computation and is frequently used at the beginning or the end of a circuit implementation of algorithms. The program is quite simple, as shown in Fig. 9. We prove:

$$\models_{\mathrm{pt}}^{\mathrm{st}} \{\!| \textstyle\bigotimes_i |0\rangle_{\mathtt{x}_i} |\!\} \texttt{ParaHadamard} \{\!| \textstyle\bigotimes_i |+\rangle_{\mathtt{x}_i} |\!\} \quad \models_{\mathrm{pt}}^{\mathrm{st}} \{\!| \textstyle\bigotimes_i |+\rangle_{\mathtt{x}_i} |\!\} \texttt{ParaHadamard} \{\!| \textstyle\bigotimes_i |0\rangle_{\mathtt{x}_i} |\!\} \tag{11}$$

$$\models_{\mathrm{pt}}^{\mathrm{st}} \{\!| |b\rangle_{\overline{\mathtt{x}}} |\!\} \texttt{ParaHadamard} \left\{\!\left| \frac{1}{\sqrt{2^n}} \textstyle\sum_t (-1)^{\sum_i b_i t_i} |t\rangle_{\overline{\mathtt{x}}} \right|\!\right\} \tag{12}$$

The `ParaHadamard` transforms $\bigotimes_i |0\rangle_{\mathtt{x}_i}$ to $\bigotimes_i |+\rangle_{\mathtt{x}_i}$ and vice versa as we expected, or more generally, transforms $|b\rangle_{\overline{\mathtt{x}}}$ to $\frac{1}{\sqrt{2^n}} \sum_t (-1)^{\sum_i b_i t_i} |t\rangle_{\overline{\mathtt{x}}}$.

Statistics. We use (Ax.UTPF') to reason about Eqn. (11) in one step with only 3 lines of proof code each and Eqn. (11) with about 20 lines since we should derive $\sum_i b_i t_i$; it takes 60 lines to prove several goals for tuple and finite functions.

$\{\!|1|\!\}$

- `p := |0⟩; q := |0⟩; r := |0⟩;` (Ax.InF′)

$\{\!| |0\rangle_{\mathtt{p}}|0\rangle_{\mathtt{q}}|0\rangle_{\mathtt{r}} |\!\} \xRightarrow{\text{(R.Or)}} \{R\}$

- **while** `meas[r] = 0` **do** (R.LP.P)

  $\{P\} \xRightarrow{\text{(R.TI)}} \{\!| |0\rangle_{\mathtt{p}}|0\rangle_{\mathtt{r}} |\!\}$

  - `q := |0⟩;` (Ax.InF′)

  $\{\!| |0\rangle_{\mathtt{p}}|0\rangle_{\mathtt{q}}|0\rangle_{\mathtt{r}} |\!\}$

  - `q :=` U_b`[q];` (Ax.UTF′)

  $\{\!| |0\rangle_{\mathtt{p}}(U_b[\mathtt{q}]|0\rangle_{\mathtt{q}})|0\rangle_{\mathtt{r}} |\!\} \Leftrightarrow \{\!| |0\rangle_{\mathtt{p}}|b\rangle_{\mathtt{q}}|0\rangle_{\mathtt{r}} |\!\}$

  - `p :=` H_u`[p];` (Ax.UTF′)

  $\{\!| (H_u[\mathtt{p}]|0\rangle_{\mathtt{p}})|b\rangle_{\mathtt{q}}|0\rangle_{\mathtt{r}} |\!\} \Leftrightarrow \left\{\!\left| \frac{1}{\sqrt{n+1}} \sum_{\tau:[n+1]} |\tau\rangle_{\mathtt{p}}|b\rangle_{\mathtt{q}}|0\rangle_{\mathtt{r}} \right|\!\right\}$

  - `[p,q] :=` U_f`[p,q];` (Ax.UTF′)

  $\left\{\!\left| \frac{1}{\sqrt{n+1}} \sum_{\tau}(U_f[\mathtt{p},\mathtt{q}]|\tau\rangle_{\mathtt{p}}|b\rangle_{\mathtt{q}})|0\rangle_{\mathtt{r}} \right|\!\right\} \Leftrightarrow \left\{\!\left| \frac{1}{\sqrt{n+1}} \sum_j \left(\sum_\tau \beta_j e^{\mathrm{i}\tau\lambda_j t_0/(n+1)}|\tau\rangle_{\mathtt{p}}\right)|u_j\rangle_{\mathtt{q}}|0\rangle_{\mathtt{r}} \right|\!\right\}$

  - `p := IQFT[p];` (Ax.UTF′)

  $\left\{\!\left| \frac{1}{\sqrt{n+1}} \sum_j \left(\mathtt{IQFT}[\mathtt{p}] \sum_\tau \beta_j e^{\mathrm{i}\tau\lambda_j t_0/(n+1)}|\tau\rangle_{\mathtt{p}}\right)|u_j\rangle_{\mathtt{q}}|0\rangle_{\mathtt{r}} \right|\!\right\} \Leftrightarrow \{\!| \sum_j \beta_j|\delta_j\rangle_{\mathtt{p}}|u_j\rangle_{\mathtt{q}}|0\rangle_{\mathtt{r}} |\!\}$

  - `[p,r] :=` U_c`[p,r];` (Ax.UTF′)

  $\{\!| \sum_j \beta_j|u_j\rangle_{\mathtt{q}}(U_c[\mathtt{p},\mathtt{r}]|\delta_j\rangle_{\mathtt{p}}|0\rangle_{\mathtt{r}}) |\!\} \Leftrightarrow \{\!| \sum_j \beta_j|\delta_j\rangle_{\mathtt{p}}|u_j\rangle_{\mathtt{q}}|v_j\rangle_{\mathtt{r}} |\!\}$

  - `p := QFT[p];` (Ax.UTF′)

  $\{\!| \sum_j \beta_j(\mathtt{QFT}[\mathtt{p}]|\delta_j\rangle_{\mathtt{p}})|u_j\rangle_{\mathtt{q}}|v_j\rangle_{\mathtt{r}} |\!\} \Leftrightarrow \left\{\!\left| \frac{1}{\sqrt{n+1}} \sum_{j,\tau} \beta_j e^{\mathrm{i}2\pi\delta_j\tau/T}|\tau\rangle_{\mathtt{p}}|u_j\rangle_{\mathtt{q}}|v_j\rangle_{\mathtt{r}} \right|\!\right\}$

  - `[p,q] :=` $U_f^\dagger$`[p,q];` (Ax.UTF′)

  $\left\{\!\left| \frac{1}{\sqrt{n+1}} \sum_{j,\tau} \beta_j e^{\mathrm{i}2\pi\delta_j\tau/T}(U_f^{-1}[\mathtt{p},\mathtt{q}]|\tau\rangle_{\mathtt{p}}|u_j\rangle_{\mathtt{q}})|v_j\rangle_{\mathtt{r}} \right|\!\right\} \Leftrightarrow \left\{\!\left| \frac{1}{\sqrt{n+1}} \sum_\tau |\tau\rangle_{\mathtt{p}} \sum_j \beta_j|u_j\rangle_{\mathtt{q}}|v_j\rangle_{\mathtt{r}} \right|\!\right\}$

  - `p :=` $H_n^\dagger$`[p];` (Ax.UTF′)

  $\left\{\!\left| \left(H_n^\dagger[\mathtt{p}] \frac{1}{\sqrt{n+1}} \sum_\tau |\tau\rangle_{\mathtt{p}}\right) \sum_j \beta_j|u_j\rangle_{\mathtt{q}}|v_j\rangle_{\mathtt{r}} \right|\!\right\} \Leftrightarrow \{\!| \underline{|0\rangle_{\mathtt{p}} \sum_j \beta_j|u_j\rangle_{\mathtt{q}}|v_j\rangle_{\mathtt{r}}} |\!\}$

  $\xRightarrow[\text{subspace theory}]{\text{(R.Or)}} \{R\}$ $\quad = \underbrace{C|0\rangle_{\mathtt{p}}|x\rangle_{\mathtt{q}}|1\rangle_{\mathtt{r}}}_{\in Q} + \underbrace{|0\rangle_{\mathtt{p}}\left(\sum_j \beta_j \sqrt{1-\frac{C^2}{\delta_j^2}}|u_j\rangle_{\mathtt{q}}\right)|0\rangle_{\mathtt{r}}}_{\in P}$

- **do** (R.LP.P)

$\{Q\} \xRightarrow{\text{(R.Or)}} \{\!| |x\rangle_{\mathtt{q}} |\!\}$

Extra definitions/lemmas

$P \triangleq |0\rangle_{\mathtt{p}}\langle 0| \otimes I_q \otimes |0\rangle_{\mathtt{r}}\langle 0|$

$Q \triangleq |0\rangle_{\mathtt{p}}\langle 0| \otimes |x\rangle_{\mathtt{q}}\langle x| \otimes |1\rangle_{\mathtt{r}}\langle 1|$

$R \triangleq P + Q$

$R = |0\rangle_{\mathtt{r}}\langle 0|P|0\rangle_{\mathtt{r}}\langle 0| + |1\rangle_{\mathtt{r}}\langle 1|Q|1\rangle_{\mathtt{r}}\langle 1|$

$|v_j\rangle_{\mathtt{r}} \triangleq \sqrt{1-\frac{C^2}{\delta_j^2}}|0\rangle_{\mathtt{r}} + \frac{C}{\delta_j}|1\rangle_{\mathtt{r}}, \quad \forall 1 \le j \le n$

Fig. 8. Proof outline for HHL algorithm. The left-right arrow represents the rewrite of predicates. `QFT` and `IQFT` are built-in (inverse) quantum Fourier transformation for $\mathbb{Z}_p$ type. H_n is the built-in unitary that maps default state ($|0\rangle$ here) to the uniform superposition state.

```
(* x : vars 𝔹 *)
(* s : sequence of (vars 𝔹)) *)
Definition QFT_sub x s :=
  for i < size(s) do
    [x,s_i] := CU(Ph(e^{πi/2^{i+1}}))[x,s_i].

Fixpoint QFT_iter s :=
  match s with
  | [::] ⇒ skip
  | x :: t ⇒ x := H[x];
             (QFT_sub x t);
             (QFT_iter t)
  end.

(* s : n-tuple of (vars 𝔹) *)
Definition QFT_cir s :=
  QFT_iter s;
  rev_circuit s.

Definition QPE :=
  x := |0⟩;
  x := H_n[x];
  [x,y] := Multiplexer(fun i ⇒ U^i)[x,y];
  x := IQFT[x].

(* x : n-tuple of (vars 𝔹)) *)
Definition ParaHadamard :=
  for i < n do x_i := H[x_i].

(* x : n-tuple of (vars T)) *)
Definition rev_circuit :=
  for i < ⌊n/2⌋ do
    [x_i,x_{n-i}] := SWAP[x_i,x_{n-i}].

Definition HLF :=
  for i do x_i := |0⟩;
  for i do x_i := H[x_i];
  for i ∈ SD do x_i := S[x_i];
  for i ∈ SS do
    [x_{i_1},x_{i_2}]:= CZ[x_{i_1},x_{i_2}];
  for i do x_i := H[x_i].

Definition Grover (r : ℕ) :=
  x := |t_0⟩;
  x := H_n[x];
  for i < r do (
    x := PhOracle(f)[x];
    x := H_n^†[x];
    x := PhOracle(fun i ⇒ i == t_0)[x];
    x := H_n[x];).
```

Fig. 9. Programs for Parallel Hadamard, reverse circuit, QFT circuit, Hidden linear function, Quantum phase estimation and Grover search algorithm. CU, Ph, H, CZ are the built-in controlled-unitary gate, parameterized phase gate, Hadamard gate and controlled-Pauli Z gate respectively. Multiplexer, SWAP, PhOracle, H_n are built-in multiplexer, SWAP gate, phase oracle and uniform transformation (from default state ($|0\rangle$ in QPE and $|\mathtt{t}_0\rangle$ in Grover) to uniform superposition state). The programs use metaprogramming to introduce classical variables; the subroutine QFT_iter of QFT_cir further employs Coq's control flow and is written using a fixpoint definition.

8.4 QFT Circuit and Reverse Circuit

As one of the most fundamental building blocks in designing quantum algorithms, verifying the correctness of the QFT circuit is a common task for a program verifier. The QFT circuit is parameterized by a meta variable n (its size) and thus is defined using the fixpoint function rather than the concrete syntax. It also employs a reverse circuit at the end, which is used to reverse the order of qubits. We show:

$$\models^{\mathrm{st}}_{\mathrm{pt}} \{\!| |t\rangle_{\overline{\mathtt{x}}} |\!\} \mathtt{rev_circuit} \left\{\!\!\left| |t\rangle_{\overline{\mathtt{rev(x)}}} \right|\!\!\right\} \qquad \models^{\mathrm{st}}_{\mathrm{pt}} \left\{\!\!\left| |t\rangle_{\overline{\mathtt{rev(x)}}} \right|\!\!\right\} \mathtt{rev_circuit} \{\!| |t\rangle_{\overline{\mathtt{x}}} |\!\} \tag{13}$$

$$\models^{\mathrm{st}}_{\mathrm{pt}} \{\!| |b\rangle_{\overline{\mathtt{s}}} |\!\} \mathtt{QFT_cir} \{\!| |\mathtt{QFTbv}\ b\rangle_{\overline{\mathtt{s}}} |\!\} \quad \text{where } |\mathtt{QFTbv}\ b\rangle = \frac{1}{\sqrt{2^n}} \sum_{t:\{0,1\}^n} e^{2\pi \mathrm{i} f(b) f(t)/2^n} |t\rangle \tag{14}$$

where the f converts a bit string ($\mathbb{B}$ tuple) to a natural number $\mathbb{N}$. The reverse circuit rev_circuit simply reverses the order of a tuple of variables $\overline{\mathtt{x}}$ (with arbitrary types rather than just qubit). Eqn. 14 can be easily interpreted as converting $|b\rangle_{\overline{\mathtt{s}}}$ to its corresponding QFT basis $|\mathtt{QFTbv}\ b\rangle$.

Statistics for the reverse circuit. It is about 80 lines of code to formalize the reverse circuit and finish the proof of Eqn. (13); it is slightly longer since we need to show the side condition of using (Ax.UTFP) – disjointness of each SWAP.

Statistics for QFT circuit. The formalization and proof use mathematical induction and take about 140+ lines of code in total (excluding the code of the reverse circuit).

8.5 BGK Algorithm

The Bravyi-Gosset-Konig, or BGK, algorithm [Bravyi et al. 2018] is an algorithm to solve the hidden linear function (HLF). Suppose A is a $n \times n$ symmetric boolean matrix; the goal of the HLF problem is to find a boolean vector $z \in \{0,1\}^n$ such that

$$\forall x, (Ax = 0 \mod 2) \to \left(q(x) = 2\textstyle\sum_i z_i x_i \mod 4\right)$$

where $q(x) \triangleq \sum_{i,j} A_{ij} x_i x_j \mod 4$. We define two set $\mathtt{SD} = \{i \mid A_{ii} = 1\}$ and $\mathtt{SS} = \{(i,j) \mid i < j \text{ and } A_{ij} = 1\}$ to specify the code. As summarized in Eqn. (4) in [Bravyi et al. 2018], we verify:

$$\models^{\mathrm{st}}_{\mathrm{pt}} \{\!|1|\!\} \mathtt{HLF} \left\{\!\!\left| \tfrac{1}{2^n} \textstyle\sum_{z:\{0,1\}^n} \left(\sum_{k:\{0,1\}^n} \mathbf{i}^{q(k)+2\sum_i k_i z_i} \right) |z\rangle_{\overline{\mathtt{x}}} \right|\!\!\right\}. \tag{15}$$

The algorithm is implemented as a circuit-building program that works on 2-dimensional grids of qubits.

Statistics. All code is about 160 lines: 30+ lines to set up the variables, parameters and algorithm; and 120+ lines to finish the proof of Eqn. (15).

8.6 QPE

Quantum phase estimation (QPE) is a quantum algorithm that is often used as a subroutine in other quantum algorithms to estimate the phase (eigenvalue) of an eigenvector of a unitary operator. Given a unitary operator U and an eigenvector $|\phi\rangle$ of U; the goal is to find an approximation $0 \le \theta < 1$ such that $U|\phi\rangle = e^{2\pi \mathrm{i}\theta}$. The correctness of QPE is stated as follows:

$$\forall a < n,\ \models^{\mathrm{st}}_{\mathrm{pt}} \left\{\!\left| c(a) |\phi\rangle_{\mathtt{y}} \right|\!\right\} \mathtt{QPE} \left\{\!\left| |a\rangle_{\mathtt{x}} |\phi\rangle_{\mathtt{y}} \right|\!\right\} \quad \text{with } c(a) \triangleq \textstyle\sum_{j<n} e^{2\pi \mathrm{i}(a/n-\theta)j/n}. \tag{16}$$

where $\mathtt{y} : \mathtt{vars\ T}$ stores $|\phi\rangle$ and $\mathtt{x} : \mathtt{vars}\ \mathbb{Z}_n$ is the control system that is used to approximate $n\theta$.

This judgment tells us if we measure register $\mathtt{x}$ at the end, we have probability $|c(a)|^2$ to obtain outcome a. A straightforward calculation shows that, if $n\theta \in \mathbb{N}$, then we have probability 1 to get $n\theta$; otherwise, we have at least probability $4/\pi^2$ to obtain $\mathrm{round}(n\theta)$ (i.e., the closest integer to $n\theta$).

Statistics. Setting up algorithms and proving Eqn. (16) only takes 40+ lines of code; we take another 50 lines to show the property of function $c(a)$, i.e., $|c(\mathrm{round}(n\theta))|$ is 1 if the algorithm is exact, and at least $2/\pi$ otherwise.

8.7 Grover's Algorithm

Grover's algorithm is a quantum algorithm for unstructured data search and offers a quadratic speedup compared to the classical algorithm. We assume that the type of data is $\mathtt{T}$ and the given function $f : \mathtt{T} \to \mathbb{B}$ which can be accessed by a phase oracle (i.e., modelled as $\mathtt{PhOracle}[f]$). Let $|v\rangle \triangleq \frac{1}{\sqrt{|f|}} \sum_{i:f(i)=1} |i\rangle$ the superposition of all solutions (i.e., $f(i) = 1$). We show that if we run the subroutine for r times, the following Hoare triple holds:

$$\models_{\mathrm{pt}} \left\{\sin^2((2r+1)t)\right\} \mathtt{Grover(r)} \left\{\textstyle\sum_{i:f(i)=1} |i\rangle_{\mathtt{x}}\langle i|\right\} \quad \text{with } t \triangleq \arcsin\sqrt{\tfrac{|f|}{|T|}}. \tag{17}$$

It implies that if we measure $\mathtt{x}$ at the end, we have at least the probability $\sin^2((2r+1)t)$ to obtain a solution of f. With a proper choice of r, i.e., let $\sin^2((2r+1)t)$ close to 1, Grover's algorithm succeeds with high probability.

Statistics. The code is about 180 lines, including 30+ lines to set up the parameters and algorithms and 140+ lines for proving the Hoare triple.

9 RELATED WORK

There is a large body of work in the design, implementation, and verification of quantum programs. For convenience, we distinguish approaches that are supported by artefacts (formalizations in proof assistants, verification tools) and theoretical approaches. For space reasons, we exclude approaches based on testing and program analyses.

Mechanized approaches. Table 1 summarizes the main differences between CoqQ and some other verification tools for quantum programs, based on the four criteria discussed in the introduction. We comment on the tools below.

*Q*WIRE [Paykin et al. 2017; Rand et al. 2017] is a Coq formalization of quantum programs written in a circuit-like language. The formalization includes a denotational semantics of programs in terms of density matrices, and has been used to verify several interesting algorithms. A recent extension of *Q*WIRE, called *ReQ*WIRE [Rand et al. 2018b] develops a verified compiler to compile classical circuits to reversible quantum circuits. *Q*WIRE does not include program logic. *Q*WIRE is built on top of the standard library of Coq for the theory of real numbers, and builds its own libraries for complex numbers and matrix theory. Interestingly, the authors of *Q*WIRE report that MathComp was also considered as an external library for early development of *Q*WIRE, but due to the overhead caused by dependent types, the authors use phantom type [Rand et al. 2018a] instead. In contrast, the use of MathComp is more important in our setting, as we aim to support general notions of states.

SQIR [Hietala et al. 2021a,b] is a Coq formalization for formal verification of quantum programs. SQIR provides a semantics of programs based on a density matrix representation of quantum states, and proves the correctness of a quantum circuit optimizer. The formalization does not include program logic. Like *Q*WIRE, SQIR is built on top of the standard library of Coq. For the latter work that extracts OpenQASM programs from the Coq representation of SQIR programs, the Coq extraction mechanism is in the Trusted Computing Base.

The QHLProver [Liu et al. 2019] is an Isabelle formalization for formal verification of quantum programs based on quantum Hoare logic [Ying 2011]. Their formalization includes a denotational semantics of **qwhile** programs, and a Hoare logic that is proved sound with respect to the program semantics. The prover is used to verify several examples, including Grover's algorithm. However, the main difference with our work is that their formalization is based on a matrix representation of states, rather than an abstract representation. This makes the formalization of examples such as HSP cumbersome. The formalization uses the library JNF – Jordan_Norm_Forms [Thiemann and Yamada 2016] for matrices, and the library DL – Deep_Learning [Bentkamp et al. 2019] for tensors.

qrhl-tool [Unruh 2019b] is an Isabelle formalization for formal verification of quantum programs. Programs in qrhl-tool are written in a high-level language that supports rich types for quantum variables. Programs can be verified using quantum relational Hoare logic (QRHL) [Unruh 2019b]. The main application of the qrhl-tool is security proofs of (post-quantum) cryptographic constructions. The current version of qrhl-tool is not foundational: program semantics are not built from first principles, and the program logic is not proved sound with respect to the denotational semantics. Recently, Caballero and Unruh developed CBO – Complex Bounded Operator [Caballero and Unruh 2021], an Isabelle library that is used for modelling assertions. qrhl-tool uses CBO and many other libraries. [Unruh 2021] proposed a general theory of registers by defining the register category which intuitively transforms updates on the register's domain into updates on the program state.

In contrast, we directly use labelled Dirac notation that uses auto-identified cylindrical extensions to express the update of states.

Qbricks [Chareton et al. 2021] is a verification framework for circuit-building quantum programs. The framework is a classic automated verification framework that supports the automated verification of rich specifications. Concretely, Qbricks targets SMT-solvers via the Why3 verification platform. The encoding of programs into SMT-clauses is based on the path-sum representation of quantum states introduced in [Amy 2018]. The verification approach is proved sound in Why3 with only two assumed theorems about trigonometric function sine, and the framework is used to verify many of the algorithms that are verified using CoqQ.

Isabelle Marries Dirac [Bordg et al. 2021] is an Isabelle formalization for formal verification of quantum programs. The formalization uses a shallow embedding of quantum circuits based on the JNF library for matrices [Thiemann and Yamada 2016]. The formalization also provides a rudimentary encoding of Dirac notation; for instance, no big operators are supported. The formalization is used to verify several algorithms, and helped uncover a bug in a published proof of the Quantum Prisoner Dilemma.

Theoretical approaches. We compare our approach with Ying's work on quantum Hoare logic and recent work on quantum separation logic. Then we briefly discuss other approaches.

Comparison with [Ying 2011]. [Ying 2011] proposes the first sound and relatively complete proof system for **qwhile**-programs. Ying's seminal work was subsequently extended in many directions. [Feng and Ying 2021] extend the program logic to programs with quantum and classical variables. [Ying et al. 2018, 2022] extend the logic to parallel programs. [Zhou et al. 2019] proposes a variant of quantum Hoare logic that uses projections as pre- and postconditions.

The proof rules in these systems are similar to ours in many ways. Our rules for basic constructs are inspired by [Ying 2011], the rules for **for** loops are similar to [Ying et al. 2018, 2022], and many structural rules are inspired by [Ying 2019]. One minor difference is that our rules use linear operators rather than quantum predicates. This minimizes the number of side conditions in proof rules. Note that the rule (R.Inner) is new.

Comparison with Quantum Separation Logic. Separating conjunction is a logical construct that is used by separation (and other resource-aware) logics to model disjointness between two systems. Separating conjunction was originally used to model spatial disjunction on heaps. There have been two recent proposals [Le et al. 2022; Zhou et al. 2021] to use separating conjunction for capturing separability (vs. entanglement) of quantum states. However, these proof systems are not powerful enough for programs with highly entangled subroutines. It remains to be seen whether this is a limitation of these two proof systems, or a fundamental limitation of separating conjunction. From the perspective of users familiar with quantum physics, one potential drawback of using separation conjunction (rather than labels, which enforce separation syntactically) is that the connection with labelled Dirac notation is lost.

Generalizations and other approaches. Recent works [Barbosa et al. 2021; Barthe et al. 2019; Li and Unruh 2021; Unruh 2019b] develop relational Hoare logics for quantum programs. A promising direction for future work is to enhance our formalization to support relational reasoning.

There are also many alternative approaches to verify quantum programs based on dynamic logic, temporal logic, Kleene algebra, type theory and process algebra, see for instance [Akatov 2005; Baltag and Smets 2006; Brunet and Jorrand 2004; Feng et al. 2007; Kakutani 2009; Peng et al. 2022; Singhal 2020; Yu 2019]. A long-term goal would be to build verified program verifiers for some of these approaches.

Another approach is the ZX-calculus [Coecke and Duncan 2011] that was proposed for reasoning about linear maps between qubits. The ZX-calculus originates from earlier work on categorical foundations of quantum physics [Abramsky and Coecke 2009; Coecke 2006] and has found several useful applications in simplification [Duncan et al. 2020] and equivalence checking [Peham et al. 2022] of quantum circuits. Recent work [Hadzihasanovic et al. 2018; Jeandel et al. 2018; Vilmart 2019] provides some complete axiomatizations of the ZX-calculus. Diagrammatic proof assistant and library have been introduced to realizing ZX-calculus [Kissinger and Zamdzhiev 2015] and automatically reasoning about large-scale quantum circuits and ZX-diagrams [Kissinger and van de Wetering 2020]. More recently, [Lehmann et al. 2022] proposed a verified ZX-calculus in the Coq that formally guarantees its correctness.

Language design. The design of quantum programming languages is an active area of research. Many existing works emphasize principled foundations, and develop semantics and type systems/program analyses to guarantee programs are well-behaved. Due to space limitations, we only cite a few examples. Quipper [Green et al. 2013], Q# [Svore et al. 2018] and Tower [Yuan and Carbin 2022] allow users to build complex data types from qubit; Silq [Bichsel et al. 2020] supports automatic uncomputation of quantum programs with the help of their elaborate type system; Twist [Yuan et al. 2022] develops a type system to help people manage entanglements in their quantum programs.

10 CONCLUSION

We have introduced CoqQ, the first verified quantum program verifier for a high-level quantum programming language. One main strength of CoqQ is to leverage state-of-the-art mathematical libraries, i.e. MathComp and MathComp Analysis. We have illustrated the benefits of CoqQ by mechanizing several examples from the literature. However, there remain many further steps to improve its expressiveness and usability. The first step is to introduce classical variables, so as to support reasoning about programs that mix quantum and classical computations and avoid metaprogramming. Another important step is to support data structures, leveraging a recent proposal [Yuan and Carbin 2022] to incorporate data structures in quantum programs. Naturally, it would also be interesting to provide better support for automating recurring mundane tasks. In the longer term, we would like to use CoqQ as the basis for a verified quantum software toolchain. A key element would be to develop a formally verified compiler from **qwhile** to quantum circuits (akin to CompCert [Leroy 2009] for the Verified Software Toolchain [Appel 2011]). Another exciting direction would be to build a certified abstract interpreter, following [Yu and Palsberg 2021] (akin to Verasco [Jourdan et al. 2015] for CompCert).

11 DATA-AVAILABILITY STATEMENT

The development of CoqQ can be found at https://github.com/coq-quantum/CoqQ. It is also available as a verified artifact in [Zhou et al. 2023].

ACKNOWLEDGMENTS

We thank Cyril Cohen for the suggestion on implementing labelled Dirac notation as non-dependent type and Christian Doczkal for discussing implementing the hierarchy of matrices and linear maps. This work was partly supported by the National Key R&D Program of China (Grant No: 2018YFA0306701), the National Natural Science Foundation of China (Grant No: 61832015).

REFERENCES

Samson Abramsky and Bob Coecke. 2009. Categorical quantum mechanics. *Handbook of quantum logic and quantum structures* 2 (2009), 261–325.

Dmitri Akatov. 2005. *The Logic of Quantum Program Verification.* Master's thesis. Oxford University Computing Laboratory.

Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, Luciano Bello, Yael Ben-Haim, David Bucher, Francisco Jose Cabrera-Hernádez, Jorge Carballo-Franquis, Adrian Chen, Chun-Fu Chen, Jerry M. Chow, Antonio D. Córcoles-Gonzales, Abigail J. Cross, Andrew Cross, Juan Cruz-Benito, Chris Culver, Salvador De La Puente González, Enrique De La Torre, Delton Ding, Eugene Dumitrescu, Ivan Duran, Pieter Eendebak, Mark Everitt, Ismael Faro Sertage, Albert Frisch, Andreas Fuhrer, Jay Gambetta, Borja Godoy Gago, Juan Gomez-Mosquera, Donny Greenberg, Ikko Hamamura, Vojtech Havlicek, Joe Hellmers, Łukasz Herok, Hiroshi Horii, Shaohan Hu, Takashi Imamichi, Toshinari Itoko, Ali Javadi-Abhari, Naoki Kanazawa, Anton Karazeev, Kevin Krsulich, Peng Liu, Yang Luh, Yunho Maeng, Manoel Marques, Francisco Jose Martín-Fernández, Douglas T. McClure, David McKay, Srujan Meesala, Antonio Mezzacapo, Nikolaj Moll, Diego Moreda Rodríguez, Giacomo Nannicini, Paul Nation, Pauline Ollitrault, Lee James O'Riordan, Hanhee Paik, Jesús Pérez, Anna Phan, Marco Pistoia, Viktor Prutyanov, Max Reuter, Julia Rice, Abdón Rodríguez Davila, Raymond Harry Putra Rudy, Mingi Ryu, Ninad Sathaye, Chris Schnabel, Eddie Schoute, Kanav Setia, Yunong Shi, Adenilton Silva, Yukio Siraichi, Seyon Sivarajah, John A. Smolin, Mathias Soeken, Hitomi Takahashi, Ivano Tavernelli, Charles Taylor, Pete Taylour, Kenso Trabing, Matthew Treinish, Wes Turner, Desiree Vogt-Lee, Christophe Vuillot, Jonathan A. Wildstrom, Jessica Wilson, Erick Winston, Christopher Wood, Stephen Wood, Stefan Wörner, Ismail Yunus Akhalwaya, and Christa Zoufal. 2019. Qiskit: An Open-source Framework for Quantum Computing. https://doi.org/10.5281/zenodo.2562110

Matthew Amy. 2018. Towards Large-scale Functional Verification of Universal Quantum Circuits. In *Proceedings 15th International Conference on Quantum Physics and Logic, QPL 2018, Halifax, Canada, 3-7th June 2018 (EPTCS, Vol. 287)*, Peter Selinger and Giulio Chiribella (Eds.). 1–21. https://doi.org/10.4204/EPTCS.287.1

Andrew W Appel. 2011. Verified software toolchain. In *European Symposium on Programming*. Springer, 1–17.

Alexandru Baltag and Sonja Smets. 2006. LQP: the dynamic logic of quantum information. *Mathematical structures in computer science* 16, 3 (2006), 491–525.

Manuel Barbosa, Gilles Barthe, Xiong Fan, Benjamin Grégoire, Shih-Han Hung, Jonathan Katz, Pierre-Yves Strub, Xiaodi Wu, and Li Zhou. 2021. EasyPQC: Verifying Post-Quantum Cryptography. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 2564–2586. https://doi.org/10.1145/3460120.3484567

Gilles Barthe, Justin Hsu, Mingsheng Ying, Nengkun Yu, and Li Zhou. 2019. Relational Proofs for Quantum Programs. *Proc. ACM Program. Lang.* 4, POPL, Article 21 (December 2019), 29 pages. https://doi.org/10.1145/3371089

Alexander Bentkamp, Jasmin Christian Blanchette, and Dietrich Klakow. 2019. A formal proof of the expressiveness of deep learning. *Journal of Automated Reasoning* 63, 2 (2019), 347–368.

Benjamin Bichsel, Maximilian Baader, Timon RGehr, and Martin Vechev. 2020. Silq: A high-level quantum language with safe uncomputation and intuitive semantics. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. 286–300.

Anthony Bordg, Hanna Lachnitt, and Yijun He. 2021. Certified quantum computation in Isabelle/HOL. *Journal of Automated Reasoning* 65, 5 (2021), 691–709.

Sergey Bravyi, David Gosset, and Robert König. 2018. Quantum advantage with shallow circuits. *Science* 362, 6412 (2018), 308–311. https://doi.org/10.1126/science.aar3106

Olivier Brunet and Philippe Jorrand. 2004. Dynamic quantum logic for quantum programs. *International Journal of Quantum Information* 2, 01 (2004), 45–54. https://doi.org/10.1142/S0219749904000067

Jose Manuel Rodriguez Caballero and Dominique Unruh. 2021. Complex Bounded Operators. *Archive of Formal Proofs (Sept. 2021). https://isa-afp.org/entries/Complex_Bounded_Operators.html, Formal proof development. issn* (2021).

Christophe Chareton, Sébastien Bardin, François Bobot, Valentin Perrelle, and Benoît Valiron. 2021. An Automated Deductive Verification Framework for Circuit-building Quantum Programs. In *Programming Languages and Systems*, Nobuko Yoshida (Ed.). Springer International Publishing, Cham, 148–177. https://doi.org/10.1007/978-3-030-72019-3_6

Bob Coecke. 2006. Kindergarten quantum mechanics: Lecture notes. In *AIP Conference Proceedings*, Vol. 810. American Institute of Physics, 81–98.

Bob Coecke and Ross Duncan. 2011. Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics* 13, 4 (2011), 043016.

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

Ross Duncan, Aleks Kissinger, Simon Perdrix, and John Van De Wetering. 2020. Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus. *Quantum* 4 (2020), 279.

Yuan Feng, Runyao Duan, Zhengfeng Ji, and Mingsheng Ying. 2007. Proof rules for the correctness of quantum programs. *Theoretical Computer Science* 386, 1-2 (2007), 151–166.

Yuan Feng and Mingsheng Ying. 2021. Quantum Hoare Logic with Classical Variables. *ACM Transactions on Quantum Computing* 2, 4, Article 16 (December 2021), 43 pages. https://doi.org/10.1145/3456877

Georges Gonthier, Beta Ziliani, Aleksandar Nanevski, and Derek Dreyer. 2013. How to make ad hoc proof automation less ad hoc. *Journal of Functional Programming* 23, 4 (2013), 357–401. https://doi.org/10.1017/S0956796813000051

Alexander S Green, Peter LeFanu Lumsdaine, Neil J Ross, Peter Selinger, and Benoît Valiron. 2013. Quipper: a scalable quantum programming language. In *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*. 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*. 212–219.

Amar Hadzihasanovic, Kang Feng Ng, and Quanlong Wang. 2018. Two complete axiomatisations of pure-state qubit quantum computing. In *Proceedings of the 33rd annual ACM/IEEE symposium on logic in computer science*. 502–511.

Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. 2009. Quantum Algorithm for Linear Systems of Equations. *Phys. Rev. Lett.* 103 (October 2009), 150502. Issue 15. https://doi.org/10.1103/PhysRevLett.103.150502

Kesha Hietala, Robert Rand, Shih-Han Hung, Liyi Li, and Michael Hicks. 2021a. Proving Quantum Programs Correct. In *12th International Conference on Interactive Theorem Proving (ITP 2021) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 193)*, Liron Cohen and Cezary Kaliszyk (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 21:1–21:19. https://doi.org/10.4230/LIPIcs.ITP.2021.21

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

Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. 2018. Diagrammatic reasoning beyond Clifford+ T quantum mechanics. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. 569–578.

Jacques-Henri Jourdan, Vincent Laporte, Sandrine Blazy, Xavier Leroy, and David Pichardie. 2015. A Formally-Verified C Static Analyzer. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, Sriram K. Rajamani and David Walker (Eds.). ACM, 247–259.

Ralf Jung, Jacques-Henri Jourdan, Robbert Krebbers, and Derek Dreyer. 2018a. RustBelt: securing the foundations of the rust programming language. *Proc. ACM Program. Lang.* 2, POPL (2018), 66:1–66:34. https://doi.org/10.1145/3158154

Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Ales Bizjak, Lars Birkedal, and Derek Dreyer. 2018b. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *J. Funct. Program.* 28 (2018), e20. https://doi.org/10.1017/S0956796818000151

Yoshihiko Kakutani. 2009. A logic for formal verification of quantum programs. In *Annual Asian Computing Science Conference*. Springer, 79–93.

Aleks Kissinger and John van de Wetering. 2020. PyZX: Large Scale Automated Diagrammatic Reasoning. In *Proceedings 17th International Conference on Quantum Physics and Logic, QPL 2020, Paris, France, June 2-6th, 2020 (EPTCS, Vol. 318)*, Benoît Valiron, Shane Mansfield, Pablo Arrighi, and Prakash Panangaden (Eds.). 229–241. https://doi.org/10.4204/eptcs.318.14

Aleks Kissinger and Vladimir Zamdzhiev. 2015. Quantomatic: A Proof Assistant for Diagrammatic Reasoning. In *Automated Deduction - CADE-25*, Amy P. Felty and Aart Middeldorp (Eds.). Springer International Publishing, Cham, 326–336. https://doi.org/10.1007/978-3-319-21401-6_22

Alexei Y. Kitaev. 1996. Quantum measurements and the Abelian Stabilizer Problem. *Electron. Colloquium Comput. Complex.* TR96-003 (1996). ECCC:quant-ph/9511026 https://eccc.weizmann.ac.il/eccc-reports/1996/TR96-003/index.html

Xuan-Bach Le, Shang-Wei Lin, Jun Sun, and David Sanan. 2022. A Quantum Interpretation of Separating Conjunction for Local Reasoning of Quantum Programs Based on Separation Logic. *Proc. ACM Program. Lang.* 6, POPL, Article 36 (jan 2022), 27 pages. https://doi.org/10.1145/3498697

Adrian Lehmann, Ben Caldwell, and Robert Rand. 2022. VyZX : A Vision for Verifying the ZX Calculus. In *Proceedings 19th International Conference on Quantum Physics and Logic, QPL 2022, Oxford, England, 27 June – 1 July 2012, to be published*, Bob Coecke and Matthew Leifer (Eds.). arXiv:arXiv preprint arXiv:1908.08963 https://arxiv.org/abs/2205.05781

Xavier Leroy. 2009. Formal verification of a realistic compiler. *Commun. ACM* 52, 7 (2009), 107–115.

Yangjia Li and Dominique Unruh. 2021. Quantum Relational Hoare Logic with Expectations. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 198)*, Nikhil Bansal, Emanuela Merelli, and James Worrell (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 136:1–136:20. https://doi.org/10.4230/LIPIcs.ICALP.2021.136

Junyi Liu, Bohua Zhan, Shuling Wang, Shenggang Ying, Tao Liu, Yangjia Li, Mingsheng Ying, and Naijun Zhan. 2019. Formal verification of quantum algorithms using quantum Hoare logic. In *International conference on computer aided verification*. Springer, 187–207.

Chris Lomont. 2004. The Hidden Subgroup Problem - Review and Open Problems. https://doi.org/10.48550/arxiv.quant-ph/0411037

Assia Mahboubi and Enrico Tassi. 2021. *Mathematical Components*. Zenodo. https://doi.org/10.5281/zenodo.4457887

Michael A Nielsen and Isaac Chuang. 2002. Quantum computation and quantum information.

Jennifer Paykin, Robert Rand, and Steve Zdancewic. 2017. QWIRE: a core language for quantum circuits. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, Giuseppe Castagna and Andrew D. Gordon (Eds.). ACM, 846–858. http://dl.acm.org/citation.cfm?id=3009894

Tom Peham, Lukas Burgholzer, and Robert Wille. 2022. Equivalence Checking of Quantum Circuits with the ZX-Calculus. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* (2022), 1–1. https://doi.org/10.1109/JETCAS.2022.3202204

Yuxiang Peng, Mingsheng Ying, and Xiaodi Wu. 2022. Algebraic reasoning of Quantum programs via non-idempotent Kleene algebra. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 657–670.

Robert Rand, Jennifer Paykin, Dong-Ho Lee, and Steve Zdancewic. 2018b. ReQWIRE: Reasoning about Reversible Quantum Circuits. In *Proceedings 15th International Conference on Quantum Physics and Logic, QPL 2018, Halifax, Canada, 3-7th June 2018 (EPTCS, Vol. 287)*, Peter Selinger and Giulio Chiribella (Eds.). 299–312. https://doi.org/10.4204/EPTCS.287.17

Robert Rand, Jennifer Paykin, and Steve Zdancewic. 2017. QWIRE Practice: Formal Verification of Quantum Circuits in Coq. In *Proceedings 14th International Conference on Quantum Physics and Logic, QPL 2017, Nijmegen, The Netherlands, 3-7 July 2017. (EPTCS, Vol. 266)*, Bob Coecke and Aleks Kissinger (Eds.). 119–132. https://doi.org/10.4204/EPTCS.266.8

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

Jean-Pierre Serre. 1977. *Linear representations of finite groups*. Vol. 42. Springer.

Kartik Singhal. 2020. *Quantum Hoare Type Theory*. Master's thesis. University of Chicago, Chicago, IL. arXiv:2012.02154 https://ks.cs.uchicago.edu/publication/qhtt-masters/ See also: https://ks.cs.uchicago.edu/publication/qhtt/.

Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. 2018. Q#: Enabling scalable quantum computing and development with a high-level dsl. In *Proceedings of the Real World Domain Specific Languages Workshop 2018 (RWDSL 2018)*. ACM, New York, NY, USA, Article 7, 10 pages. https://doi.org/10.1145/3183895.3183901

The Cirq Developers. 2018. quantumlib/Cirq: A Python framework for creating, editing, and invoking Noisy Intermediate Scale Quantum (NISQ) circuits. https://github.com/quantumlib/Cirq

The Coq Development Team. 2022. The Coq Proof Assistant. https://doi.org/10.5281/zenodo.5846982

The MathComp Analysis Development Team. 2022. MathComp-Analysis: Mathematical Components compliant Analysis Library. https://github.com/math-comp/analysis. Since 2017. Version 0.5.1.

René Thiemann and Akihisa Yamada. 2016. Formalizing Jordan normal forms in Isabelle/HOL. In *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*. 88–99.

Dominique Unruh. 2019a. Quantum Hoare Logic with Ghost Variables. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)* (Vancouver, Canada). ACM, New York, NY, USA, 1–13. https://doi.org/10.1109/LICS.2019.8785779

Dominique Unruh. 2019b. Quantum Relational Hoare Logic. *Proc. ACM Program. Lang.* 3, POPL, Article 33 (jan 2019), 31 pages. https://doi.org/10.1145/3290346

Dominique Unruh. 2020. Post-Quantum Verification of Fujisaki-Okamoto. In *Advances in Cryptology – ASIACRYPT 2020*, Shiho Moriai and Huaxiong Wang (Eds.). Springer International Publishing, Cham, 321–352.

Dominique Unruh. 2021. Quantum and classical registers. In *The Second International Workshop on Programming Languages for Quantum Computing (PLanQC 2021)*. https://arxiv.org/abs/2105.10914

Renaud Vilmart. 2019. A near-minimal axiomatisation of zx-calculus for pure qubit quantum mechanics. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 1–10.

Mingsheng Ying. 2011. Floyd-Hoare logic for quantum programs. *ACM Trans. Program. Lang. Syst.* 33, 6 (2011), 19:1–19:49. https://doi.org/10.1145/2049706.2049708

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

Mingsheng Ying. 2019. Toward automatic verification of quantum programs. *Formal Aspects of Computing* 31, 1 (01 Feb 2019), 3–25. https://doi.org/10.1007/s00165-018-0465-3

Mingsheng Ying, Li Zhou, and Yangjia Li. 2018. Reasoning about Parallel Quantum Programs. https://doi.org/10.48550/arxiv.1810.11334

Mingsheng Ying, Li Zhou, Yangjia Li, and Yuan Feng. 2022. A proof system for disjoint parallel quantum programs. *Theoretical Computer Science* 897 (2022), 164–184. https://doi.org/10.1016/j.tcs.2021.10.025

Nengkun Yu. 2019. Quantum Temporal Logic. (2019). https://doi.org/10.48550/arxiv.1908.00158

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

Charles Yuan and Michael Carbin. 2022. Tower: Data Structures in Quantum Superposition. *Proc. ACM Program. Lang.* 6, OOPSLA2, Article 134 (October 2022), 41 pages. https://doi.org/10.1145/3563297

Charles Yuan, Christopher McNally, and Michael Carbin. 2022. Twist: Sound Reasoning for Purity and Entanglement in Quantum Programs. *Proc. ACM Program. Lang.* 6, POPL, Article 30 (jan 2022), 32 pages. https://doi.org/10.1145/3498691

Li Zhou, Gilles Barthe, Justin Hsu, Mingsheng Ying, and Nengkun Yu. 2021. A Quantum Interpretation of Bunched Logic amp; Quantum Separation Logic. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 1–14.

https://doi.org/10.1109/LICS52264.2021.9470673

Li Zhou, Gilles Barthe, Pierre-Yves Strub, Junyi Liu, and Mingsheng Ying. 2023. CoqQ: Foundational Verification of Quantum Programs. https://doi.org/10.1145/3554343

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* (Phoenix, AZ, USA) *(PLDI 2019)*. Association for Computing Machinery, New York, NY, USA, 1149–1162. https://doi.org/10.1145/3314221.3314584

Received 2022-07-07; accepted 2022-11-07