

# Local Reasoning about Probabilistic Behaviour for Classical–Quantum Programs\*

HUILING WU, Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, China  
YUXIN DENG, Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, China  
MING XU, Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, China

Verifying the functional correctness of programs with both classical and quantum constructs is a challenging task. The presence of probabilistic behaviour entailed by quantum measurements and unbounded while loops complicate the verification task greatly. We propose a new quantum Hoare logic for local reasoning about probabilistic behaviour by introducing distribution formulas to specify probabilistic properties. We show that the proof rules in the logic are sound with respect to a denotational semantics. To demonstrate the effectiveness of the logic, we formally verify the correctness of non-trivial quantum algorithms including the HHL and Shor’s algorithms. Moreover, we embed our logic into the proof assistant Coq. The resulting logical framework, called CoqQLR, can facilitate semi-automated reasoning about classical–quantum programs.

CCS Concepts: • **Theory of computation** → **Semantics and reasoning; Logic.**

Additional Key Words and Phrases: Quantum computing, Program verification, Hoare logic, Separation logic, Local reasoning.

## ACM Reference Format:

Huiling Wu, Yuxin Deng, and Ming Xu. XXXX. Local Reasoning about Probabilistic Behaviour for Classical–Quantum Programs. *ACM Trans. Softw. Eng. Methodol.* X, X, Article X ( XXXX), 34 pages. <https://doi.org/XXXXXXX.XXXXXXX>

## 1 Introduction

Programming is an error-prone activity, and the situation is even worse for quantum programming, which is far less intuitive than classical computing. Therefore, developing verification and analysis techniques to ensure the correctness of quantum programs is an even more important task than that for classical programs.

Hoare logic [8] is probably the most widely used program logic to verify the correctness of programs. It is useful for reasoning about deterministic and probabilistic programs. A lot of efforts have been made to reuse the nice idea to verify quantum programs. Ying [21, 22] was the first to establish a sound and relatively complete quantum Hoare logic to reason about pure quantum programs, i.e., quantum programs without classical variables. This work triggered a series of research in this direction. For example, Zhou et al. [27] proposed an applied quantum Hoare logic by only using projections as preconditions and postconditions, which makes the practical use of quantum Hoare logic much easier. Barthe et al. [1] extended quantum Hoare logic to relational verification by introducing a quantum analogue of probabilistic couplings. Li and Unruh [11] defined a quantum relational Hoare logic with expectations in pre- and postconditions. Formalization of

\*This work is an extended version of a paper that has appeared in the proceedings of the 25th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2024).

Corresponding author: Yuxin Deng

Authors’ Contact Information: Huiling Wu, Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai, China, 52275902009@stu.ecnu.edu.cn; Yuxin Deng, Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai, China, yxdeng@sei.ecnu.edu.cn; Ming Xu, Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai, China, mxu@cs.ecnu.edu.cn.

quantum Hoare logic in proof assistants such as Isabelle/HOL [14] and Coq [2] was accomplished in [12] and [26], respectively. Ying et al. [24] defined a class of disjoint parallel quantum programs and generalised the logic in [21] to this setting. Zhou et al. [25] developed a quantum separation logic for local reasoning. However, all the work mentioned above cannot deal with programs that have both classical and quantum data.

In order to verify quantum programs found in almost all practical quantum programming frameworks such as Qiskit<sup>1</sup>, Q<sup>#2</sup>, Cirq<sup>3</sup>, etc., we have to explicitly consider programs with both classical and quantum constructs. Verification techniques for this kind of hybrid programs have been put forward in the literature [3–6, 9, 10, 18, 19]. For example, Chadha et al. [3] proposed an ensemble exogenous quantum propositional logic for a simple quantum language with bounded iteration. The expressiveness of the language is very limited, and algorithms involving unbounded while loops such as the HHL and Shor’s algorithms [7, 17] cannot be described. Kakutani [9] presented a quantum Hoare logic for an imperative language with while loops, but the rule for them has no invariance condition. Instead, an infinite sequence of assertions has to be used. Unruh introduced a quantum Hoare logic with ghost variables to express properties such as that a quantum variable is disentangled with others [18] and a relational Hoare logic [19] for security analysis of post-quantum cryptography and quantum protocols. Deng and Feng [4] provided an abstract and a concrete proof system for classical–quantum programs, with the former being sound and relatively complete, while the latter being sound. Feng and Ying [6] introduced classical–quantum assertions, which are a class of mappings from classical states to quantum predicates, to analyse both classical and quantum properties. The approach was extended to verify distributed quantum programs in [5]. However, except for [10], all the work above offers no support for local reasoning, which is an obvious drawback. In the case that we have a large quantum register but only a few qubits are modified, it is awkward to always reason about the global states of the quantum register. Based on this observation, Le et al. [10] provided an interesting quantum interpretation of the separating conjunction, so to infuse separation logic into a Hoare-style framework and thus support local reasoning. However, a weakness of their approach is that it cannot handle probabilistic behaviour, which exists inherently in quantum programs, in a satisfactory way. Let us illustrate this with a simple example.

*Example 1.1.* The program **addM** defined below first initialises two qubits  $q_0$  and  $q_1$ , and then applies the Hadamard gate  $H$  to each of them. By measuring them with the measurement operators  $|0\rangle\langle 0|$  and  $|1\rangle\langle 1|$ , we add the measurement results and assign the sum to the variable  $v$ .

$$\begin{aligned} \mathbf{addM} &\triangleq q_0 := |0\rangle; H[q_0]; \\ &\quad q_1 := |0\rangle; H[q_1]; \\ &\quad v_0 := M[q_0]; \\ &\quad v_1 := M[q_1]; \\ &\quad v := v_0 + v_1 \end{aligned}$$

Since the classical variable  $v_0$  takes either 0 or 1 with equal chance, and similarly for  $v_1$ , the probability that variable  $v$  is assigned to 1 should be exactly  $\frac{1}{2}$ . However, in the program logic in [10], this property cannot be specified.

We propose a novel quantum Hoare logic for a classical–quantum language. Two distribution formulas  $\oplus_{i \in I} p_i \cdot F_i$  and  $\oplus_{i \in I} F_i$  are introduced. A program state  $\mu$ , which is a partial density operator

<sup>1</sup><https://qiskit.org>

<sup>2</sup><https://github.com/microsoft/qsharp-language>

<sup>3</sup><https://github.com/quantumlib/Cirq>

valued distribution (POVD) [4], satisfies the formula  $\bigoplus_{i \in I} p_i \cdot F_i$  if  $\mu$  can be split into the weighted sum of some  $\mu_i$  with weights  $p_i$ . Each  $\mu_i$  has the same size as  $\mu$  and satisfies the formula  $F_i$  whenever  $p_i > 0$ . A state  $\mu$  satisfies the formula  $\bigoplus_{i \in I} F_i$  if there exists a collection of probabilities  $\{p_i\}_{i \in I}$  with  $\sum_{i \in I} p_i = 1$  such that  $\bigoplus_{i \in I} p_i \cdot F_i$  can be satisfied. In other words, the splitting of  $\mu$  does not necessarily follow a fixed set of weights. With distribution formulas, we can conveniently reason about the probabilistic behaviour mentioned in Example 1.1 (more details will be discussed in Example 4.8), and give an invariance condition in the proof rule for while loops. In addition, we adopt the labelled Dirac notation emphasised in [26] to facilitate local reasoning. Our program logic is shown to be sound and can be used to prove the correctness of non-trivial quantum algorithms including the HHL and Shor’s algorithms. Furthermore, recognizing the complexity and laboriousness of manual reasoning about these algorithms, we embed our logic into the Coq proof assistant. The resulting logical framework, called CoqQLR, can facilitate semi-automated reasoning. Each rule in our proof system is formulated as a theorem and rigorously validated for soundness within the Coq environment. Ultimately, we formally demonstrate the correctness of the aforementioned algorithms with machine-checkable proofs in Coq.

Therefore, the main contributions of the current work include the following aspects:

- We propose to use distribution formulas in a new quantum Hoare logic to specify the probabilistic behaviour of classical–quantum programs. Distribution formulas are useful to give an invariance condition in the proof rule for while loops, so to avoid an infinite sequence of assertions in the rule.
- We prove the soundness of our logic that enables local reasoning in the spirit of separation logic. A crucial element of the proof is the property that the denotation of each assertion constitutes a closed set.
- We demonstrate the effectiveness of the logic by proving the correctness of the HHL and Shor’s algorithms.
- We integrate our logic into the proof assistant Coq, enabling semi-automated reasoning about classical–quantum programs.

The rest of the paper is structured as follows. In Section 2 we recall some basic notations about quantum computing. In Section 3 we review the syntax and denotational semantics of a classical–quantum imperative language considered in [4]. In Section 4 we define an assertion language and propose a proof system for local reasoning about quantum programs. We also prove the soundness of the system. In Section 5 we apply our program logic to verify the HHL and Shor’s algorithms. Section 6 is dedicated to the Coq formalization of our logic. Finally, we conclude in Section 7 and discuss possible future work. Our Coq development is available at the following link:

<https://github.com/fox9909/CoqQLR.git>

## 2 Preliminaries

We briefly recall some basic notations from linear algebra and quantum mechanics which are needed in this paper. For more details, we refer to [13].

A *Hilbert space*  $\mathcal{H}$  is a complete vector space with an inner product  $\langle \cdot | \cdot \rangle : \mathcal{H} \times \mathcal{H} \rightarrow \mathbb{C}$  such that

- (1)  $\langle \psi | \psi \rangle \geq 0$  for any  $|\psi\rangle \in \mathcal{H}$ , with equality if and only if  $|\psi\rangle = 0$ ,
- (2)  $\langle \phi | \psi \rangle = \langle \psi | \phi \rangle^*$ ,
- (3)  $\langle \phi | \sum_i c_i |\psi_i\rangle = \sum_i c_i \langle \phi | \psi_i \rangle$ ,

where  $\mathbb{C}$  is the set of complex numbers, and for each  $c \in \mathbb{C}$ ,  $c^*$  stands for the complex conjugate of  $c$ . For any vector  $|\psi\rangle \in \mathcal{H}$ , its length  $\| |\psi\rangle \|$  is defined to be  $\sqrt{\langle \psi | \psi \rangle}$ , and it is said to be *normalised* if  $\| |\psi\rangle \| = 1$ . Two vectors  $|\psi\rangle$  and  $|\phi\rangle$  are *orthogonal* if  $\langle \psi | \phi \rangle = 0$ . An *orthonormal basis* of a Hilbert space  $\mathcal{H}$  is a basis  $\{|i\rangle\}$  where each  $|i\rangle$  is normalised and any pair of them are orthogonal.

Let  $\mathcal{L}(\mathcal{H})$  be the set of linear operators on  $\mathcal{H}$ . For any  $A \in \mathcal{L}(\mathcal{H})$ ,  $A$  is *Hermitian* if  $A^\dagger = A$  where  $A^\dagger$  is the adjoint operator of  $A$  such that  $\langle \psi | A^\dagger | \phi \rangle = \langle \phi | A | \psi \rangle^*$  for any  $|\psi\rangle, |\phi\rangle \in \mathcal{H}$ . A linear operator  $A \in \mathcal{L}(\mathcal{H})$  is *unitary* if  $A^\dagger A = AA^\dagger = I_{\mathcal{H}}$  where  $I_{\mathcal{H}}$  is the identity operator on  $\mathcal{H}$ . The *trace* of  $A$  is defined as  $\text{tr}(A) = \sum_i \langle i | A | i \rangle$  for some given orthonormal basis  $\{|i\rangle\}$  of  $\mathcal{H}$ . A linear operator  $A \in \mathcal{L}(\mathcal{H})$  is *positive* if  $\langle \phi | A | \phi \rangle \geq 0$  for any vector  $|\phi\rangle \in \mathcal{H}$ . The *Löwner order*  $\sqsubseteq$  on the set of Hermitian operators on  $\mathcal{H}$  is defined by letting  $A \sqsubseteq B$  if and only if  $B - A$  is positive.

Let  $\mathcal{H}_1$  and  $\mathcal{H}_2$  be two Hilbert spaces. Their *tensor product*  $\mathcal{H}_1 \otimes \mathcal{H}_2$  is defined as a vector space consisting of linear combinations of the vectors  $|\psi_1 \psi_2\rangle = |\psi_1\rangle |\psi_2\rangle = |\psi_1\rangle \otimes |\psi_2\rangle$  with  $|\psi_1\rangle \in \mathcal{H}_1$  and  $|\psi_2\rangle \in \mathcal{H}_2$ . Here the tensor product of two vectors is defined by a new vector such that

$$\left( \sum_i \lambda_i |\psi_i\rangle \right) \otimes \left( \sum_j \mu_j |\phi_j\rangle \right) = \sum_{i,j} \lambda_i \mu_j |\psi_i\rangle \otimes |\phi_j\rangle.$$

Then  $\mathcal{H}_1 \otimes \mathcal{H}_2$  is also a Hilbert space where the inner product is defined as the following: for any  $|\psi_1\rangle, |\phi_1\rangle \in \mathcal{H}_1$  and  $|\psi_2\rangle, |\phi_2\rangle \in \mathcal{H}_2$ ,

$$\langle \psi_1 \otimes \psi_2 | \phi_1 \otimes \phi_2 \rangle = \langle \psi_1 | \phi_1 \rangle_{\mathcal{H}_1} \langle \psi_2 | \phi_2 \rangle_{\mathcal{H}_2}$$

where  $\langle \cdot | \cdot \rangle_{\mathcal{H}_i}$  is the inner product of  $\mathcal{H}_i$ . Given  $\mathcal{H}_1$  and  $\mathcal{H}_2$ , the *partial trace* with respect to  $\mathcal{H}_2$ , written  $\text{tr}_{\mathcal{H}_2}$ , is a linear mapping from  $\mathcal{L}(\mathcal{H}_1 \otimes \mathcal{H}_2)$  to  $\mathcal{L}(\mathcal{H}_1)$  such that for any  $|\psi_1\rangle, |\phi_1\rangle \in \mathcal{H}_1$  and  $|\psi_2\rangle, |\phi_2\rangle \in \mathcal{H}_2$ ,

$$\text{tr}_{\mathcal{H}_2}(|\psi_1\rangle\langle\phi_1| \otimes |\psi_2\rangle\langle\phi_2|) = \langle\psi_2|\phi_2\rangle |\psi_1\rangle\langle\phi_1|.$$

By applying quantum gates to qubits, we can change their states. For example, the Hadamard gate ( $H$  gate) can be applied on a single qubit, while the controlled-NOT gate ( $CNOT$  gate) can be applied on two qubits. Their representations in terms of matrices are given as

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad \text{and} \quad CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

According to von Neumann's formalism of quantum mechanics [20], an isolated physical system is associated with a Hilbert space which is called the *state space* of the system. A *pure state* of a quantum system is a normalised vector in its state space, and a *mixed state* is represented by a density operator on the state space. Here a *density operator*  $\rho$  on Hilbert space  $\mathcal{H}$  is a positive linear operator such that  $\text{tr}(\rho) = 1$ . A *partial density operator*  $\rho$  is a positive linear operator with  $\text{tr}(\rho) \leq 1$ .

The evolution of a closed quantum system is described by a unitary operator on its state space: if the states of the system at times  $t_1$  and  $t_2$  are  $\rho_1$  and  $\rho_2$ , respectively, then  $\rho_2 = U \rho_1 U^\dagger$  for some unitary operator  $U$  which depends only on  $t_1$  and  $t_2$ . A quantum *measurement* is described by a collection  $\{M_m\}$  of positive operators, called measurement operators, where the indices  $m$  refer to the measurement outcomes. It is required that the measurement operators satisfy the completeness equation  $\sum_m M_m^\dagger M_m = I_{\mathcal{H}}$ . If the system is in state  $\rho$ , then the probability that measurement result  $m$  occurs is given by

$$p(m) = \text{tr}(M_m^\dagger M_m \rho),$$

and the state of the post-measurement system is  $M_m \rho M_m^\dagger / p(m)$ .

Table 1. Syntax of quantum programs

(Aexp)	$a$	$::=$	$n \mid x, y, \dots \mid f_m(a, \dots, a)$
(Bexp)	$b$	$::=$	$\mathbf{true} \mid \mathbf{false} \mid P_m(a, \dots, a) \mid b \wedge b \mid \neg b \mid \forall x. b$
(Com)	$c$	$::=$	$\mathbf{skip} \mid \mathbf{abort} \mid x := a \mid c; c$ $\mid \mathbf{if } b \mathbf{ then } c \mathbf{ else } c \mathbf{ fi} \mid \mathbf{while } b \mathbf{ do } c \mathbf{ od}$ $\mid q :=  0\rangle \mid U[\bar{q}] \mid x := M[\bar{q}]$

### 3 A Classical–Quantum Language

Here, we recall the simple classical–quantum imperative language **QIMP** as defined in [4]. It is essentially similar to a few imperative languages considered in the literature [6, 16, 19, 23]. We introduce its syntax and denotational semantics.

#### 3.1 Syntax

We assume three types of data in our language: **Bool** for booleans, **Int** for integers, and **Qbt** for quantum bits (qubits). Let  $\mathbb{Z}$  be the set of integer constants, ranged over by  $n$ . Let **Cvar**, ranged over by  $x, y, \dots$ , be the set of classical variables, and **Qvar**, ranged over by  $q, q', \dots$ , the set of quantum variables. It is assumed that both **Cvar** and **Qvar** are countable. We assume a set **Aexp** of arithmetic expressions over **Int**, which includes **Cvar** as a subset and is ranged over by  $a, a', \dots$ , and a set of boolean-valued expressions **Bexp**, ranged over by  $b, b', \dots$ , with the usual boolean constants **true**, **false** and boolean connectives such as  $\neg, \wedge$  and  $\vee$ . We assume a set of arithmetic functions (e.g.  $+$ ,  $-$ ,  $*$ , etc.) ranged over by the symbol  $f_m$ , and a set of boolean predicates (e.g.  $=, \leq, \geq$ , etc.) ranged over by  $P_m$ , where  $m$  indicates the number of variables involved. We further assume that only classical variables can occur freely in both arithmetic and boolean expressions.

We let  $U$  range over unitary operators, which can be user-defined matrices or built in if the language is implemented. For example, a concrete  $U$  could be the 1-qubit Hadamard operator  $H$ , or the 2-qubit controlled-NOT operator  $CNOT$ , etc. Similarly, we write  $M$  for the measurement described by a collection  $\{M_i\}$  of measurement operators, with each index  $i$  representing a measurement outcome. For example, to describe the measurement of the qubit referred to by variable  $q$  in the computational basis, we can write  $M := \{M_0, M_1\}$ , where  $M_0 = |0\rangle_q \langle 0|$  and  $M_1 = |1\rangle_q \langle 1|$ .

Sometimes, we use metavariables which are primed or subscripted, e.g.  $x', x_0$  for classical variables. We abbreviate a tuple of quantum variables  $\langle q_1, \dots, q_n \rangle$  as  $\bar{q}$  if the length  $n$  of the tuple is not important. If two tuples of quantum variables  $\bar{q}$  and  $\bar{q}'$  are disjoint, where  $\bar{q} = \langle q_1, \dots, q_n \rangle$  and  $\bar{q}' = \langle q_{n+1}, q_{n+2}, \dots, q_{n+m} \rangle$ , then their concatenation is a larger tuple  $\bar{q}\bar{q}' = \langle q_1, \dots, q_n, q_{n+1}, \dots, q_{n+m} \rangle$ . If no confusion arises, we occasionally use a tuple to stand for a set.

The formation rules for arithmetic and boolean expressions as well as commands are defined in Table 1. An arithmetic expression can be an integer, a variable, or built from other arithmetic expressions by some arithmetic functions. A boolean expression can be a boolean constant, built from arithmetic expressions by some boolean predicates or formed by using the usual boolean operations. A command can be a skip statement, an abort statement, a classical assignment, a conditional statement, or a while-loop, as in many classical imperative languages. The command **abort** represents the unsuccessful termination of programs. In addition, there are three commands that involve quantum data. The command  $q := |0\rangle$  initialises the qubit referred to by variable  $q$  to be the basis state  $|0\rangle$ . The command  $U[\bar{q}]$  applies the unitary operator  $U$  to the quantum system referred to by  $\bar{q}$ . The command  $x := M[\bar{q}]$  performs a measurement  $M$  on  $\bar{q}$  and assigns the

measurement outcome to  $x$ . It differs from a classical assignment because the measurement  $M$  may change the quantum state of  $\bar{q}$ , besides the fact that the value of  $x$  is updated.

For convenience, we further define the following syntactic sugar for the initialization of a sequence of quantum variables:  $\bar{q} = |0\rangle^{\otimes n}$ , where  $\bar{q} = \langle q_1, \dots, q_n \rangle$ , is an abbreviation of the commands:

$$q_1 = |0\rangle; q_2 = |0\rangle; \dots; q_n := |0\rangle.$$

### 3.2 Denotational Semantics

In the presence of classical and quantum variables, the execution of a **QIMP** program involves two types of states: classical states and quantum states.

As usual, a classical state is a function  $\sigma : \mathbf{Cvar} \rightarrow \mathbb{Z}$  from classical variables to integers, where  $\sigma(x)$  represents the value of classical variable  $x$ . For each quantum variable  $q \in \mathbf{Qvar}$ , we assume a 2-dimensional Hilbert space  $\mathcal{H}_q$  to be the state space of the  $q$ -system. For any finite subset  $V$  of  $\mathbf{Qvar}$ , we denote

$$\mathcal{H}_V = \bigotimes_{q \in V} \mathcal{H}_q.$$

That is,  $\mathcal{H}_V$  is the Hilbert space spanned by tensor products of the individual state spaces of the quantum variables in  $V$ . Throughout the paper, when we refer to a subset of  $\mathbf{Qvar}$ , it is assumed to be finite. Given  $V \subseteq \mathbf{Qvar}$ , the set of *quantum states* consists of all partial density operators in the space  $\mathcal{H}_V$ , denoted by  $\mathcal{D}^-(\mathcal{H}_V)$ . For any quantum state  $\rho \in \mathcal{D}^-(\mathcal{H}_V)$ , its *domain* is defined as  $\text{dom}(\rho) \triangleq V$ . Sometimes, we simply use the notation  $\rho_{\bar{q}}$  to denote a quantum state  $\rho$  such that  $\rho \in \mathcal{D}^-(\mathcal{H}_{\bar{q}})$ . Similarly, we use  $\rho_V$  to denote some  $\rho \in \mathcal{D}^-(\mathcal{H}_V)$ . Let  $V'$  be a set of quantum variables with  $V' \subseteq V$ . We write  $\rho|_{V'}$  for the reduced density operator  $\text{tr}_{V \setminus V'}(\rho)$  obtained by restricting  $\rho$  to  $V'$ .

For any set  $V \subseteq \mathbf{Qvar}$ , a *machine state* in the space  $\mathcal{H}_V$  is a pair  $\langle \sigma, \rho \rangle$ , where  $\sigma$  is a classical state and  $\rho$  is a quantum state in the space  $\mathcal{H}_V$ . Similarly, for any state  $s$  in the space  $\mathcal{H}_V$ , its *domain* is defined as  $\text{dom}(s) \triangleq V$ . In the presence of measurements, we often need to consider an ensemble of states. For that purpose, we introduce a notion of distribution.

**Definition 3.1.** [4] Suppose  $V \subseteq \mathbf{Qvar}$  and  $\Sigma$  is the set of classical states, i.e., the set of functions of type  $\mathbf{Cvar} \rightarrow \mathbb{Z}$ . A *partial density operator valued distribution in the space  $\mathcal{H}_V$*  ( $\text{POVD}_V$ ) is a function  $\mu : \Sigma \rightarrow \mathcal{D}^-(\mathcal{H}_V)$  with  $\sum_{\sigma \in \Sigma} \text{tr}(\mu(\sigma)) \leq 1$ .

Intuitively, a  $\text{POVD}_V \mu$  represents a collection of machine states with domain  $V$ , where each classical state  $\sigma$  is associated with a quantum state  $\mu(\sigma)$ . We write  $\mathbf{POVD}_V$  for the set of  $\text{POVD}_V$ . The subscript  $V$  is utilized to indicate the specific space. When the context makes it clear or the space is not important, we may simply refer to it as  $\text{POVD}$ . This notation of  $\text{POVD}$  is called classical-quantum state in [6]. In this paper, we sometimes call it a distribution state and abbreviate it as distribution. If the collection has only one element  $\sigma$ , we explicitly write  $(\sigma, \mu(\sigma))$  for  $\mu$ . The support of  $\mu$ , written  $[\mu]$ , is the set  $\{\sigma \in \Sigma \mid \mu(\sigma) \neq 0\}$ . The size of  $\mu$ , written  $\|\mu\|$ , means  $\sum_{\sigma \in \Sigma} \text{tr}(\mu(\sigma))$ . We write  $\varepsilon$  for the special  $\text{POVD}$  whose support is the empty set. Similarly, for any distribution  $\mu$  in the space  $\mathcal{H}_V$ , its *domain* is defined as  $\text{dom}(\mu) \triangleq V$ . The notation  $\mu_V$  is used to denote the distribution  $\mu$  with domain  $V$ . Let  $V'$  be a subset of quantum variables such that  $V' \subseteq V$ . We denote the reduced distribution of  $\mu$  on  $V'$  by  $\mu|_{V'}$ , which is a  $\text{POVD}_{V'}$  defined by the equation  $\mu|_{V'}(\sigma) = (\mu(\sigma))|_{V'}$ . We can also define the scaling of a distribution by letting  $(p \cdot \mu)(\sigma) = p \cdot (\mu(\sigma))$  for some probability  $p$  and the addition of two distributions by letting  $(\mu_1 + \mu_2)(\sigma) = \mu_1(\sigma) + \mu_2(\sigma)$ .

We interpret a program  $c$  as  $\text{POVD}$  transformers. Given an expression  $e$ , we denote its interpretation with respect to machine state  $(\sigma, \rho)$  by  $\llbracket e \rrbracket_{(\sigma, \rho)}$ . For any program  $c$ , we denote the set of quantum variables modified by  $c$  as  $\text{qmod}(c)$  and the set of classical and quantum variables modified

Table 2. Denotational semantics of commands

$$\begin{aligned}
[[\text{skip}]]_{(\sigma, \rho)} &= (\sigma, \rho) \\
[[\text{abort}]]_{(\sigma, \rho)} &= \varepsilon \\
[[x := a]]_{(\sigma, \rho)} &= (\sigma[[a]]_{\sigma}/x, \rho) \\
[[c_0; c_1]]_{(\sigma, \rho)} &= [[c_1]]_{[[c_0]]_{(\sigma, \rho)}} \\
[[\text{if } b \text{ then } c_0 \text{ else } c_1 \text{ fi}]]_{(\sigma, \rho)} &= \begin{cases} [[c_0]]_{(\sigma, \rho)} & \text{if } [[b]]_{\sigma} = \text{true} \\ [[c_1]]_{(\sigma, \rho)} & \text{if } [[b]]_{\sigma} = \text{false} \end{cases} \\
[[\text{while } b \text{ do } c \text{ od}]]_{(\sigma, \rho)} &= \lim_{n \rightarrow \infty} [[(\text{if } b \text{ then } c \text{ fi})^n; \text{if } b \text{ then abort fi}]]_{(\sigma, \rho)} \\
[[q := |0\rangle]]_{(\sigma, \rho)} &= (\sigma, \rho') \\
&\quad \text{where } \rho' := |0\rangle_q \langle 0|_q |0\rangle_q \langle 0| + |0\rangle_q \langle 1|_q |1\rangle_q \langle 0| \\
[[U[\bar{q}]]]_{(\sigma, \rho)} &= (\sigma, U\rho U^\dagger) \\
[[x := M[\bar{q}]]]_{(\sigma, \rho)} &= \mu \\
&\quad \text{where } M = \{M_i\}_{i \in I} \text{ and} \\
&\quad \mu(\sigma') = \sum_i \{M_i \rho M_i^\dagger \mid \sigma[i/x] = \sigma'\} \\
[[c]]_{\mu} &= \sum_{\sigma \in [\mu]} [[c]]_{(\sigma, \mu(\sigma))}.
\end{aligned}$$

by  $c$  as  $\text{mod}(c)$ . Let  $V$  be a set of quantum variables. For any program  $c$  with  $\text{qmod}(c) \subseteq V$ , we define the denotational semantics of  $c$  with domain  $V$  as a mapping  $[[c]]_V : \mathbf{POVD}_V \rightarrow \mathbf{POVD}_V$ , displayed in Table 2. Sometimes, we omit the subscript  $V$  when it can be inferred from the context. Additionally, here we omit the denotational semantics of arithmetic and boolean expressions such as  $[[a]]_{\sigma}$  and  $[[b]]_{\sigma}$ , which is almost the same as in the classical setting because the quantum part plays no role in those expressions. A state evolves into a POVD after some quantum qubits are measured, with the measurement outcomes assigned to a classical variable. Two other quantum commands, initialisation of qubits and unitary operations, are deterministic and only affect the quantum part of a state. As usual, we define the semantics of a loop (**while**  $b$  **do**  $c$  **od**) as the limit of its lower approximations, where the  $n$ -th lower approximation of  $[[\text{while } b \text{ do } c \text{ od}]]_{(\sigma, \rho)}$  is  $[[(\text{if } b \text{ then } c \text{ fi})^n; \text{if } b \text{ then abort fi}]]_{(\sigma, \rho)}$ , where  $(\text{if } b \text{ then } c \text{ fi})$  is shorthand for  $(\text{if } b \text{ then } c \text{ else skip fi})$  and  $c^n$  is the command  $c$  iterated  $n$  times with  $c^0 \equiv \text{skip}$ . The limit exists because the sequence  $([[(\text{if } b \text{ then } c \text{ fi})^n; \text{if } b \text{ then abort fi}]]_{(\sigma, \rho)})_{n \in \mathbb{N}}$  is strictly increasing and bounded with respect to the Löwner order [4, Lemma 3.2].

We remark that the semantics  $[[c]]_{(\sigma, \rho)}$  of a command  $c$  in initial state  $(\sigma, \rho)$  is a POVD. The lifted semantics  $[[c]]_{\mu}$  of a command  $c$  in initial POVD  $\mu$  is also a POVD. Furthermore, the function  $[[c]]$  is linear in the sense that

$$[[c]]_{\sum_i p_i \cdot \mu_i} = \sum_i p_i \cdot [[c]]_{\mu_i}.$$

Additionally, the function  $[[c]]$  commutes with the restriction of distributions, i.e.,

$$[[c]]_{(\mu|_V)} = ([[c]]_{\mu})|_V,$$

for any  $V$  such that  $\text{qmod}(c) \subseteq V \subseteq \text{dom}(\mu)$ .

Similarly as in [26], we take advantage of the labelled Dirac notation throughout the paper with subscripts identifying the subsystem where a ket/bra/operator lies or operates. For example, the subscript in  $|a\rangle_{\bar{q}}$  indicates the Hilbert space  $\mathcal{H}_{\bar{q}}$  where the state  $|a\rangle$  lies. The notation  $|a\rangle|a\rangle$  and  $|a\rangle_{\bar{q}}|a\rangle_{\bar{q}}$  are the abbreviations of  $|a\rangle \otimes |a\rangle$  and  $|a\rangle_{\bar{q}} \otimes |a\rangle_{\bar{q}}$  respectively. We also use operators with subscripts like  $A_{\bar{q}}$  to identify the Hilbert space  $\mathcal{H}_{\bar{q}}$  where the operator  $A$  is applied.

Table 3. Syntax of assertion languages

(Pure expression)	$e$	$::=$	$n \mid x, y, \dots \mid f_m(e, \dots, e)$
(Pure formula)	$P$	$::=$	<b>true</b> $\mid$ <b>false</b> $\mid P_m(e, \dots, e) \mid P \wedge P \mid \neg P \mid \forall x. P(x)$
(Quantum expression)	$ s\rangle$	$::=$	$ a\rangle_{\bar{q}} \mid  s\rangle \otimes  s\rangle$
(State formula)	$F$	$::=$	$P \mid  s\rangle \mid F \odot F \mid F \wedge F$
(Distribution formula)	$D$	$::=$	$\oplus_{i \in I} p_i \cdot F_i \mid \oplus_{i \in I} F_i$

## 4 Proof System

In this section we present a proof system for local reasoning about probabilistic behaviour of quantum programs. We first define an assertion language, then propose a Hoare-style proof system, and finally prove the soundness of the system.

### 4.1 Assertion Language

We now introduce an assertion language for our programs, whose syntax is given in Table 3. Pure expressions are arithmetic expressions with integer constants, ranged over by  $e$ . Pure formulas, ranged over by  $P$ , include the boolean constants **true** and **false**, boolean predicates in the form  $P_m$  and any  $P$  connected by boolean operators such as negation, conjunction, and universal quantification. They are intended to capture properties of classical states. Quantum expressions, ranged over by  $|s\rangle$ , include quantum states of the form  $|a\rangle_{\bar{p}}$  and the tensor product  $|s\rangle \otimes |s\rangle$  which we abbreviate as  $|s\rangle |s\rangle$ . Here  $|a\rangle_{\bar{p}}$  can be any computational basis or their linear combinations in the Hilbert space  $\mathcal{H}_{\bar{p}}$ . State formulas, ranged over by  $F$ , are used to express properties on both classical and quantum states, which include the pure formula  $P$ , the quantum expression  $|s\rangle$  and any expression connected by boolean operator conjunction. In addition, we introduce a new connective  $\odot$  to express an assertion of two separable systems. Following [10], we use  $\text{free}(F)$  to denote the set of all free classical and quantum variables in  $F$ . For example,  $\text{free}(|a_1\rangle_{\bar{q}_1} \otimes |a_2\rangle_{\bar{q}_2}) = \bar{q}_1 \bar{q}_2$ . Moreover, we use  $\text{qfree}(F)$  to denote the set of all quantum variables in  $F$ . For the formula  $F_1 \odot F_2$  to be well defined, we impose the syntactical restriction that  $\text{qfree}(F_1) \cap \text{qfree}(F_2) = \emptyset$ . Intuitively, a quantum state satisfies  $F_1 \odot F_2$  if the state mentions two disjoint subsystems whose states satisfy  $F_1$  and  $F_2$  respectively. Distribution formulas consist of some state formulas  $F_i$  connected by the connective  $\oplus$  with the weights given by  $p_i$  satisfying  $\sum_{i \in I} p_i = 1$  as well as the non-probabilistic formula  $\oplus_{i \in I} F_i$ . If there is a collection of distribution formulas  $D_i = \oplus_j p_{ij} \cdot F_{ij}$  and a collection of probabilities  $p_i$  with  $\sum_{i \in I} p_i = 1$ , we sometimes write  $\oplus_i p_i \cdot D_i$  to mean the formula  $\oplus_{ij} p_i p_{ij} \cdot F_{ij}$ .

We use the notation  $\mu \models F$  to indicate that the state  $\mu$  satisfies the assertion  $F$ . The satisfaction relation  $\models$  is defined in Table 4. When writing  $(\sigma, \rho) \models F$ , we mean that  $(\sigma, \rho)$  is a machine state and  $\rho$  is its quantum part representing the status of the whole quantum system in a program under consideration. We use  $\llbracket P \rrbracket_\sigma$  to denote the evaluation of the pure predicate  $P$  with respect to the classical state  $\sigma$ .

A machine state  $(\sigma, \rho)$  satisfies the formula  $|s\rangle$  if the reduced density operator obtained by first normalising  $\rho$  and then restricting it to  $\text{free}(|s\rangle)$  becomes  $|s\rangle\langle s|$ , given that  $\text{qfree}(|s\rangle) \subseteq \text{dom}(\rho)$ . The state  $(\sigma, \rho)$  satisfies the formula  $F_1 \odot F_2$  if  $\text{qfree}(F_1)$  and  $\text{qfree}(F_2)$  are disjoint and the restrictions of  $\rho$  to  $\text{qfree}(F_1)$  and  $\text{qfree}(F_2)$  satisfy the two sub-formulas  $F_1$  and  $F_2$ , respectively. The assertion  $F$  holds on a distribution  $\mu$  when  $F$  holds on each state in the support of  $\mu$ . A distribution state  $\mu$  satisfies the distribution formula  $\oplus_{i \in I} p_i \cdot F_i$  if  $\mu$  is a linear combination of some  $\mu_i$  with weights  $p_i$  such that each  $\mu_i$  has the same size as  $\mu$  and satisfies  $F_i$  whenever  $p_i > 0$ . The formula  $\oplus_{i \in I} F_i$  is



Table 4. Semantics of assertions

$(\sigma, \rho) \models P$	if	$\llbracket P \rrbracket_\sigma = \mathbf{true}$
$(\sigma, \rho) \models  s\rangle$	if	$\frac{\rho}{\text{tr}(\rho)} _{\bar{q}} =  s\rangle\langle s $ where $\bar{q} = \text{free}( s\rangle) \subseteq \text{dom}(\rho)$
$(\sigma, \rho) \models F_1 \wedge F_2$	if	$(\sigma, \rho) \models F_1 \wedge (\sigma, \rho) \models F_2$
$(\sigma, \rho) \models F_1 \odot F_2$	if	$(\sigma, \rho _{\text{qfree}(F_1)}) \models F_1 \wedge (\sigma, \rho _{\text{qfree}(F_2)}) \models F_2$ $\wedge \text{qfree}(F_1) \cap \text{qfree}(F_2) = \emptyset$
$\mu \models F$	if	$\forall \sigma \in [\mu]. (\sigma, \mu(\sigma)) \models F$
$\mu \models \oplus_{i \in I} p_i \cdot F_i$	if	$\exists \mu_1 \dots \exists \mu_m.$ $[\mu = \sum_{i \in I} p_i \cdot \mu_i \wedge (\forall i, p_i \neq 0 \rightarrow \ \mu_i\  = \ \mu\  \wedge \mu_i \models F_i)]$ for $I = \{1, \dots, m\}$
$\mu \models \oplus_{i \in I} F_i$	if	$\exists p_1 \dots \exists p_m. [(\bigwedge_{i \in I} p_i \geq 0) \wedge \sum_i p_i = 1 \wedge \mu \models \oplus_{i \in I} p_i \cdot F_i]$ for $I = \{1, \dots, m\}$

a nondeterministic version of the distribution formulas in the form  $\oplus_{i \in I} p_i \cdot F_i$  without fixing the weights  $p_i$ , so the weights can be arbitrarily chosen as long as their sum is 1. For other assertions, the semantics should be self-explanatory.

Now we present several basic lemmas that form the foundation for our analysis of assertions within the given framework. These lemmas establish the soundness of our proof system and facilitate the manipulation of assertions.

**LEMMA 4.1.** *Let  $\mu$  be a distribution and  $F$  an assertion. For any  $V \subseteq \mathbf{QVar}$  such that  $\text{qfree}(F) \subseteq V \subseteq \text{dom}(\mu)$ , we have that  $\mu \models F \Leftrightarrow \mu|_V \models F$ .*

This lemma indicates that any assertion  $F$  is restrictive, enabling assertion verification by restricting attention to quantum variables referred to by  $F$ .

**LEMMA 4.2.** *The satisfaction relation is linear. Specifically, for any collection of probabilities  $p_i$  with  $0 < \sum_i p_i \leq 1$  and distributions  $\mu_i$  such that  $\mu_i \models F$  whenever  $p_i > 0$ , it holds that  $\sum_{i \in I} p_i \cdot \mu_i \models F$ .*

**LEMMA 4.3.** *Let  $\mu$  be a distribution and  $F$  an assertion. If  $\mu \models F$  and  $\text{free}(F) \cap \text{mod}(c) = \emptyset$ , then  $\llbracket c \rrbracket_\mu \models F$ .*

All the lemmas presented above can be proved by induction on the structure of assertions and programs. The difference between the two connectives  $\wedge$  and  $\odot$  is reflected by the following lemma, which is directly derivable from their semantic definitions.

**LEMMA 4.4.** (i)  $\mu \models F_1 \wedge F_2 \Leftrightarrow \mu \models F_1 \wedge \mu \models F_2$ ;  
(ii)  $\mu \models F_1 \odot F_2 \Leftrightarrow \mu \models F_1 \wedge \mu \models F_2 \wedge \text{qfree}(F_1) \cap \text{qfree}(F_2) = \emptyset$ .

The next lemma shows that the semantics for  $\odot$  aligns with the conventional understanding of separable conjunction.

**LEMMA 4.5.** *Let  $\sigma$  be a classical state,  $\rho$  a quantum state, and  $F_1, F_2$  two assertions. Then  $(\sigma, \rho) \models F_1 \odot F_2$  if and only if there exist  $V_1, V_2, \rho_1, \rho_2$  such that  $V_1 \cap V_2 = \emptyset$ ,  $\rho_1 \in \mathcal{D}^-(\mathcal{H}_{V_1})$ ,  $\rho_2 \in \mathcal{D}^-(\mathcal{H}_{V_2})$ , and*

$$\rho|_{V_1 \cup V_2} = \rho_1 \otimes \rho_2 \wedge (\sigma, \rho_1) \models F_1 \wedge (\sigma, \rho_2) \models F_2.$$

Detailed proofs of these lemmas are formalized in Coq. Finally, the following proposition is important for proving the soundness of the [While] rule presented in Subsection 4.2. Its proof proceeds by induction on the structure of  $D$ , and the details are presented in Appendix A.

**PROPOSITION 4.6.** *Let  $V \subseteq \mathbf{QVar}$ . For any assertion  $D$ , the set  $\llbracket D \rrbracket_V \triangleq \{\mu \mid \mu \models D \wedge \text{dom}(\mu) = V\}$  is a closed set.*

From the relation  $\models$ , we can derive a quantitative definition of satisfaction, where a distribution satisfies a predicate only to certain degree.

**Definition 4.7.** Let  $F$  be an assertion and  $p \in [0, 1]$  a real number. We say that the probability of a distribution  $\mu$  satisfying  $F$  is  $p$ , written by

$$\mathbb{P}_\mu(F) = p,$$

if there exist two distributions  $\mu_1$  and  $\mu_2$  such that  $\mu = p\mu_1 + (1 - p)\mu_2$ ,  $\mu_1 \models F$  and  $\mu_2 \models \neg F$ , and moreover,  $p$  is the maximum probability for this kind of decomposition of  $\mu$ .

In [10], assertions with disjunctions are used as the postconditions of measurement statements. However, this approach is awkward when reasoning about probabilities. Let us take a close look at the problem in Example 4.8.

**Example 4.8.** We revisit the program **addM** discussed in Example 1.1, which measures the variables  $q_0$  and  $q_1$ , both in the state  $|+\rangle\langle+|$ , and subsequently adds their results of measurements. Here  $M$  is a projective measurement  $\{|0\rangle\langle 0|, |1\rangle\langle 1|\}$ .

$$\begin{aligned} \mathbf{addM} &\triangleq q_0 := |0\rangle; H[q_0]; \\ &\quad q_1 := |0\rangle; H[q_1]; \\ &\quad v_0 := M[q_0]; \\ &\quad v_1 := M[q_1]; \\ &\quad v := v_0 + v_1 \end{aligned}$$

After the measurements on  $q_0$  and  $q_1$ , the classical variables  $v_0$  and  $v_1$  are assigned to either 0 or 1 with equal probability. By executing the last command, we assign the sum of  $v_0$  and  $v_1$  to  $v$ , and obtain the following POVD  $\mu$ .

$v_0 v_1 v$	$\mapsto$	$\frac{1}{4}$	$q_0 q_1$
000	$\mapsto$	$\frac{1}{4}$	$ 00\rangle\langle 00 $
011	$\mapsto$	$\frac{1}{4}$	$ 01\rangle\langle 01 $
101	$\mapsto$	$\frac{1}{4}$	$ 10\rangle\langle 10 $
112	$\mapsto$	$\frac{1}{4}$	$ 11\rangle\langle 11 $

The second column represents the four classical states while the last column shows the four quantum states. For example, we have  $\sigma_1(v_0 v_1 v) = 000$  and  $\rho_1 = |00\rangle\langle 00|$ . Let  $\mu_i = (\sigma_i, \rho_i)$  for  $1 \leq i \leq 4$ . Then  $\mu = \frac{1}{2}\mu_{14} + \frac{1}{2}\mu_{23}$ , where  $\mu_{14} = \frac{1}{2}\mu_1 + \frac{1}{2}\mu_4$  and  $\mu_{23} = \frac{1}{2}\mu_2 + \frac{1}{2}\mu_3$ . Since  $\mu_i \models (v = 1)$  for  $i = 2, 3$ , it follows that  $\mu_{23} \models (v = 1)$  by Lemma 4.2. On the other hand, we have  $\mu_{14} \not\models (v = 1)$ . Therefore, it follows that  $\mathbb{P}_\mu(v = 1) = \frac{1}{2}$ . That is, the probability for  $\mu$  to satisfy the assertion  $v = 1$  is  $\frac{1}{2}$ .

Alternatively, we can express the above property as a distribution formula. Let  $D = \frac{1}{2} \cdot (v = 1) \oplus \frac{1}{2} \cdot (v \neq 1)$ . We see that  $\mu \models D$  due to the fact that  $\mu = \frac{1}{2}\mu_{23} + \frac{1}{2}\mu_{14}$ ,  $\mu_{23} \models (v = 1)$  and  $\mu_{14} \models (v \neq 1)$ .

In [10], there is no distribution formula. The best we can do is to use a disjunctive assertion to describe the postcondition of the above program.

$$\begin{aligned} F &\triangleq \frac{1}{4} \cdot (v = 0 \wedge |00\rangle_{q_0 q_1}) \vee \frac{1}{4} \cdot (v = 1 \wedge |01\rangle_{q_0 q_1}) \vee \\ &\quad \frac{1}{4} \cdot (v = 1 \wedge |10\rangle_{q_0 q_1}) \vee \frac{1}{4} \cdot (v = 2 \wedge |11\rangle_{q_0 q_1}). \end{aligned}$$

This assertion does not take the mutually exclusive correlations between different branches into account. For example, it is too weak for us to prove that  $\mathbb{P}(v = 1) = \frac{1}{4} + \frac{1}{4} = \frac{1}{2}$ , as discussed in more details in [10]. From this example, we see that distribution formulas give us a more accurate way of describing the behaviour of measurement statements than disjunctive assertions.

## 4.2 Proof System

In this subsection, we present a series of inference rules for classical–quantum programs, which will be proved to be sound at the end of this subsection. As usual, we use the Hoare triple  $\{D_1\} c \{D_2\}$  to express the correctness of our programs, where  $c$  is a program and  $D_1, D_2$  are the assertions specified in Table 3.

Table 5. Inference rules for classical statements

$$\begin{array}{c}
\frac{}{\{D\} \text{ skip } \{D\}} [\text{Skip}] \quad \frac{}{\{D\} \text{ abort } \{\text{false}\}} [\text{Abort}] \\
\frac{}{\{D[a/x]\} x := a \{D\}} [\text{Assgn}] \quad \frac{}{\{D_0\} c_0 \{D_1\} \{D_1\} c_1 \{D_2\}} [\text{Seq}] \\
\frac{}{\{F_1 \wedge b\} c_1 \{F'_1\} \{F_2 \wedge \neg b\} c_2 \{F'_2\}} [\text{Cond}] \\
\frac{}{\{p(F_1 \wedge b) \oplus (1-p)(F_2 \wedge \neg b)\} \text{ if } b \text{ then } c_1 \text{ else } c_2 \text{ fi } \{pF'_1 \oplus (1-p)F'_2\}} [\text{Cond}] \\
\frac{}{\{\text{false}\} c \{D\}} [\text{Absurd}] \quad \frac{D_0 \Rightarrow D_1 \quad \{D_1\} c \{D_2\} \quad D_2 \Rightarrow D_3}{\{D_0\} c \{D_3\}} [\text{Conseq}] \\
\frac{}{D = (F_0 \wedge b) \oplus (F_1 \wedge \neg b) \quad \{D_0\} c \{D_3\} \quad \{F_0 \wedge b\} c \{D\}} [\text{While}] \\
\frac{}{\{D\} \text{ while } b \text{ do } c \text{ od } \{F_1 \wedge \neg b\} \{F_1\} c \{F'_1\} \{F_2\} c \{F'_2\}} [\text{Conj}] \\
\frac{}{\{F_1 \wedge F_2\} c \{F'_1 \wedge F'_2\} \quad \{F_1\} c \{F_2\} \quad \text{free}(F_3) \cap \text{mod}(c) = \emptyset} [\text{QFrame}] \\
\frac{}{\{F_1 \odot F_3\} c \{F_2 \odot F_3\} \quad \forall i \in I. \{D_i\} c \{D'_i\} \quad \sum_{i \in I} p_i = 1} [\text{Sum}] \\
\frac{}{\{\oplus_{i \in I} p_i \cdot D_i\} c \{\oplus_{i \in I} p_i \cdot D'_i\}} [\text{Sum}]
\end{array}$$

Table 5 lists the rules for classical statements, with most of them being standard and thus self-explanatory. The assertion  $D[a/x]$  is the same as  $D$  except that all the free occurrences of variable  $x$  in  $D$  are replaced by  $a$ . The assertion  $D_1 \Rightarrow D_2$  indicates that  $D_1$  logically implies  $D_2$ . When all the formulas involved in the [While] and [Cond] rules presented in Table 5 are pure formulas, they can deduce their corresponding classical counterparts, respectively. The quantum frame rule [QFrame] is introduced for local reasoning, which allows us to add assertion  $F_3$  to the pre/post-conditions of the local proof  $\{F_1\} c \{F_2\}$ . The condition  $\text{free}(F_3) \cap \text{mod}(c) = \emptyset$  indicates that all the free classical and quantum variables in  $F_3$  are not modified by program  $c$ . Note that when  $F_3$  is a pure assertion,  $\odot$  can be replaced by  $\wedge$ . The rule [Sum] allows us to reason about a probability distribution by considering each pure state individually.

Table 6 displays the inference rules for quantum statements. In rule [QInit], we see that the execution of the command  $\bar{q} := |0\rangle_{\bar{q}}$  sets the quantum system  $\bar{q}$  to  $|0\rangle$ , no matter what the initial state is. In rule [QUnit], for the postcondition  $F$  we have the precondition  $U_{\bar{q}}^\dagger F$ . Here,  $U_{\bar{q}}^\dagger$  distributes over those connectives of state formulas and eventually applies to quantum expressions. For example, if  $F = |v\rangle_{\bar{q}\bar{q}'} \wedge P$ , then  $U_{\bar{q}}^\dagger F = U_{\bar{q}}^\dagger(|v\rangle_{\bar{q}\bar{q}'} \wedge P)$ . In rule [QMeas], the combined state of the variables  $\bar{q}\bar{q}'$  is specified because there may be an entanglement between the subsystems for  $\bar{q}$  and  $\bar{q}'$ . In that rule, we write  $P_i[i/x]$  for the assertion obtained from  $P_i$  by replacing the variable  $x$  with value

Table 6. Inference rules for quantum statements

$$\begin{array}{c}
\frac{}{\{\mathbf{true}\} \bar{q} := |0\rangle \ \{ |0\rangle_{\bar{q}} \}} [\text{QInit}] \quad \frac{}{\{U_{\bar{q}}^\dagger F\} U[\bar{q}] \ \{F\}} [\text{QUnit}] \\
\\
\frac{M = \{M_i\}_{i \in I} \quad p_i = \|M_{i\bar{q}} |v\rangle\|^2}{\{\wedge_i (P_i[i/x] \wedge |v\rangle_{\bar{q}\bar{q}'}))\} x := M[\bar{q}] \ \{\oplus_i p_i \cdot (P_i \wedge (M_{i\bar{q}} |v\rangle_{\bar{q}\bar{q}'} / \sqrt{p_i}))\}} [\text{QMeas}]
\end{array}$$

Table 7. Inference rules for entailment reasoning

$$\begin{array}{c}
\frac{}{F \vdash \mathbf{true}} [\text{PT}] \quad \frac{}{F \odot \mathbf{true} \dashv\vdash F} [\text{OdotE}] \\
\\
\frac{}{F_1 \odot F_2 \dashv\vdash F_2 \odot F_1} [\text{OdotC}] \quad \frac{}{F_1 \odot (F_2 \odot F_3) \dashv\vdash (F_1 \odot F_2) \odot F_3} [\text{OdotA}] \\
\\
\frac{}{P_1 \odot P_2 \dashv\vdash P_1 \wedge P_2} [\text{OdotO}] \quad \frac{}{P \odot F \dashv\vdash P \wedge F} [\text{OdotOP}] \\
\\
\frac{}{P \wedge (F_1 \odot F_2) \dashv\vdash (P \wedge F_1) \odot F_2} [\text{OdotOA}] \\
\\
\frac{}{F_1 \odot (F_2 \wedge F_3) \dashv\vdash (F_1 \odot F_2) \wedge (F_1 \odot F_3)} [\text{OdotOC}] \\
\\
\frac{}{|u\rangle_{\bar{p}} |v\rangle_{\bar{q}} \dashv\vdash |v\rangle_{\bar{q}} |u\rangle_{\bar{p}}} [\text{ReArr}] \quad \frac{}{|u\rangle_{\bar{p}} |v\rangle_{\bar{q}} \dashv\vdash |u \otimes v\rangle_{\bar{p}\bar{q}}} [\text{Separ}] \\
\\
\frac{}{|u\rangle_{\bar{p}} |v\rangle_{\bar{q}} \dashv\vdash |u\rangle_{\bar{p}} \odot |v\rangle_{\bar{q}}} [\text{OdotT}] \\
\\
\frac{}{p_0 \cdot F \oplus p_1 \cdot F \oplus p_2 \cdot F' \dashv\vdash (p_0 + p_1) \cdot F \oplus p_2 \cdot F'} [\text{OMerg}] \\
\\
\frac{}{\oplus_{i \in I} p_i \cdot F_i \vdash \oplus_{i \in I} F_i} [\text{Oplus}] \quad \frac{\forall i \in I, F_i \vdash F'_i}{\oplus_{i \in I} p_i \cdot F_i \vdash \oplus_{i \in I} p_i \cdot F'_i} [\text{OCon}]
\end{array}$$

*i*. The postcondition is a distribution formula, with each assertion  $P_i \wedge (M_{i\bar{q}} |v\rangle_{\bar{q}\bar{q}'} / \sqrt{p_i})$  assigned probability  $p_i$ , i.e., the probability of obtaining outcome *i* after the measurement *M*.

Table 7 presents several rules for entailment reasoning about quantum predicates. The notation  $D_1 \vdash D_2$  says that  $D_1$  proves  $D_2$ . Intuitively, it means that any state satisfying  $D_1$  also satisfies  $D_2$ . We write  $D_1 \dashv\vdash D_2$  if the other direction also holds.

The connective  $\odot$  is commutative and associative, according to the rules [OdotC] and [OdotA]. If one or two assertions are pure, the rules [OdotO] and [OdotOP] replace  $\odot$  with  $\wedge$ . The rule [OdotOA] replaces  $P \wedge (F_1 \odot F_2)$  with  $(P \wedge F_1) \odot F_2$  and vice versa. The rule [OdotOC] assists us to distribute  $\odot$  into conjunctive assertions. The rule [ReArr] allows us to rearrange quantum expressions while [Separ] allows us to split/join the quantum expressions, given that  $\bar{p}$  and  $\bar{q}$  are not entangled with each other. The rules [ReArr] and [Separ] can be obtained naturally via the properties of tensor products. The rule [OdotT] replaces the  $\odot$  connective with  $\otimes$  when both assertions are state expressions. The rule [OMerg] allows us to merge the probabilities of two branches in a distribution formula if the two branches are the same. The rule [Oplus] is easy to be understood as  $\oplus_{i \in I} F_i$  is essentially a relaxed form of  $\oplus_{i \in I} p_i \cdot F_i$ . The rule [OCon] says that if each  $F_i$  entails  $F'_i$ , then the entailment relation is preserved between the combinations  $\oplus_{i \in I} p_i \cdot F_i$  and  $\oplus_{i \in I} p_i \cdot F'_i$ . We use the notation  $\vdash \{D_1\} c \{D_2\}$  to mean that the Hoare triple  $\{D_1\} c \{D_2\}$  is provable by applying the rules in Tables 5–7.

*Example 4.9.* Suppose there are two separable systems  $\bar{q}$  and  $\bar{q}'$ . They satisfy the precondition  $\frac{1}{2} |u_1\rangle_{\bar{q}} |v_1\rangle_{\bar{q}'} \oplus \frac{1}{2} |u_2\rangle_{\bar{q}} |v_2\rangle_{\bar{q}'}$ . After applying the operator  $U[\bar{q}]$ , they satisfy the postcondition  $\frac{1}{2} (U_{\bar{q}} |u_1\rangle_{\bar{q}}) |v_1\rangle_{\bar{q}'} \oplus \frac{1}{2} (U_{\bar{q}} |u_2\rangle_{\bar{q}}) |v_2\rangle_{\bar{q}'}$ . In other words, the following Hoare triple holds.

$$\begin{aligned} & \vdash \{ \tfrac{1}{2} |u_1\rangle_{\bar{q}} |v_1\rangle_{\bar{q}'} \oplus \tfrac{1}{2} |u_2\rangle_{\bar{q}} |v_2\rangle_{\bar{q}'} \} \\ & \quad U[\bar{q}] \\ & \{ \tfrac{1}{2} (U_{\bar{q}} |u_1\rangle_{\bar{q}}) |v_1\rangle_{\bar{q}'} \oplus \tfrac{1}{2} (U_{\bar{q}} |u_2\rangle_{\bar{q}}) |v_2\rangle_{\bar{q}'} \} \end{aligned} \quad (1)$$

This Hoare triple can be proved as follows. Firstly, we apply the rule [QUnit] to obtain

$$\vdash \{ |u_1\rangle_{\bar{q}} \} U[\bar{q}] \{ U_{\bar{q}} |u_1\rangle_{\bar{q}} \}. \quad (2)$$

Then we use the rules [QFrame] and [OdotT] to get

$$\vdash \{ |u_1\rangle_{\bar{q}} |v_1\rangle_{\bar{q}'} \} U[\bar{q}] \{ (U_{\bar{q}} |u_1\rangle_{\bar{q}}) |v_1\rangle_{\bar{q}'} \}. \quad (3)$$

Similarly, we have

$$\vdash \{ |u_2\rangle_{\bar{q}} |v_2\rangle_{\bar{q}'} \} U[\bar{q}] \{ (U_{\bar{q}} |u_2\rangle_{\bar{q}}) |v_2\rangle_{\bar{q}'} \}. \quad (4)$$

Combining (3) with (4) by rule [Sum], we obtain the Hoare triple in (1).

*Example 4.10.* Let us consider the program **addM** and the distribution formula  $D = \frac{1}{2} \cdot (v = 1) \oplus \frac{1}{2} \cdot (v \neq 1)$  discussed in Example 4.8. Now we would like to formally prove that

$$\{\text{true}\} \text{addM} \{D\}.$$

The proof outline is given in Table 8. Following [10], we use the following notations to highlight the application of the frame rule [QFrame]:

$$\begin{aligned} \Longleftrightarrow \{F_1\} c \{F_2\} \quad \text{or equivalently} \quad & \begin{array}{c} \Longrightarrow \{F_1\} \\ c \\ \Longleftarrow \{F_2\} \end{array} \end{aligned}$$

Both notations indicate that  $\{F_1\} c \{F_2\}$  is a local proof for  $c$  and is useful for long proofs. The frame assertion  $F_3$  can be deduced from the assertions before  $F_1$  or after  $F_2$ .

Our inference system is sound in the following sense: any Hoare triple in the form  $\{D_1\} c \{D_2\}$  derived from the inference system is valid, denoted by  $\models \{D_1\} c \{D_2\}$ , meaning that for any state  $\mu$  we have  $\mu \models D_1$  implies  $\llbracket c \rrbracket_\mu \models D_2$ .

**THEOREM 4.1 (SOUNDNESS).** *For any trace-preserving program  $c$ , if  $\vdash \{D_1\} c \{D_2\}$  then  $\models \{D_1\} c \{D_2\}$ .*

**PROOF.** The proof is carried out by rule induction. We focus on some difficult rules, [Cond], [While] and [QFrame], while omitting the others. Readers interested in the full proof details are invited to consult our Coq formalization.

Consider the rule [Cond] first.

$$\frac{\{F_0 \wedge b\} c_0 \{F'_0\} \quad \{F_1 \wedge \neg b\} c_1 \{F'_1\}}{\{p(F_0 \wedge b) \oplus (1-p)(F_1 \wedge \neg b)\} \text{ if } b \text{ then } c_0 \text{ else } c_1 \text{ fi } \{pF'_0 \oplus (1-p)F'_1\}}$$

Assume that  $\mu \models p(F_0 \wedge b) \oplus (1-p)(F_1 \wedge \neg b)$ . Then there exist some states  $\mu_0$  and  $\mu_1$  such that  $\mu = p\mu_0 + (1-p)\mu_1$ ,  $\|\mu_0\| = \|\mu_1\| = \|\mu\|$ ,  $\mu_0 \models F_0 \wedge b$  and  $\mu_1 \models F_1 \wedge \neg b$ . By induction, the premises

Table 8. Proof outline for the **addM** program

```

{ true }
{ true  $\odot$  true }    by rule [OdotE]
 $\Longleftrightarrow$  { true }    $q_0 := |0\rangle$ ; {  $|0\rangle_{q_0}$  }    by rule [QInit]
 $\Longleftrightarrow$  {  $|0\rangle_{q_0}$  }    $H[q_0]$ ; {  $|+\rangle_{q_0}$  }    by rule [QUnit]
 $\Longleftrightarrow$  { true }    $q_1 := |0\rangle$ ; {  $|0\rangle_{q_1}$  }    by rule [QInit]
 $\Longleftrightarrow$  {  $|0\rangle_{q_1}$  }    $H[q_1]$ ; {  $|+\rangle_{q_1}$  }    by rule [QUnit]
{  $|+\rangle_{q_0 q_1} \wedge (v_0 = 0)[0/v_0] \wedge (v_0 = 1)[1/v_0]$  }    by rule [QFrame] and [Conseq]
 $v_0 := M[q_0]$ ;
{  $\frac{1}{2} \cdot (|0+\rangle_{q_0 q_1} \wedge v_0 = 0) \oplus \frac{1}{2} \cdot (|1+\rangle_{q_0 q_1} \wedge v_0 = 1)$  }    by rule [QMeas]
{  $\frac{1}{2} \cdot (|0+\rangle_{q_0 q_1} \wedge v_0 = 0 \wedge (v_1 = 0)[0/v_1] \wedge (v_1 = 1)[1/v_1])$ 
 $\oplus \frac{1}{2} \cdot (|1+\rangle_{q_0 q_1} \wedge v_0 = 1 \wedge (v_1 = 0)[0/v_1] \wedge (v_1 = 1)[1/v_1])$  }    by rule [Conseq]
 $v_1 := M[q_1]$ ;
{  $\frac{1}{4} \cdot (|00\rangle_{q_0 q_1} \wedge v_0 = 0 \wedge v_1 = 0)$ 
 $\oplus \frac{1}{4} \cdot (|01\rangle_{q_0 q_1} \wedge v_0 = 0 \wedge v_1 = 1)$ 
 $\oplus \frac{1}{4} \cdot (|10\rangle_{q_0 q_1} \wedge v_0 = 1 \wedge v_1 = 0)$ 
 $\oplus \frac{1}{4} \cdot (|11\rangle_{q_0 q_1} \wedge v_0 = 1 \wedge v_1 = 1)$  }    by rules [QMeas] and [Sum]
{  $\frac{1}{4} \cdot (|00\rangle_{q_0 q_1} \wedge v_0 = 0 \wedge v_1 = 0 \wedge (v = 0)[(v_0 + v_1)/v])$ 
 $\oplus \frac{1}{4} \cdot (|01\rangle_{q_0 q_1} \wedge v_0 = 0 \wedge v_1 = 1 \wedge (v = 1)[(v_0 + v_1)/v])$ 
 $\oplus \frac{1}{4} \cdot (|10\rangle_{q_0 q_1} \wedge v_0 = 1 \wedge v_1 = 0 \wedge (v = 1)[(v_0 + v_1)/v])$ 
 $\oplus \frac{1}{4} \cdot (|11\rangle_{q_0 q_1} \wedge v_0 = 1 \wedge v_1 = 1 \wedge (v = 2)[(v_0 + v_1)/v])$  }    by rule [Conseq]
 $v := v_0 + v_1$ 
{  $\frac{1}{4} \cdot (|00\rangle_{q_0 q_1} \wedge v_0 = 0 \wedge v_1 = 0 \wedge v = 0)$ 
 $\oplus \frac{1}{4} \cdot (|01\rangle_{q_0 q_1} \wedge v_0 = 0 \wedge v_1 = 1 \wedge v = 1)$ 
 $\oplus \frac{1}{4} \cdot (|10\rangle_{q_0 q_1} \wedge v_0 = 1 \wedge v_1 = 0 \wedge v = 1)$ 
 $\oplus \frac{1}{4} \cdot (|11\rangle_{q_0 q_1} \wedge v_0 = 1 \wedge v_1 = 1 \wedge v = 2)$  }    by rule [Assgn] and [Sum]
{  $\frac{1}{4} \cdot (v = 0) \oplus \frac{1}{4} \cdot (v = 1) \oplus \frac{1}{4} \cdot (v = 1) \oplus \frac{1}{4} \cdot (v = 2)$  }    by rule [OCon]
{  $\frac{1}{4} \cdot (v \neq 1) \oplus \frac{1}{4} \cdot (v = 1) \oplus \frac{1}{4} \cdot (v = 1) \oplus \frac{1}{4} \cdot (v \neq 1)$  }    by rule [OCon]
{  $\frac{1}{2} \cdot (v = 1) \oplus \frac{1}{2} \cdot (v \neq 1)$  }    by rule [OMerg]

```

$\{F_0 \wedge b\} c_0 \{F'_0\}$  and  $\{F_1 \wedge \neg b\} c_1 \{F'_1\}$  are valid. It follows that  $\llbracket c_0 \rrbracket_{\mu_0} \models F'_0$  and  $\llbracket c_1 \rrbracket_{\mu_1} \models F'_1$ . Since the semantic function  $\llbracket \cdot \rrbracket$  is linear, we then have

$$\begin{aligned}
& \llbracket \text{if } b \text{ then } c_0 \text{ else } c_1 \text{ fi} \rrbracket_{\mu} \\
&= p \llbracket \text{if } b \text{ then } c_0 \text{ else } c_1 \text{ fi} \rrbracket_{\mu_0} + (1-p) \llbracket \text{if } b \text{ then } c_0 \text{ else } c_1 \text{ fi} \rrbracket_{\mu_1} \\
&= p \llbracket c_0 \rrbracket_{\mu_0} + (1-p) \llbracket c_1 \rrbracket_{\mu_1}.
\end{aligned}$$

Since we consider programs that are trace-preserving, it is easy to see that  $p \llbracket c_0 \rrbracket_{\mu_0} + (1-p) \llbracket c_1 \rrbracket_{\mu_1} \models pF'_0 \oplus (1-p)F'_1$ .

Now let us consider the rule [While].

$$\frac{D = (F_0 \wedge b) \oplus (F_1 \wedge \neg b), \quad \{F_0 \wedge b\} c \{D\}}{\{D\} \text{ while } b \text{ do } c \text{ od } \{F_1 \wedge \neg b\}}$$

Suppose  $\mu \models D$  for some state  $\mu$ . Then there exist states  $\mu_0, \mu_1$  and a probability  $p$  such that  $\mu = p\mu_0 + (1-p)\mu_1$ ,  $\|\mu_0\| = \|\mu_1\| = \|\mu\|$ ,  $\mu_0 \models F_0 \wedge b$  and  $\mu_1 \models F_1 \wedge \neg b$ . By induction, the premise  $\{F_0 \wedge b\} c \{D\}$  is valid. It follows that  $\llbracket c \rrbracket_{\mu_0} \models D$ .

For any  $n \geq 0$ , we write **while** <sup>$n$</sup>  for the  $n$ -th iteration of the while loop, i.e.

$$(\text{if } b \text{ then } c \text{ fi})^n; \text{if } b \text{ then abort fi.}$$

We claim that if  $\mu \models D$  then  $\llbracket \text{while}^n \rrbracket_\mu \models F_1 \wedge \neg b$  for any  $n \geq 0$ . This can be proved by induction on  $n$ .

- $n = 0$ . In this case we have that

$$\begin{aligned} \llbracket \text{while}^0 \rrbracket_\mu &= \llbracket \text{if } b \text{ then abort fi} \rrbracket_\mu \\ &= p \llbracket \text{if } b \text{ then abort fi} \rrbracket_{\mu_0} + (1-p) \llbracket \text{if } b \text{ then abort fi} \rrbracket_{\mu_1} \\ &= p\varepsilon + (1-p)\mu_1 \\ &= (1-p)\mu_1. \end{aligned}$$

Since  $\mu_1 \models F_1 \wedge \neg b$ , it implies that  $(1-p)\mu_1 \models F_1 \wedge \neg b$  by Lemma 4.2.

- $n = k + 1$ . We infer that

$$\begin{aligned} \llbracket \text{while}^{k+1} \rrbracket_\mu &= \llbracket \text{if } b \text{ then } c \text{ fi; while}^k \rrbracket_\mu \\ &= p \llbracket \text{if } b \text{ then } c \text{ fi; while}^k \rrbracket_{\mu_0} + (1-p) \llbracket \text{if } b \text{ then } c \text{ fi; while}^k \rrbracket_{\mu_1} \\ &= p \llbracket \text{while}^k \rrbracket_{\llbracket c \rrbracket_{\mu_0}} + (1-p) \llbracket \text{while}^k \rrbracket_{\mu_1} \\ &= p \llbracket \text{while}^k \rrbracket_{\llbracket c \rrbracket_{\mu_0}} + (1-p)\mu_1. \end{aligned}$$

Note that  $\llbracket c \rrbracket_{\mu_0} \models D$ . Therefore, it follows from induction hypothesis that  $\llbracket \text{while}^k \rrbracket_{\llbracket c \rrbracket_{\mu_0}} \models F_1 \wedge \neg b$ . Since  $\mu_1 \models F_1 \wedge \neg b$ , we immediately have that  $\llbracket \text{while}^{k+1} \rrbracket_\mu \models F_1 \wedge \neg b$  by Lemma 4.2.

Thus we have completed the proof of the claim. Note that if the domain of  $\mu$  is  $V \subseteq \mathbf{QVar}$ , then the domain of  $\llbracket \text{while}^n \rrbracket_\mu$  remains  $V$  for all  $n \geq 0$ . Consequently, by Proposition 4.6, we have

$$\llbracket \text{while } b \text{ do } c \text{ od} \rrbracket_\mu = \lim_{n \rightarrow \infty} \llbracket \text{while}^n \rrbracket_\mu \models F_1 \wedge \neg b.$$

Finally, we consider the rule [QFrame]:

$$\frac{\{F_1\} c \{F_2\} \quad \text{free}(F_3) \cap \text{mod}(c) = \emptyset}{\{F_1 \odot F_3\} c \{F_2 \odot F_3\}} [\text{QFrame}].$$

Consider a distribution  $\mu$  such that  $\mu \models F_1 \odot F_3$ . By Lemma 4.4, we can infer that  $\mu \models F_1$  and  $\mu \models F_3$ . Define  $\mu' = \llbracket c \rrbracket_\mu$ . Suppose that  $\text{qfree}(F_2) \cap \text{qfree}(F_3) = \emptyset$ . Our objective is to demonstrate that:

$$\mu' \models F_2 \quad \text{and} \quad \mu' \models F_3.$$

By induction, the premise  $\{F_1\} c \{F_2\}$  is valid. We immediately obtain  $\mu' \models F_2$ . Since  $\text{free}(F_3) \cap \text{mod}(c) = \emptyset$ , it follows from Lemma 4.3 that  $\mu' \models F_3$ . Consequently, by invoking Lemma 4.4, we conclude that  $\mu' \models F_2 \odot F_3$ .  $\square$

## 5 Case Analysis

Here, we apply the proof system to verify the functional correctness of two non-trivial algorithms: the HHL and Shor's algorithms in Subsections 5.1 and 5.2, respectively.

### 5.1 HHL Algorithm

The HHL algorithm [7] aims to obtain a vector  $x$  such that  $Ax = b$ , where  $A$  is a given Hermitian operator and  $b$  is a given vector. Suppose that  $A$  has the spectral decomposition  $A = \sum_j \lambda_j |j\rangle \langle j|$ , where each  $\lambda_j$  is an eigenvalue and  $|j\rangle$  is the corresponding eigenvector of  $A$ . On the basis  $\{|j\rangle\}_{j \in J}$ , we have  $A^{-1} = \sum_j \lambda_j^{-1} |j\rangle \langle j|$  and  $|b\rangle = \sum_j b_j |j\rangle$ . Then the vector  $x$  can be expressed as

$|x\rangle = A^{-1} |b\rangle = \sum_j \lambda_j^{-1} b_j |j\rangle$ . Here we require that  $|j\rangle$ ,  $|b\rangle$  and  $|x\rangle$  are all normalized vectors. Hence, we have

$$\sum_j |\lambda_j^{-1} b_j|^2 = 1. \quad (5)$$

A quantum program implementing the HHL algorithm is presented in Table 9. The  $n$ -qubit subsystem  $\bar{p}$  is used as a control system in the phase estimation step with  $N = 2^n$ . The  $m$ -qubit subsystem  $\bar{q}$  stores the vector  $|b\rangle = \sum_i b_i |i\rangle$ . The one-qubit subsystem  $r$  is used to control the while loop. The measurement  $M = \{M_0, M_1\}$  in the loop is the simplest two-value measurement:  $M_0 = |0\rangle_r \langle 0|$  and  $M_1 = |1\rangle_r \langle 1|$ . The results of measurements will be assigned to the classical variable  $v$ , which is initialized to be 0. If the value of  $v$  is 0, then the while loop is repeated until it is 1. The unitary operator  $U_b$  is assumed to map  $|0\rangle^{\otimes m}$  to  $|b\rangle$ . The controlled unitary operator  $U_f$  has a control system  $\bar{p}$  and a target system  $\bar{q}$ , that is,

$$U_f = \sum_{\tau=0}^{N-1} |\tau\rangle_p \langle \tau| \otimes U^\tau,$$

where  $U = e^{iAt}$ . Equivalently, we have  $U |j\rangle = e^{i\lambda_j t} |j\rangle$ . We denote  $\phi_j = \frac{\lambda_j t}{2\pi}$  and  $\tilde{\phi}_j = \phi_j \cdot N$ , then we have  $U |j\rangle = e^{2\pi i \phi_j} |j\rangle$ . Following [26], to make the algorithm exact, we further assume that there exists  $t \in \mathbb{R}$ , for all  $j \in J$ ,  $\tilde{\phi}_j = \frac{\lambda_j t \cdot N}{2\pi} \in \{1, 2, \dots, N-1\}$ . A given controlled unitary operator  $U_c$  has control system  $\bar{p}$  and target system  $r$ , more precisely,

$$U_c |0\rangle_{\bar{p}} |0\rangle_r = |0\rangle_{\bar{p}} |0\rangle_r, \quad U_c |j\rangle_{\bar{p}} |0\rangle_r = |j\rangle_{\bar{p}} (\sqrt{1 - \frac{C^2}{j^2}} |0\rangle + \frac{C}{j} |1\rangle)_r,$$

where  $1 \leq j \leq N-1$  and  $C$  is a given non-zero parameter such that  $|\frac{C}{j}| \leq 1$  for any  $0 \leq j \leq N-1$ . The symbol  $H^{\otimes n}$  represents  $n$  Hadamard gates applied to the variables  $\bar{p}$ ; QFT and  $\text{QFT}^{-1}$  are the quantum Fourier transform and the inverse quantum Fourier transform acting on the variable  $\bar{p}$ .

The correctness of the HHL algorithm can be specified by the Hoare triple:

$$\{ \text{true} \} \text{HHL} \{ |x\rangle_{\bar{q}} \}.$$

To prove the correctness, we first consider the loop body of the program in Table 9 and give its proof outline in Table 19 of Appendix B. Now let  $D \triangleq (v = 0) \oplus ((|0\rangle_{\bar{p}}^{\otimes n} |x\rangle_{\bar{q}} |1\rangle_r) \wedge (v = 1))$ , and  $S$  the body of the while loop of the HHL algorithm. From the reasoning in Table 19, we see that

$$\{ v = 0 \} S \{ D \}.$$

So  $D$  is an invariant of the while loop of the HHL algorithm. Then by rule [While] we obtain that

$$\{ D \} \text{while } (v = 0) \text{ do } S \text{ od } \{ |0\rangle_{\bar{p}}^{\otimes n} |x\rangle_{\bar{q}} |1\rangle_r \wedge (v = 1) \}.$$

Finally, we can establish the correctness of the HHL algorithm as given in Table 10, where we highlight the invariant of the while loop in red.

## 5.2 Shor's Algorithm

Shor's algorithm relies on the order-finding algorithm [17]. So we first verify the correctness of the latter. Given two co-prime positive integers  $x$  and  $N$ , the smallest positive integer  $r$  that satisfies the equation  $x^r = 1 \pmod{N}$  is called the order of  $x$  modulo  $N$ , denoted by  $\text{ord}(x, N)$ . The problem of order-finding is to find the order  $r$  defined above, which is solved by the program presented in Table 11. Let  $L \triangleq \lceil \log(N) \rceil$ ,  $\epsilon \in (0, 1)$  and  $t \triangleq 2L + 1 + \lceil \log(2 + 1/2\epsilon) \rceil$ . The order-finding algorithm can successfully obtain the order of  $x$  with probability at least  $(1 - \epsilon)/2 \log(N)$ , by using  $O(L^3)$  operations as discussed in [13].



Table 9. A quantum program for the HHL algorithm

**HHL**  $\triangleq$

```

1 :  $v := 0$ 
2 : while  $v = 0$  do
3 :    $\bar{p} := |0\rangle^{\otimes n}$ ;
4 :    $\bar{q} := |0\rangle^{\otimes m}$ ;
5 :    $r := |0\rangle$ ;
6 :    $U_b[\bar{q}]$ ;
7 :    $H^{\otimes n}[\bar{p}]$ ;
8 :    $U_f[\bar{p}\bar{q}]$ ;
9 :    $\text{QFT}^{-1}[\bar{p}]$ ;
10 :   $U_c[\bar{p}r]$ ;
11 :   $\text{QFT}[\bar{p}]$ ;
12 :   $U_f^\dagger[\bar{p}\bar{q}]$ ;
13 :   $H^{\otimes n}[\bar{p}]$ ;
14 :   $v := M[r]$  od

```

The variables in  $\bar{q}$  correspond to a  $t$ -qubit subsystem while  $\bar{p}$  represents an  $L$ -qubit subsystem. We introduce the variable  $z$  to store the order computed by the program **OF**, and initialize it to 1. The unitary operator  $U_+$  maps  $|0\rangle$  to  $|1\rangle$ , that is  $U_+ |0\rangle = |1\rangle$ . The notation  $H^{\otimes t}$  means  $t$  Hadamard gates applied to the system  $\bar{q}$  and  $\text{QFT}^{-1}$  is the inverse quantum Fourier transform. The function  $f(x)$  stands for the continued fraction algorithm which returns the minimal denomination  $n$  of all convergents  $m/n$  of the continued fraction for  $x$  with  $|m/n - x| < 1/(2n^2)$  [6]. The unitary operator CU is the controlled- $U$ , with  $\bar{q}$  being the control system and  $\bar{p}$  the target system, that is  $\text{CU } |i\rangle_{\bar{q}} |j\rangle_{\bar{p}} = |i\rangle_{\bar{q}} U^i |j\rangle_{\bar{p}}$ , where for each  $0 \leq y \leq 2^L$ ,

$$U |y\rangle = \begin{cases} |xy \bmod N\rangle & \text{if } y \leq N \\ |y\rangle & \text{otherwise.} \end{cases}$$

Note that the states defined by

$$|u_s\rangle \triangleq \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-2\pi i s k / r} |x^k \bmod N\rangle$$

for integer  $0 \leq s \leq r-1$  are eigenstates of  $U$  and  $\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |u_s\rangle = 1$ . Similarly, we assume that for all  $0 \leq s \leq r-1$ ,  $(\frac{s}{r} \cdot (2^t)) \in \{0, 1, \dots, 2^t - 1\}$ .

The variable  $b$  stores the value of  $(x^z \bmod N)$ , and the operator  $(a \bmod b)$  computes the modulo of  $a$  divided by  $b$ . If the value of  $b$  is not equal to 1, which means that the value of  $z$  computed by **OF** is not equal to the actual order of  $x$  modulo  $N$ , then the program will repeat the body of the while loop until  $b = 1$ . The while loop in the **OF** program exhibits probabilistic behaviour due to a measurement in the loop body.

Table 10. Proof outline of the HHL algorithm

```

{ true }
{ (v = 0)[0/v] }    by rule [Conseq]
v := 0;
{ v = 0 }    by rule [Assgn]
{ (v = 0)  $\oplus$  ( $|0\rangle_{\bar{p}}^{\otimes n} |x\rangle_{\bar{q}} |1\rangle_r \wedge (v = 1)$ ) }    by rule [Oplus]
while v = 0 do
  { v = 0 }
  S
  { (v = 0)  $\oplus$  ( $|0\rangle_{\bar{p}}^{\otimes n} |x\rangle_{\bar{q}} |1\rangle_r \wedge (v = 1)$ ) }
od
{  $|0\rangle_{\bar{p}}^{\otimes n} |x\rangle_{\bar{q}} |1\rangle_r \wedge (v = 1)$  }    by rule [While]
{  $|0\rangle_{\bar{p}}^{\otimes n} |x\rangle_{\bar{q}} |1\rangle_r$  }    by rule [Conseq]
{  $|0\rangle_{\bar{p}}^{\otimes n} \odot |x\rangle_{\bar{q}} \odot |1\rangle_r$  }    by rule [OdotT]
{ true  $\odot |x\rangle_{\bar{q}} \odot$  true }    by rule [PT]
{  $|x\rangle_{\bar{q}}$  }    by rule [OdotE]

```

Table 11. A quantum program for the order-finding algorithm

```

OF(x, N)  $\equiv$ 
1 : z := 1;
2 : b := xz (mod N);
3 : while (b  $\neq$  1) do
4 :    $\bar{q} := |0\rangle^{\otimes t}$ ;
5 :    $\bar{p} = |0\rangle^{\otimes L}$ ;
6 :    $H^{\otimes t}[\bar{q}]$ ;
7 :    $U_+[\bar{p}]$ ;
8 :    $CU[\bar{q}\bar{p}]$ ;
9 :    $QFT^{-1}[\bar{q}]$ ;
10 :  z' := M[\bar{q}];
11 :  z := f( $\frac{z'}{2^t}$ );
12 :  b := xz (mod N) od

```

The correctness of the order-finding algorithm can be specified as

$$\{ 2 \leq x \leq N - 1 \wedge \gcd(x, N) = 1 \} \text{ OF } \{ 2 \leq x \leq N - 1 \wedge z = r \} .$$

Table 12. Proof outline of the **OF** program
$$\begin{aligned}
& \{ 2 \leq x \leq N \wedge \gcd(x, N) = 1 \} \\
& \implies \{(z = 1)[1/z]\} \\
& z := 1; \\
& \longleftarrow \{z = 1\} \quad \text{by rule [Assgn]} \\
& \implies \{(z = 1 \wedge b = x^1 \bmod N)[(x^z \bmod N)/b]\} \\
& b := x^z \bmod N; \\
& \longleftarrow \{z = 1 \wedge b = x^1 \bmod N\} \quad \text{by rule [Assgn]} \\
& \implies \{z < \text{ord}(x, N) \wedge b \neq 1\} \quad \text{by rule [Conseq]} \\
& \{(z = \text{ord}(x, N) \wedge b = 1) \oplus (z < \text{ord}(x, N) \wedge b \neq 1)\} \quad \text{by rule [Conseq]} \\
& \text{while } (b \neq 1) \text{ do} \\
& \quad \{z < \text{ord}(x, N) \wedge b \neq 1\} \\
& \quad S' \\
& \quad \{(z = \text{ord}(x, N) \wedge b = 1) \oplus (z < \text{ord}(x, N) \wedge b \neq 1)\} \\
& \text{od} \\
& \longleftarrow \{z = \text{ord}(x, N) \wedge b = 1\} \quad \text{by rule [While]} \\
& \{ 2 \leq x \leq N - 1 \wedge z = \text{ord}(x, N) \} \quad \text{by rule [Conseq]}
\end{aligned}$$

To prove this correctness, we first consider the loop body of the order-finding algorithm, as displayed in Table 20 of Appendix B. Now let  $D' \triangleq (z = r \wedge b = 1) \oplus (z < r \wedge b \neq 1)$ , and  $S'$  the body of the while loop **OF**. From the reasoning in Table 20, we specify the correctness of  $S'$  as follows:

$$\{ z < r \wedge b \neq 1 \} S' \{ D' \} .$$

So the invariant of the while loop of **OF** can be  $D'$ , and by rule [While] we have

$$\{ D' \} \text{ while } (b \neq 1) \text{ do } S' \text{ od } \{ z = r \wedge b = 1 \} .$$

Finally, a proof outline of **OF** is given in Table 12 .

Then we introduce Shor's algorithm in Table 13. The function  $\text{random}(a, b)$  is used to randomly generate a number between  $a$  and  $b$ . The function  $\gcd(a, b)$  returns the greatest common divisor of  $a$  and  $b$ . The operator  $\equiv_N$  represents identity modulo  $N$ . **OF**( $x, N$ ) is the order-finding algorithm given before, which will return the order of  $x$  modulo  $N$  and assign the order to the classical variable  $z$ . The classical variable  $y$  stores one of the divisors of  $N$  and we use  $y|N$  to represent that  $N$  is divisible by  $y$ .

The correctness of Shor's algorithm can be specified by the Hoare triple:

$$\{ \text{cmp}(N) \} \text{Shor} \{ y|N \wedge y \neq 1 \wedge y \neq N \}$$

where  $\text{cmp}(N)$  is a predicate stating that  $N$  is a composite number greater than 0. The invariant of the while loop in Shor's algorithm can be

$$\begin{aligned}
& (2 \leq x \leq N - 1) \wedge ((y = \gcd(x, N) \wedge y \neq N) \\
& \quad \vee (y = \gcd(x^{r/2} - 1, N) \wedge y \neq N \wedge y \neq 1) \\
& \quad \vee (y = \gcd(x^{r/2} + 1, N) \wedge y \neq N \wedge y \neq 1)).
\end{aligned}$$

Table 13. A program for Shor's algorithm

```

1:  if 2 | N then
2:    y := 2;
3:  else
4:    x := random(2, N - 1);
5:    y := gcd(x, N);
6:    while y = 1 do
7:      z := OF(x, N);
8:      if 2 | z and  $x^{z/2} \not\equiv_N -1$  then
9:        if  $1 < \gcd(x^{z/2} - 1, N) < N$  then
10:          y := gcd( $x^{z/2} - 1, N$ );
11:        else
12:          y := gcd( $x^{z/2} + 1, N$ );
13:        fi
14:      else
15:        x := random(2, N - 1);
16:        y := gcd(x, N);
17:      fi
18:    od
19:  fi

```

A proof outline for the correctness of the algorithm is given in Table 21 of Appendix B.

## 6 Coq Formalization

As we can see in Section 5, manual reasoning about the correctness of quantum algorithms are indeed cumbersome. In order to alleviate this burden, we have formalized all the aforementioned concepts within the Coq proof assistant. Our framework, called CoqQLR, offers several distinct advantages: 1) The majority of the concepts and lemmas presented in this paper have been formalized in Coq with minimal simplification, preserving the full logical expressiveness of the theories; 2) In addition to the rules demonstrated within this paper, we have proved some other useful theorems that enhance our capacity for semi-automated reasoning about programs; 3) The algorithms discussed in the last section have all undergone correctness verification within Coq.

Our formalization relies on the library **QuantumLib**<sup>4</sup>. Figure 1 illustrates the interrelationships between different folders. In our formalization, we primarily utilize three key files from **QuantumLib**: **Summation** for summation operations, **Matrix** for matrix computations, and **Quantum** for basic quantum computing concepts. Building upon the foundation of **QuantumLib**, we first define the notations of basis vectors and distribution states, along with their associated concepts in the **QState** folder. The **QIMP** folder focuses on the syntax and semantics of classical-quantum programs, while the **QAssert** folder deals with assertions. In the **QRule** folder, we delineate the formalization of the rules mentioned in Section 4.2 and rigorously establish their soundness. Finally, in the **Examples** folder, we formally prove the correctness of the **addM** program, HHL algorithm, order-finding algorithm, and Shor's algorithm. Below we illustrate the formalization of selected key concepts in Coq. More details can be found in our repository of Coq scripts.

In the **Basic** file, we define the type `Base_vec` with two parameters  $n$  and  $i$  of type `nat`, representing the basis vector  $|i\rangle$  in an  $n$ -dimensional space. Some properties of vectors are given in

<sup>4</sup><https://github.com/inQWIRE/QuantumLib.git>

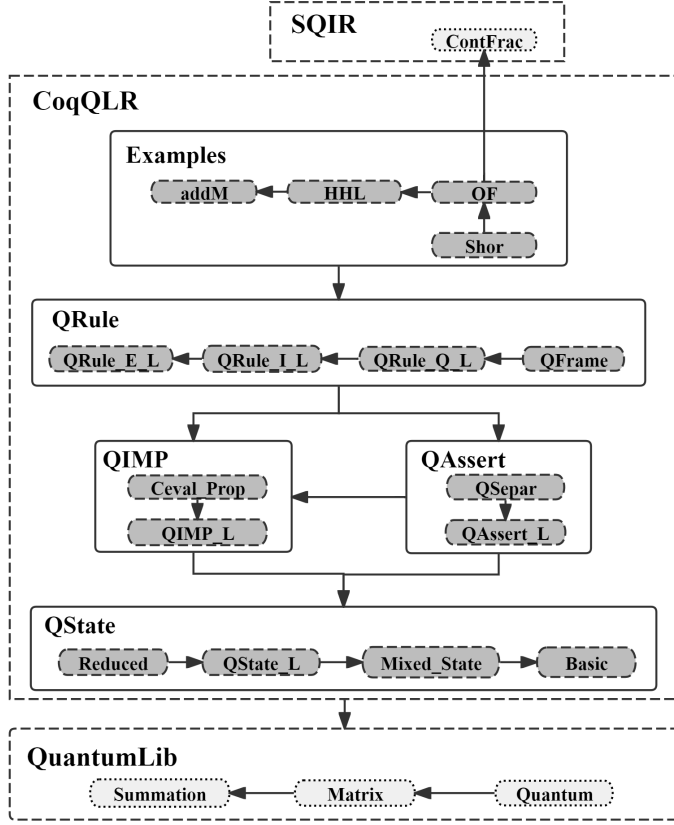


Fig. 1. Relationship between different files

Table 14. Before defining the concept of quantum state, we first define mixed states in the file **Mixed\_State**. Recall that in **QuantumLib**, a mixed state is defined as an ensemble with the sum of traces equal to one:

```
Inductive Mixed_State {n} : Matrix n n -> Prop :=
| Pure_S : forall q, Pure_State q -> Mixed_State q
| Mix_S : forall (p : R) q1 q2, 0 < p < 1 -> Mixed_State q1 -> Mixed_State q2
      -> Mixed_State (p .* q1 .+ (1-p) .* q2),
```

where the type **Pure\_State** denotes a pure state. This definition is equivalent to the notion of density operator. Since we mainly work with partial density operators, we define our mixed states by allowing the sum of all traces to be less than one.

```
Inductive Mixed_State {n} : Matrix n n -> Prop :=
| Pure_S : forall q, (0 <= p <= 1) -> Pure_State q -> Mixed_State (p .* q)
| Mix_S : forall (p1 p2 : R) q1 q2, 0 <= p1 -> 0 <= p2 -> p1+p2 <= 1 -> Mixed_State q1
      -> Mixed_State q2 -> Mixed_State (p1 .* q1 .+ p2 .* q2).
```

Table 14. Properties of Basis Vector

Name	Function
base1	$\forall (i \in \text{nat}),  i\rangle_{i0} = 1$
base0	$\forall (i, j \in \text{nat}), i \neq j \rightarrow  i\rangle_{j0} = 0$
base_decom	$\forall (n \in \text{nat}) (v \in \text{Vector } n), v = \sum_i (v_{i0}) \cdot  i\rangle$
base_inner_M	$\forall (m, n \in \text{nat}) (A : \text{Matrix } m \ n), \langle i   A   i \rangle = A_{ii}$
trace_base	$\forall (m, n \in \text{nat}) (A : \text{Matrix } m \ n), \sum_i \langle i   A   i \rangle = \text{tr}(A)$
base_inner_1	$\forall (i \in \text{nat}), \langle i   i \rangle = 1$
base_inner_0	$\forall (i, j \in \text{nat}), i \neq j \rightarrow \langle i   j \rangle = 0$
norm_base_1	$\forall (i \in \text{nat}), \   i\rangle \  = 1$
base_kron	$\forall (i, m, n \in \text{nat}),  i/n\rangle^m \otimes  i\%n\rangle^n =  i\rangle^{m*n}$
big_sum_I	$\sum_i  i\rangle \langle i  = I$
qubit0_base_kron	$\forall (i, n \in \text{nat}),  0\rangle \otimes  i\rangle^{2^n} =  i\rangle^{2^{(n+1)}}$
qubit1_base_kron	$\forall (i, n \in \text{nat}),  1\rangle \otimes  i\rangle^{2^n} =  i + 2^n\rangle^{2^{(n+1)}}$

When discussing the semantics of programs and the satisfaction relation of assertions, we focus on non-zero states with finite supports. To facilitate the discussion of the properties of these states, we further define the set of non-zero mixed states:

```

Inductive NZ_Mixed_State {n} : Matrix n n -> Prop :=
| Pure_S : forall q, (0 < p <= 1) -> Pure_State q -> NZ_Mixed_State (p .* q)
| Mix_S : forall (p1 p2 : R) q1 q2, 0 < p1 -> 0 < p2 -> p1+p2 <= 1 -> NZ_Mixed_State q1
  -> NZ_Mixed_State q2 -> NZ_Mixed_State (p1 .* q1 .+ p2 .* q2).

```

It can be verified that the type `NZ_Mixed_State` is preserved after many operations. Additionally, some important properties are summarized in Table 15.

In the `QState_L` file, we define classical states, quantum states, and distribution states. Within our framework CoqQLR, we use subscripts to represent variables. For instance, we denote the classical variable  $x_i$  with  $i$  and the set of variables  $\langle q_s, q_{s+1}, \dots, q_{e-1} \rangle$  with  $(s, e)$ . We define the type `cstate` as a Coq type of `list nat`, representing a classical state where the  $i$ -th element corresponds to the value of variable  $x_i$ . Syntactically, we represent a quantum state  $\rho$  with domain  $(s, e)$  using a  $2^{(e-s)}$ -dimensional matrix.

Definition `qstate (s e : nat) := Density (2 ^ (e-s))`.

In order to ensure that this definition is well-formed, we require that the matrix represents a mixed state. Similarly, since we are almost exclusively concerned with the properties of non-zero quantum states, we define a type `WF_qstate` to further ensure that this matrix is a non-zero mixed state and the domain is valid, that is,  $s \leq e$ :

Definition `WF_qstate {s e : nat} (rho : qstate s e) :=`  
`@NZ_Mixed_State (2 ^ (e - s)) rho /\ (s <= e)`.

Then a state is a pair with product type `(cstate * (qstate s e))`. A state is considered well-formed if its quantum component is well-formed.

Next, we proceed to define the notion of distribution state. Notably, a distribution is characterized by a mapping with a finite support set. To represent this distribution, we utilize the `FMapList` structure in Coq, which employs a `list` to encapsulate the finite mapping. A finite mapping from type `X`.`t` to type `elt` can be created with the core `FMapList.Make(X).t(elt)`, with the following internal structure:

Table 15. Properties of Mixed State

Name	Function
pure_state_trace_1	$\forall \rho, \text{Pure\_State } \rho \rightarrow \text{tr}(\rho) = 1$
nz_mixed_state_trace_real	$\forall \rho, \text{NZ\_Mixed\_State } \rho \rightarrow \text{Im}(\text{tr}(\rho)) = 0$
nz_mixed_state_trace_in01	$\forall \rho, \text{NZ\_Mixed\_State } \rho \rightarrow 0 < \text{Re}(\text{tr}(\rho)) \leq 1$
nz_mixed_state_Cmod_1	$\forall \rho, \text{NZ\_Mixed\_State } \rho \rightarrow 0 <  \text{tr}(\rho)  \leq 1$
nz_mixed_state_Cmod_plus	$\forall \rho_1, \rho_2, \text{NZ\_Mixed\_State } \rho_1$ $\rightarrow \text{NZ\_Mixed\_State } \rho_2$ $\rightarrow  \text{tr}(\rho_1 + \rho_2)  =  \text{tr}(\rho_1)  +  \text{tr}(\rho_2) $
Cauchy_Schwartz_ver1'	$\forall (n \in \text{nat})(u, v \in \text{Vector } n), u \neq 0$ $\rightarrow (\neg(\exists (c \in \mathbb{C}), c * u = v))$ $\rightarrow  \langle u, v \rangle ^2 <  \langle u, u \rangle  *  \langle v, v \rangle $
mixed_mult_trace_le_1	$\forall \rho_1, \rho_2, \text{Mixed\_State } \rho_1 \rightarrow \text{Mixed\_State } \rho_2$ $\rightarrow 0 \leq  \text{tr}(\rho_1 \times \rho_2)  \leq 1$
nz_mixed_mult_trace_lt_1	$\forall \rho_1, \rho_2, \text{NZ\_Mixed\_State } \rho_1 \rightarrow \text{NZ\_Mixed\_State } \rho_2$ $\rightarrow \neg(\exists c \in \mathbb{C}, c * \rho_1 = \rho_2) \rightarrow  \text{tr}(\rho_1 \times \rho_2)  < 1$
pure_sqrt_trace	$\forall \rho, \text{Pure\_State } \rho \rightarrow  \text{trace}(\rho \times \rho)  = 1$
nz_mixed_sqrt_trace	$\forall \rho, \text{NZ\_Mixed\_State } \rho \rightarrow \neg \text{Pure\_State } \rho$ $\rightarrow  \text{tr}(\rho \times \rho)  < 1$
nz_mixed_pure	$\forall \rho_1, \rho_2, \phi, \text{NZ\_Mixed\_State } \rho_1$ $\rightarrow \text{NZ\_Mixed\_State } \rho_2$ $\rightarrow \ \phi\  = 1 \rightarrow \rho_1 + \rho_2 =  \phi\rangle \langle \phi $ $\rightarrow \exists p_i, 0 < p_i \leq 1 \wedge \rho_i = p_i *  \phi\rangle \langle \phi  \text{ for } i = 1, 2$

Record slist (elt:Type) :=  
 {this :> list (X.t \* elt); sorted: sort (KeyOrderedType X.ltk elt) this}.

Definition t (elt:Type): Type := slist elt.

The structure is composed of two parts: this and sorted. The this component uses a list to represent a finite mapping  $f$ , where each element is a pair  $(a, b)$  of types  $(X.t * \text{elt})$ , indicating that  $f(a) = b$ . The sorted aspect ensures that the list is sorted, which is essential for maintaining distinct domains and ensuring that functions  $f_1$  and  $f_2$ , defined by the structure FMapList, are equal if and only if their list components are identical. This sorted requirement implies that type  $X$  must be an OrderedType. Consequently, we define the type dstate to represent the distribution state by the following code, a mapping from cstate to qstate:

```
Module Import StateMap := FMapList.Make(Cstate_as_OT).
Definition dstate (s e:nat) := StateMap.t (qstate s e).
```

The module Cstate\_as\_OT defines cstate as an OrderedType. We employ an element of type dstate to document the mapping relationships within the support set of a distribution  $\mu$ , thereby any element absent from its list part has the default value 0.

```
Definition option_qstate {s e:nat} (q: option (qstate s e)): (qstate s e) :=
  match q with
  |None => Zero
  |Some x => x
  end.
```

```
Definition d_find {s e:nat} (sigma:cstate) (mu: dstate s e): qstate s e :=
```

Table 16. Properties of Distribution States

Name	Function
d_trace_scale	$\forall \mu, p, 0 \leq p \rightarrow \ p * \mu\  = p * \ \mu\ $
d_app_empty_l	$\forall \mu, \mu + \varepsilon = \mu$
d_trace_app	$\forall \mu_1, \mu_2, \ \mu_1 + \mu_2\  = \ \mu_1\  + \ \mu_2\ $
d_trace_le_1_big_dapp	$\forall \{p_i\}_{i \in I}, \{\mu_i\}_{i \in I}, (\forall i, 0 < p_i) \rightarrow \ \sum_i p_i \mu_i\  \leq \sum_i p_i$
d_find_empty	$\forall \sigma, \varepsilon(\sigma) = \mathbf{0}$
d_find_scale	$\forall \mu, p, \sigma, (p * \mu)(\sigma) = p * (\mu(\sigma))$
d_find_app	$\forall \mu_1, \mu_2, \sigma, (\mu_1 + \mu_2)(\sigma) = \mu_1(\sigma) + \mu_2(\sigma)$
d_scale_empty	$\forall p, \mu, p * \varepsilon = \varepsilon$
d_scale_integral	$\forall p, \mu, p * \mu = \varepsilon \rightarrow p = 0 \vee \mu = \varepsilon$
d_scale_1_l	$\forall \mu, 1 * \mu = \mu$
d_scale_assoc	$\forall p_1, p_2, \mu, p_1 * (p_2 * \mu) = (p_1 * p_2) * \mu$
d_app_comm	$\forall \mu_1, \mu_2, \mu_1 + \mu_2 = \mu_2 + \mu_1$
d_app_assoc	$\forall \mu_1, \mu_2, \mu_3, \mu_1 + \mu_2 + \mu_3 = \mu_1 + (\mu_2 + \mu_3)$
d_scale_app_distr	$\forall p, \mu_1, \mu_2, p * (\mu_1 + \mu_2) = p * \mu_1 + p * \mu_2$
dstate_equal	$\forall \mu_1, \mu_2, \mu_1 = \mu_2 \leftrightarrow (\forall \sigma), \mu_1(\sigma) = \mu_2(\sigma)$

option\_qstate (StateMap.find sigma mu).

To guarantee the well-formedness of this construct, we introduce the type WF\_dstate.

```
Inductive WF_dstate_aux {s e:nat}: list(cstate * (qstate s e)) -> Prop:=
|WF_nil: WF_dstate_aux nil
|WF_cons st mu': WF_state st -> WF_dstate_aux mu' -> (d_trace_aux (st::mu'))<= 1
-> WF_dstate_aux (st::mu').
```

Definition WF\_dstate {s e:nat} (mu: dstate s e):Prop:= WF\_dstate\_aux (this mu).

This type is defined with the requirement that any quantum state mapped by an element within the support set is a non-zero mixed state, and the sum of their traces must be less than 1. Naturally, we have proved in Coq that the well-formedness of quantum states, states, and distribution states are preserved under their respective operations. In addition, we present some useful properties in Table 16.

In the **Reduced** file, we formalize the notation  $\rho|_V$  using the Reduced function and  $\mu|_V$  using the d\_reduced function. Similarly, we employ the subscript pair  $(l, r)$  to denote the set of quantum variables  $V = \langle q_l, q_{l+1}, \dots, q_{r-1} \rangle$ . A selection of lemmas and their corresponding functions are listed in Table 17. The table includes examples for Reduced, while properties for d\_reduced are analogous and thus omitted for brevity.

The formalization of our classical-quantum languages, assertions, and rules follows the structure of classical Hoare Logic. The syntax and semantics of our programming language are defined in the **QIMP\_L** file. The syntax of commands is specified by the type com. Let  $\mu$  be of type dstate. The semantics of a program  $c$  applied to  $\mu$  are determined by the function ceval in Coq, which applies  $c$  to each element in the list component of  $\mu$ .

```
Inductive ceval {s e:nat}: com -> dstate s e-> dstate s e-> Prop:=
|E_com: forall c (mu mu':dstate s e), WF_dstate mu
-> (ceval_single c (StateMap.this mu) (StateMap.this mu')) -> ceval c mu mu'.
```



Table 17. Properties of Restriction

Name	Function
Reduced_trace	$\forall \rho, V. \text{tr}(\rho _V) = \text{tr}(\rho)$
Reduce_Zero	$\forall V. \text{tr}(\mathbf{0} _V) = \mathbf{0}$
Reduced_refl	$\forall \rho, \rho _{\text{dom}(\rho)} = \rho$
Reduced_tensor_l	$\forall \rho_1, \rho_2. (\rho_1 \otimes \rho_2) _{\text{dom}(\rho_1)} = \text{tr}(\rho_2) * \rho_1$
Reduced_tensor_r	$\forall \rho_1, \rho_2. (\rho_1 \otimes \rho_2) _{\text{dom}(\rho_2)} = \text{tr}(\rho_1) * \rho_2$
Reduced_scale	$\forall \rho, V, p. (p * \rho) _V = p * (\rho _V)$
Reduced_plus	$\forall \rho_1, \rho_2, V. (\rho_1 + \rho_2) _V = (\rho_1 _V) + (\rho_2 _V)$
big_sum_Reduced	$\forall \{p_i\}_{i \in I}, V. (\sum_i (\rho_i)) _V = \sum_i ((\rho_i) _V)$
Reduced_assoc	$\forall \rho, V_1, V_2, V_1 \subseteq V_2 \rightarrow (\rho _{V_2}) _{V_1} = \rho _{V_1}$

The `ceval_single` function is sufficiently complex that we omit it here. For the sake of simplification, we formalize the semantics of the **while**-statement as:

$$[[\text{while } b \text{ then } c \text{ od}]]_{(\sigma, \rho)} = \begin{cases} [[c; \text{while } b \text{ then } c \text{ od}]]_{(\sigma, \rho)} & \text{if } [[b]]_{\sigma} = \text{true} \\ [[\text{skip}]]_{(\sigma, \rho)} & \text{if } [[b]]_{\sigma} = \text{false}. \end{cases}$$

This formalization is equivalent to the definition provided in our paper, as discussed in detail in the work of [4]. Additionally, we consider only those statements that terminate successfully. Furthermore, to ensure that the definition of `ceval` is well-defined, we establish two lemmas: `ceval_sorted` and `WF_ceval`.

```
Lemma ceval_sorted {s e:nat}: forall c (mu mu':list (cstate *qstate s e))
(Hm: Sorted (StateMap.Raw.PX.ltk (elt:=qstate s e)) mu)
(H:ceval_single c mu mu'),
Sorted (StateMap.Raw.PX.ltk (elt:=qstate s e)) mu'.
```

```
Lemma WF_ceval {s' e':nat}: forall c (mu mu':list (cstate * qstate s' e')),
WF_dstate_aux mu -> ceval_single c mu mu' -> WF_dstate_aux mu'.
```

The former ensures the sorted order of the output list, enabling the output distribution to be well-defined as a `dstate`, while the latter guarantees that the well-formedness of `dstate` is preserved, demonstrating that each statement that terminates successfully maintains the trace.

The **Ceval\_Prop** file contains two key theorems: `ceval_big_dapp` and `Reduced_ceval_swap`. The `ceval_big_dapp` theorem characterizes the linear behavior of the function  $[[c]]$ , considering those coefficients  $p_i$  such that  $p_i > 0$ . The `Reduced_ceval_swap` theorem elucidates the commutativity of the function with the restriction of the distribution.

The **QAssert\_L** file introduces the syntax and semantics of the assertion language as defined in Section 4.1. Some useful properties such as Lemma 4.2 and 4.4 are proved within this document. Furthermore, Lemma 4.1, 4.5 and A.1 are shown in the file titled **QSepar**. It is worth mentioning that in the **QSepar** file, we focus exclusively on assertions for which the sets of quantum variables can be expressed as pair  $(s, e)$ , as specified by `Considered_Formula`.

The rules in Table 5, except for the `[QFrame]` rule, are validated in the **QRule\_I\_L** file. The `[QFrame]` rule is specifically addressed in the **QFrame** file. The key to proving this rule is based on Lemma 4.3. Similarly, we restrict our attention to the assertions that satisfy `Considered_Formula` in this file. The rules in Table 6 are shown to be sound in **QRule\_Q\_L**. The validity of the rules presented in Table 7 is established within the context of **QRule\_E\_L**. In addition to the rules mentioned above, we also demonstrate some useful lemmas that aid in better automated reasoning

about algorithms. All the rules are formalized as theorems whose names are prefixed with "rule\_". Let us consider the [QInit] rule as an example:

```
Theorem rule_QInit: forall s e,
  {{BTrue}}
  <{ [[ s e ]] : Q= 0 }>
  {{(QExp_s s e (| 0 > _ (2 ^ (e-s))))}}.
Proof.
  unfold hoare_triple.
  intros s e s' e' (mu,IHmu) (mu', IHmu').
  intros.
  inversion_clear H; simpl in H2.
  rewrite sat_Assert_to_State in *.
  inversion_clear H0.
  apply sat_F.
  eapply WF_ceval. apply H. apply H2.
  apply rule_Qinit_aux' with mu.
  intuition. intuition. assumption.
Qed.
```

Finally, in the **Examples** folder, we demonstrate the correctness the algorithms presented in this paper. In the verification of the order-finding algorithm, we referred to a file named **ContFrac**, which involves the continued fraction algorithm and has been formally established by the work of [15]. Let us use the **addM** program as a case study to elucidate the process of algorithmic verification in Coq, by employing our developed framework. That program is formalized as follows:

```
Definition addM : com :=
  <{ [[0 1]] :Q= 0 ;
    [[1 2]] :Q= 0 ;
    hadamard [[0 1]];
    hadamard [[1 2]];
    v1 :=M [[0 1]];
    v2 :=M [[1 2]];
    v := (AId v1) + (AId v2) }>.
```

The correctness of the above program can be established by the following theorem.

```
Lemma correctness_addM:
  {{ BTrue }}
  addM
  {{APro [((1/2), (SPure (BEq (AId v) 1))) ; ((1/2), SPure (BNeq (AId v) 1))]} } .
Proof.
  (*The proof of the initialization part*)
  eapply rule_seq. eapply rule_QInit. simpl sub.
  eapply rule_seq. eapply rule_conseq_1. apply rule_OdotE.
  eapply rule_qframe' with (F2:= | | 0 > _ (2) >[ 1, 2]); simpl; try lia.
  apply Qsys_inter_empty'; try lia.
  split. eapply rule_QInit. split. apply inter_empty. left. reflexivity. lia.
  ...
Qed.
```

Table 18. Code Metric

Documents	Files	Lines of Code	Total
QState	Basic	600	5700
	Mixed_State	2000	
	QState_L	2200	
	Reduced	900	
QIMP	QIMP_L	2200	4100
	Ceval_Prop	1900	
QAssert	QAssert_L	2000	7000
	QSepar	5000	
QRule	QRule_E_L	2500	8500
	QRule_I_L	2300	
	QRule_Q_L	1300	
	QFrame	2400	
Examples	addM	500	3700
	HHL	1100	
	OF	1600	
	Shor	500	
Total			29000

In summary, our Coq development contains over 29,000 lines of code. A detailed view of the code statistics for each file is given in Table 18.

## 7 Conclusion and Future Work

We have presented a new quantum Hoare logic for classical–quantum programs. It includes distribution formulas for specifying probabilistic properties of classical assertions naturally, and at the same time allows for local reasoning. We have proved the soundness of the logic with respect to a denotational semantics and exhibited its usefulness in reasoning about the functional correctness of the HHL and Shor’s algorithms, which are non-trivial algorithms involving probabilistic behaviour due to quantum measurements and unbounded while loops. Finally, we have embedded the logic into the Coq proof assistant to alleviate the burden of manually reasoning about classical–quantum programs.

We have not yet precisely delimited the expressiveness of our logic. It is unclear whether the logic is relatively complete, which is an interesting future work to consider. Usually there are two categories of program logics when dealing with probabilistic behaviour: satisfaction-based or expectation-based [4]. Our logic belongs to the first category. In an expectation-based logic, e.g. the logic in [21, 22], the Hoare triple  $\{P\}c\{Q\}$  is valid in the sense that the expectation of an initial state satisfying  $P$  is a lower bound of the expectation of the final state satisfying  $Q$ . It would be interesting to explore local reasoning in expectation-based logics for classical–quantum programs such as those proposed in [5, 6].

## Acknowledgments

We would like to thank Professor Jean-François Monin for his valuable suggestions on the implementation of Coq. The work was supported by the National Key R&D Program of China under Grant No. 2023YFA1009403, the National Natural Science Foundation of China under Grant Nos.

62472175, 62072176, and 12271172, Shanghai Trusted Industry Internet Software Collaborative Innovation Center, and the “Digital Silk Road” Shanghai International Joint Lab of Trustworthy Intelligent Software under Grant No. 22510750100.

## References

- [1] Gilles Barthe, Justin Hsu, Mingsheng Ying, Nengkun Yu, and Li Zhou. 2020. Relational proofs for quantum programs. *Proc. ACM Program. Lang.* 4, POPL, Article 21 (Jan. 2020), 29 pages. <https://doi.org/10.1145/3371089>
- [2] Yves Bertot and Pierre Castéran. 2004. *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*. Springer, Berlin, Heidelberg. <https://doi.org/10.1007/978-3-662-07964-5>
- [3] Rohit Chadha, Paulo Mateus, and Amílcar Sernadas. 2006. Reasoning About Imperative Quantum Programs. *Electronic Notes in Theoretical Computer Science* 158 (2006), 19–39. <https://doi.org/10.1016/j.entcs.2006.04.003>
- [4] Yuxin Deng and Yuan Feng. 2022. Formal semantics of a classical-quantum language. *Theoretical Computer Science* 913 (2022), 73–93. <https://doi.org/10.1016/j.tcs.2022.02.017>
- [5] Yuan Feng, Sanjiang Li, and Mingsheng Ying. 2022. Verification of Distributed Quantum Programs. *ACM Trans. Comput. Log.* 23, 3 (2022), 19:1–19:40.
- [6] Yuan Feng and Mingsheng Ying. 2021. Quantum Hoare logic with classical variables. *ACM Transactions on Quantum Computing* 2, 4 (2021), 16:1–16:43.
- [7] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. 2009. Quantum algorithm for linear systems of equations. *Physical review letters* 103, 15 (2009), 150502.
- [8] Charles Antony Richard Hoare. 1969. An axiomatic basis for computer programming. *Commun. ACM* 12, 10 (1969), 576–580.
- [9] Yoshihiko Kakutani. 2009. A Logic for Formal Verification of Quantum Programs. In *Proceedings of the 13th Asian Computing Science Conference (Lecture Notes in Computer Science, Vol. 5913)*. Springer, Seoul, Korea, 79–93.
- [10] 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.
- [11] Yangjia Li and Dominique Unruh. 2021. Quantum Relational Hoare Logic with Expectations. In *48th International Colloquium on Automata, Languages, and Programming (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 198)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Virtual (Online), 136:1–136:20.
- [12] 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 (Lecture Notes in Computer Science, Vol. 11562)*. Springer, New York, USA, 187–207.
- [13] Michael A. Nielsen and Isaac L. Chuang. 2000. *Quantum computation and quantum information*. Cambridge university press, Cambridge, UK.
- [14] Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. 2002. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer, Berlin, Heidelberg.
- [15] Yuxiang Peng, Keshu Hietala, Runzhou Tao, Liyi Li, Robert Rand, Michael Hicks, and Xiaodi Wu. 2023. A formally certified end-to-end implementation of Shor’s factorization algorithm. *Proceedings of the National Academy of Sciences* 120, 21 (2023), e2218775120. <https://doi.org/10.1073/pnas.2218775120>
- [16] Peter Selinger. 2004. Towards a quantum programming language. *Mathematical Structures in Computer Science* 14, 4 (2004), 527–586.
- [17] Peter W. Shor. 1994. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, Santa Fe, NM, USA, 124–134.
- [18] Dominique Unruh. 2019. Quantum Hoare Logic with Ghost Variables. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science*. IEEE, Vancouver, Canada, 1–13.
- [19] Dominique Unruh. 2019. Quantum relational Hoare logic. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 1–31.
- [20] John von Neumann. 1955. *Mathematical Foundations of Quantum Mechanics*. Princeton University Press, Princeton, NJ.
- [21] Mingsheng Ying. 2012. Floyd–Hoare logic for quantum programs. *ACM Transactions on Programming Languages and Systems* 33, 6 (2012), 1–49.
- [22] Mingsheng Ying. 2016. *Foundations of Quantum Programming*. Morgan Kaufmann, San Francisco, CA.
- [23] Mingsheng Ying and Yuan Feng. 2011. A Flowchart Language for Quantum Programming. *IEEE Trans. Software Eng.* 37, 4 (2011), 466–485.
- [24] Mingsheng Ying, Li Zhou, Yangjia Li, and Yuan Feng. 2022. A proof system for disjoint parallel quantum programs. *Theor. Comput. Sci.* 897 (2022), 164–184.

- [25] Li Zhou, Gilles Barthe, Justin Hsu, Mingsheng Ying, and Nengkun Yu. 2021. A Quantum Interpretation of Bunched Logic & Quantum Separation Logic. In *Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science*. IEEE, Rome, Italy, 1–14.
- [26] 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 (2023), 33 pages.
- [27] 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*. Association for Computing Machinery (ACM), Phoenix, AZ, USA, 1149–1162. <https://doi.org/10.1145/3314221.3314583>

## A Proof of Proposition 4.6

Before delving into the proof of Proposition 4.6, we establish some technical lemmas that will be instrumental in our argument.

**LEMMA A.1.** *Assume that  $V \subseteq \mathbf{QVar}$ . Let  $\sigma$  be a classical state,  $\rho \in \mathcal{D}^-(\mathcal{H}_V)$  a quantum state, and  $F$  an assertion. If  $(\sigma, \rho) \models F$ , there exist  $\rho_1 \in \mathcal{D}^-(\mathcal{H}_{\text{qfree}(F)})$  and  $\rho_2 \in \mathcal{D}^-(\mathcal{H}_{V \setminus \text{qfree}(F)})$  such that  $\rho = \rho_1 \otimes \rho_2$ .*

The proofs of Lemma A.1 are detailed in our CoqQLR framework.

**LEMMA A.2.** *Let  $V \subseteq \mathbf{QVar}$  and  $|s\rangle$  be a quantum expression. The set  $\{\rho \in \mathcal{D}^-(\mathcal{H}_V) \mid \rho \models |s\rangle\} \cup \{0\}$  is a closed set.*

**PROOF.** By Lemma A.1, for any  $\rho$ , if  $\rho \models |s\rangle$  then there exist  $\rho_1 \in \mathcal{D}^-(\mathcal{H}_{\text{qfree}(|s|)})$  and  $\rho_2 \in \mathcal{D}^-(\mathcal{H}_{V \setminus \text{qfree}(|s|)})$  such that  $\rho = \rho_1 \otimes \rho_2$ . Furthermore, by the semantics of quantum expressions, we have  $\rho|_{\text{qfree}(|s|)} = p \cdot |s\rangle \langle s|$  for some  $p \in (0, 1]$ . Consequently,

$$\begin{aligned}
 & \{\rho \in \mathcal{D}^-(\mathcal{H}_V) \mid \rho \models |s\rangle\} \cup \{0\} \\
 &= \{(p \cdot |s\rangle \langle s|) \otimes \rho' \mid p \in (0, 1], \rho' \in \mathcal{D}^-(\mathcal{H}_{V \setminus \text{qfree}(|s|)})\} \cup \{0\} \\
 &= \{(p \cdot |s\rangle \langle s|) \otimes \rho' \mid p \in [0, 1], \rho' \in \mathcal{D}^-(\mathcal{H}_{V \setminus \text{qfree}(|s|)})\} \\
 &= \{(p \cdot |s\rangle \langle s|) \mid p \in [0, 1]\} \otimes \mathcal{D}^-(\mathcal{H}_{V \setminus \text{qfree}(|s|)}).
 \end{aligned}$$

Since the state space  $\{(p \cdot |s\rangle \langle s|) \mid p \in [0, 1]\}$  and  $\mathcal{D}^-(\mathcal{H}_{V \setminus \text{qfree}(|s|)})$  are both closed sets, it follows that  $\{\rho \in \mathcal{D}^-(\mathcal{H}_V) \mid \rho \models |s\rangle\} \cup \{0\}$  is also a closed set.  $\square$

**LEMMA A.3.** *Let  $A$  and  $B$  be two closed sets of distributions,  $\lambda$  a real number in  $[0, 1]$ . The set  $C = \{\lambda a + (1 - \lambda)b \mid a \in A, b \in B\}$  is also closed.*

**PROOF.** To prove that  $C$  is closed, we first observe that for any continuous function  $F$ , if  $a$  and  $b$  both range over closed sets, then the range of  $F(a, b)$  is also closed. Consider the function  $F(a, b) = \lambda a + (1 - \lambda)b$ , where  $\lambda$  is a fixed real number in  $[0, 1]$ . It is evident that  $F$  is a continuous function, as it represents a linear combination of  $a$  and  $b$  with fixed coefficients. Therefore, given that  $A$  and  $B$  are closed sets, the set  $C = \{F(a, b) \mid a \in A, b \in B\}$  is closed.  $\square$

**LEMMA A.4.** *Let  $A$  and  $B$  be two closed sets. The set  $C = \{\lambda a + (1 - \lambda)b \mid a \in A, b \in B, \lambda \in [0, 1]\}$  is also closed.*

**PROOF.** The proof follows a similar structure to that of the preceding lemma. Let  $F(a, b, \lambda) = \lambda a + (1 - \lambda)b$ . We get that  $F$  is a continuous function. Given that  $A$  and  $B$  are both closed sets, and noting that the interval  $[0, 1]$  is also a closed set, it follows that  $C = \{F(a, b, \lambda) \mid a \in A, b \in B, \lambda \in [0, 1]\}$  is a closed set.  $\square$

**LEMMA A.5.** *Let  $V \subseteq \mathbf{QVar}$ . For any state formula  $F$ , the set  $\llbracket F \rrbracket_V \triangleq \{\mu \mid \mu \models F, \text{dom}(\mu) = V\}$  is a closed set.*

PROOF. We proceed by induction on the structure of  $F$ . For brevity, we omit the subscript.

- **Base Case:**  $F \equiv P$ . Suppose there exists a sequence of distributions  $\{\mu_n\}_{n \geq 0}$  such that each  $\mu_n \in \llbracket P \rrbracket$ . Let  $\mu$  be the limit  $\lim_{n \rightarrow \infty} \mu_n$  if it exists. We want to show that  $\mu \in \llbracket P \rrbracket$ . By definition, we need to prove that for any  $\sigma \in \Sigma$ , if  $\text{tr}(\mu(\sigma)) > 0$ , then  $\sigma \models P$ . Since  $\mu(\sigma) = \lim_{n \rightarrow \infty} (\mu_n(\sigma))$ ,  $\text{tr}(\mu(\sigma)) > 0$  implies that there exists  $N$  such that for all  $n > N$ ,  $\text{tr}(\mu_n(\sigma)) > 0$ . By definition, this implies  $\sigma \models P$ .
- **Base Case:**  $F \equiv |s\rangle$ . Similarly, let  $\{\mu_n\}_{n \geq 0}$  be a sequence of distributions such that each  $\mu_n \in \llbracket |s\rangle \rrbracket$ , and let  $\mu$  be its limit. We show that  $\mu \in \llbracket |s\rangle \rrbracket$ . By definition, we need to prove that for any  $\sigma \in \Sigma$ , if  $\text{tr}(\mu(\sigma)) > 0$ , then  $\mu(\sigma) \models |s\rangle$ . For any  $\sigma \in \Sigma$ , since  $\mu(\sigma) = \lim_{n \rightarrow \infty} (\mu_n(\sigma))$  and  $\mu_n(\sigma) \in \{\rho \in \mathcal{D}^-(\mathcal{H}_V) \mid \rho \models |s\rangle\} \cup \{0\}$  for any  $n$ , a closed set by Lemma A.2, it follows that  $\mu(\sigma) \in \{\rho \in \mathcal{D}^-(\mathcal{H}_V) \mid \rho \models |s\rangle\} \cup \{0\}$ . If  $\text{tr}(\mu(\sigma)) > 0$ , then  $\mu(\sigma) \neq 0$ , meaning that  $\mu(\sigma) \models |s\rangle$ .
- **Inductive Step:**  $F \equiv F_1 \wedge F_2$ . By Lemma 4.4, we see that  $\llbracket F_1 \wedge F_2 \rrbracket = \llbracket F_1 \rrbracket \cap \llbracket F_2 \rrbracket$ . By the induction hypothesis, both  $\llbracket F_1 \rrbracket$  and  $\llbracket F_2 \rrbracket$  are closed sets. Since the intersection of two closed sets is a closed set, we conclude that  $\llbracket F_1 \wedge F_2 \rrbracket$  is closed.
- **Inductive Step:**  $F \equiv F_1 \odot F_2$ . When  $\text{dom}(F_1) \cap \text{dom}(F_2) \neq \emptyset$ , we have  $\llbracket F_1 \odot F_2 \rrbracket = \emptyset$ , which is a closed set. Otherwise, we have  $\llbracket F_1 \odot F_2 \rrbracket = \llbracket F_1 \rrbracket \cap \llbracket F_2 \rrbracket$  by Lemma 4.4. By the conclusion in the previous item, we see that  $\llbracket F_1 \odot F_2 \rrbracket$  is also closed.  $\square$

PROPOSITION A.6 (PROPOSITION 4.6). *Let  $V \subseteq \mathbf{Qvar}$ . For any assertion  $D$ , the set  $\llbracket D \rrbracket_V \triangleq \{\mu \mid \mu \models D, \text{dom}(\mu) = V\}$  is a closed set.*

PROOF. We proceed by induction on the structure of  $D$ .

- $D \equiv F$ : It immediately follows from Lemma A.5.
- $D \equiv \oplus_{i \in I} p_i F_i$ : Since  $\llbracket \oplus_{i \in I} p_i F_i \rrbracket = \{\sum_{i \in I} p_i \mu_i \mid \mu_i \in \llbracket F_i \rrbracket\}$ , which is a linear combination of  $\{\llbracket F_i \rrbracket\}_{i \in I}$  with weights  $\{p_i\}_{i \in I}$ . By Lemma A.5,  $\llbracket F_i \rrbracket$  is closed for any  $i$ . It follows from lemma A.3 that  $\llbracket \oplus_{i \in I} p_i F_i \rrbracket$  is also closed.
- $D \equiv \oplus_{i \in I} F_i$ : Note that  $\llbracket \oplus_{i \in I} F_i \rrbracket = \{\mu \mid \mu = \sum_{i \in I} p_i \mu_i\}$  for some  $p_i$  such that  $\sum_{i \in I} p_i = 1$  and  $\mu_i$  such that  $\mu_i \in \llbracket F_i \rrbracket$ . By Lemma A.5, the set  $\llbracket F_i \rrbracket$  is closed for each  $i \in I$ . It follows from Lemma A.4 that  $\llbracket \oplus_{i \in I} F_i \rrbracket$  is a closed set.  $\square$

## B Proof outlines for functional correctness of programs

Table 19. Proof outline for the loop body of the HHL algorithm

```

{ v = 0 }
{ true }      by rule [PT]
{ true  $\odot$  true  $\odot$  true }      by rule [OdotE]
 $\Longleftrightarrow$  { true }  $\bar{p} := |0\rangle^{\otimes n}; \{ |0\rangle_{\bar{p}}^{\otimes n} \}$       by rule [QInit]
 $\Longleftrightarrow$  { true }  $\bar{q} := |0\rangle^{\otimes m}; \{ |0\rangle_{\bar{q}}^{\otimes m} \}$       by rule [QInit]
 $\Longleftrightarrow$  { true }  $r := |0\rangle; \{ |0\rangle_r \}$       by rule [QInit]
 $\Longleftrightarrow$  {  $|0\rangle_{\bar{q}}^{\otimes m} \}$   $U_b[\bar{q}]; \{ |b\rangle_{\bar{q}} \}$       by rule [QUnit]
 $\Longleftrightarrow$  {  $|0\rangle_{\bar{p}}^{\otimes n} \}$   $H^{\otimes n}[\bar{p}]; \{ \frac{1}{\sqrt{N}} \sum_{z=0}^{N-1} |z\rangle_{\bar{p}} \}$       by rule [QUnit]
{  $(\frac{1}{\sqrt{N}} \sum_{z=0}^{N-1} |z\rangle_{\bar{p}})(\sum_j b_j |j\rangle_{\bar{q}}) |0\rangle_r \}$       by rule [QFrame]
 $\Rightarrow$  {  $\sum_j b_j \frac{1}{\sqrt{N}} \sum_{z=0}^{N-1} |z\rangle_{\bar{p}} |j\rangle_{\bar{q}} \}$ 
 $U_f[\bar{p}\bar{q}];$ 
 $\Longleftrightarrow$  {  $\sum_j b_j \frac{1}{\sqrt{N}} \sum_{z=0}^{N-1} e^{2\pi i \phi_j z} |z\rangle_{\bar{p}} |j\rangle_{\bar{q}} \}$       by rule [QUnit]
```

$$\begin{aligned}
& \text{QFT}^{-1}[\bar{p}]; \\
& \{ \sum_j b_j |\phi_j \cdot N\rangle_{\bar{p}} |j\rangle_{\bar{q}} |0\rangle_r \} \quad \text{by rule [QUnit]} \\
& \{ \sum_j b_j |\phi_j \cdot N\rangle_{\bar{p}} |0\rangle_r |j\rangle_{\bar{q}} \} \quad \text{by rule [ReArr]} \\
& U_c[\bar{p}r]; \\
& \{ \sum_j b_j |\tilde{\phi}_j\rangle_{\bar{p}} (\sqrt{1 - \frac{C^2}{\phi_j^2}} |0\rangle + \frac{C}{\phi_j} |1\rangle)_r |j\rangle_{\bar{q}} \} \quad \text{by rule [QUnit]} \\
& \{ \sum_j b_j |\tilde{\phi}_j\rangle_{\bar{p}} |j\rangle_{\bar{q}} (\sqrt{1 - \frac{C^2}{\phi_j^2}} |0\rangle + \frac{C}{\phi_j} |1\rangle)_r \} \quad \text{by rule [ReArr]} \\
& \text{QFT}[\bar{p}]; \\
& \{ \sum_{j=1}^N b_j (\frac{1}{\sqrt{N}} \sum_{z=0}^{N-1} e^{2\pi i \phi_j z} |z\rangle_{\bar{p}}) |j\rangle_{\bar{q}} (\sqrt{1 - \frac{C^2}{\phi_j^2}} |0\rangle + \frac{C}{\phi_j} |1\rangle)_r \} \quad \text{by rule [QUnit]} \\
& U_f^\dagger[\bar{p}\bar{q}]; \\
& \{ \sum_j b_j \frac{1}{\sqrt{N}} \sum_{z=0}^{N-1} |z\rangle_{\bar{p}} |j\rangle_{\bar{q}} (\sqrt{1 - \frac{C^2}{\phi_j^2}} |0\rangle + \frac{C}{\phi_j} |1\rangle)_r \} \quad \text{by rule [QUnit]} \\
& H^{\otimes n}[\bar{p}]; \\
& \{ \sum_j b_j |0\rangle_{\bar{p}}^{\otimes n} |j\rangle_{\bar{q}} (\sqrt{1 - \frac{C^2}{\phi_j^2}} |0\rangle + \frac{C}{\phi_j} |1\rangle)_r \} \quad \text{by rule [QUnit]} \\
& \{ \sum_j b_j |0\rangle_{\bar{p}}^{\otimes n} |j\rangle_{\bar{q}} (\sqrt{1 - \frac{C^2}{\phi_j^2}} |0\rangle + \frac{C}{\phi_j} |1\rangle)_r \wedge (v=0)[0/v] \wedge (v=1)[1/v] \} \quad \text{by rule [Conseq]} \\
& v := M[r]; \\
& \{ \sum_j |b_j|^2 (1 - \frac{C^2}{\phi_j^2}) \cdot (\sum_j b_j \sqrt{1 - \frac{C^2}{\phi_j^2}} |0\rangle_{\bar{p}}^{\otimes n} |j\rangle_{\bar{q}} |0\rangle_r / \sqrt{\sum_j |b_j|^2 (1 - \frac{C^2}{\phi_j^2})}) \wedge (v=0) \\
& \oplus \sum_j |\frac{b_j C}{\phi_j}|^2 \cdot (\sum_j b_j \frac{C}{\phi_j} |0\rangle_{\bar{p}}^{\otimes n} |j\rangle_{\bar{q}} |1\rangle_r / \sqrt{\sum_j |\frac{b_j C}{\phi_j}|^2}) \wedge (v=1) \} \quad \text{by rule [QMeas]} \\
& \{ (\sum_j b_j \sqrt{1 - \frac{C^2}{\phi_j^2}} |0\rangle_{\bar{p}}^{\otimes n} |j\rangle_{\bar{q}} |0\rangle_r / \sqrt{\sum_j |b_j|^2 (1 - \frac{C^2}{\phi_j^2})}) \wedge (v=0) \\
& \oplus (\sum_j \frac{b_j C}{\phi_j} |0\rangle_{\bar{p}}^{\otimes n} |j\rangle_{\bar{q}} |1\rangle_r / \sqrt{\sum_j |\frac{b_j C}{\phi_j}|^2}) \wedge (v=1) \} \quad \text{by rule [Oplus]} \\
& \{ (v=0) \oplus ((\sum_j \frac{b_j}{\lambda_j} |0\rangle_{\bar{p}}^{\otimes n} |j\rangle_{\bar{q}} |1\rangle_r / \sqrt{\sum_j |\frac{b_j}{\lambda_j}|^2}) \wedge (v=1)) \} \quad \text{by rule [Conseq]} \\
& \{ (v=0) \oplus ((|0\rangle_{\bar{p}}^{\otimes n} |x\rangle_{\bar{q}} |1\rangle_r) \wedge (v=1)) \} \quad \text{by (5) and rule [Conseq]}
\end{aligned}$$

Table 20. Proof outline of the loop body of the **OF** program

$$\begin{aligned}
& \{ z < r \wedge b \neq 1 \} \\
& \{ z < r \wedge b \neq 1 \odot \text{true} \odot \text{true} \} \quad \text{by rule [OdotE]} \\
& \iff \{ \text{true} \} \quad \bar{q} := |0\rangle_{\bar{q}}^{\otimes t}; \quad \{ |0\rangle_{\bar{q}}^{\otimes t} \} \quad \text{by rule [QInit]} \\
& \iff \{ \text{true} \} \quad \bar{p} := |0\rangle_{\bar{p}}^{\otimes L}; \quad \{ |0\rangle_{\bar{p}}^{\otimes L} \} \quad \text{by rule [QInit]} \\
& \iff \{ |0\rangle_{\bar{q}}^{\otimes t} \} \quad H^{\otimes t}[\bar{q}]; \quad \{ |+\rangle_{\bar{q}}^{\otimes L} \} \quad \text{by rule [QUnit]} \\
& \iff \{ |0\rangle_{\bar{p}}^{\otimes L} \} \quad U_+[\bar{p}]; \quad \{ |1\rangle_{\bar{p}} \} \quad \text{by rule [QUnit]} \\
& \{ \frac{1}{\sqrt{r^{2t}}} \sum_{s=0}^{r-1} \sum_{j=0}^{2^t-1} |j\rangle_{\bar{q}} |u_s\rangle_{\bar{p}} \} \quad \text{by rule [QFrame]} \\
& CU[\bar{q}\bar{p}]; \\
& \{ \frac{1}{\sqrt{r^{2t}}} \sum_{s=0}^{r-1} \sum_{j=0}^{2^t-1} e^{2\pi i j s / r} |j\rangle_{\bar{q}} |u_s\rangle_{\bar{p}} \} \quad \text{by rule [QUnit]} \\
& \{ \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} (\frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} e^{\frac{2\pi i}{2^t} j \frac{s}{r} 2^t} |j\rangle_{\bar{q}}) |u_s\rangle_{\bar{p}} \} \quad \text{by rule [Conseq]}
\end{aligned}$$

$$\begin{aligned}
& \text{QFT}^{-1}[\bar{q}]; \\
& \{\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |\frac{s}{r} 2^t\rangle_{\bar{q}} |u_s\rangle_{\bar{p}}\} \quad \text{by rule [QUnit]} \\
& \{\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |\frac{s}{r} 2^t\rangle_{\bar{q}} |u_s\rangle_{\bar{p}} \wedge \wedge_{s=0}^{r-1} (z' = \frac{s}{r} 2^t) [\frac{s}{r} 2^t / z']\} \quad \text{by rule [Conseq]} \\
& z' := M[\bar{q}]; \\
& \{\oplus_s \frac{1}{r} \cdot (|\frac{s}{r} 2^t\rangle_{\bar{q}} |u_s\rangle_{\bar{p}} \wedge z' = \frac{s}{r} 2^t)\} \quad \text{by rule [QMeas]} \\
& \{\oplus_s |\frac{s}{r} 2^t\rangle_{\bar{q}} |u_s\rangle_{\bar{p}} \wedge z' = \frac{s}{r} 2^t\} \quad \text{by rule [Oplus]} \\
& \{\oplus_s z' = \frac{s}{r} 2^t\} \quad \text{by rule [Conseq]} \\
& \{\oplus_s ((z = r \oplus z < r) [f(\frac{z'}{2^t}) / z])\} \quad \text{by rule [Conseq]} \\
& z := f(\frac{z'}{2^t}); \\
& \{z = r \oplus z < r\} \quad \text{by rule [Assgn]} \\
& \{((z = r \wedge b = 1) \oplus (z < r \wedge b \neq 1)) [(x^z \bmod N) / b]\} \quad \text{by rule [Conseq]} \\
& b := x^z \bmod N; \\
& \{(z = r \wedge b = 1) \oplus (z < r \wedge b \neq 1)\} \quad \text{by rule [Assgn]}
\end{aligned}$$

Table 21. Proof outline for Shor's algorithm

$$\begin{aligned}
& \{\text{cmp}(N)\} \\
1: & \text{if } 2 \mid N \text{ then} \\
& \quad \{\text{cmp}(N) \wedge N \bmod 2 = 0\} \\
& \quad \{\text{cmp}(N) \wedge N \bmod 2 = 0 \wedge (y = 2) [2/y]\} \quad \text{by rule [Conseq]} \\
& \quad \Rightarrow \{(y = 2) [2/y]\} \\
2: & \quad y := 2; \\
& \quad \Leftarrow \{y = 2\} \quad \text{by rule [Assgn]} \\
& \quad \{\text{cmp}(N) \wedge N \bmod 2 = 0 \wedge y = 2\} \quad \text{by rule [Qframe]} \\
& \quad \{y \mid N \wedge y \neq 1 \wedge y \neq N\} \quad \text{by rule [Conseq]} \\
3: & \quad \text{else} \\
& \quad \{\text{cmp}(N) \wedge N \bmod 2 \neq 0\} \\
& \quad \{(2 \leq x \leq N-1) [\text{random}(2, N-1)/x]\} \quad \text{by rule [Conseq]} \\
4: & \quad x := \text{random}(2, N-1); \\
& \quad \{(2 \leq x \leq N-1)\} \quad \text{by rule [Assgn]} \\
& \quad \{(2 \leq x \leq N-1) \wedge (y = \text{gcd}(x, N)) [\text{gcd}(x, N)/y]\} \quad \text{by rule [Conseq]} \\
& \quad \Rightarrow \{(y = \text{gcd}(x, N)) [\text{gcd}(x, N)/y]\} \\
5: & \quad y := \text{gcd}(x, N); \\
& \quad \Leftarrow \{(y = \text{gcd}(x, N))\} \quad \text{by rule [Assgn]} \\
& \quad \{(2 \leq x \leq N-1) \wedge (y = \text{gcd}(x, N))\} \quad \text{by rule [Qframe]} \\
& \quad \{(2 \leq x \leq N-1) \wedge ((y = \text{gcd}(x, N) \wedge y \neq N) \\
& \quad \quad \vee (y = \text{gcd}(x^{r/2} - 1, N) \wedge y \neq N \wedge y \neq 1) \\
& \quad \quad \vee (y = \text{gcd}(x^{r/2} + 1, N) \wedge y \neq N \wedge y \neq 1))\} \quad \text{by rule [Conseq]} \\
6: & \quad \text{while } y = 1 \text{ do} \\
& \quad \quad \{(2 \leq x \leq N-1) \wedge \text{gcd}(x, N) = 1\} \quad \text{by rule [Conseq]} \\
7: & \quad \quad OF(x, N); \\
& \quad \quad \{(2 \leq x \leq N-1) \wedge z = \text{ord}(x, N)\} \\
8: & \quad \quad \text{if } 2 \mid z \text{ and } x^{z/2} \not\equiv -1 \pmod{N} \text{ then} \\
& \quad \quad \quad \{(2 \leq x \leq N-1) \wedge z = \text{ord}(x, N) \wedge 2 \mid z \wedge x^{z/2} \not\equiv_N -1\} \\
& \quad \quad \quad \{(2 \leq x \leq N-1) \wedge z = \text{ord}(x, N) \\
& \quad \quad \quad \quad \wedge ((\text{gcd}(x^{z/2} - 1, N) \neq 1 \wedge \text{gcd}(x^{z/2} - 1, N) \neq N))\}
\end{aligned}$$



9:  $\vee(\gcd(x^{z/2} + 1, N) \neq 1 \wedge \gcd(x^{z/2} + 1, N) \neq N))\}$  by rule [Conseq]  
**if**  $1 < (\gcd(x^{z/2} - 1, N) < N)$  **then**  
 $\{(2 \leq x \leq N - 1) \wedge z = \text{ord}(x, N) \wedge (\gcd(x^{z/2} - 1, N) \neq 1 \wedge \gcd(x^{z/2} - 1, N) \neq N)\}$   
 $\{(2 \leq x \leq N - 1)$   
 $\wedge (y = \gcd(x^{r/2} - 1, N) \wedge y \neq N \wedge y \neq 1)[\gcd(x^{z/2} - 1, N)/y]\}$  by rule [Conseq]  
 $\implies \{(y = \gcd(x^{r/2} - 1, N) \wedge y \neq N \wedge y \neq 1)[\gcd(x^{z/2} - 1, N)/y]\}$   
 10:  $y := \gcd(x^{z/2} - 1, N);$   
 $\Leftarrow \{(y = \gcd(x^{r/2} - 1, N) \wedge y \neq N \wedge y \neq 1)\}$  by rule [Assgn]  
 $\{(2 \leq x \leq N - 1) \wedge (y = \gcd(x^{r/2} - 1, N) \wedge y \neq N \wedge y \neq 1)\}$  by rule [Qframe]  
 $\{(2 \leq x \leq N - 1) \wedge ((y = \gcd(x, N) \wedge y \neq N)$   
 $\vee (y = \gcd(x^{r/2} - 1, N) \wedge y \neq N \wedge y \neq 1)$   
 $\vee (y = \gcd(x^{r/2} + 1, N) \wedge y \neq N \wedge y \neq 1))\}$  by rule [Conseq]  
 11: **else**  
 $\{(2 \leq x \leq N - 1) \wedge z = \text{ord}(x, N) \wedge (\gcd(x^{z/2} + 1, N) \neq 1 \wedge \gcd(x^{z/2} + 1, N) \neq N)\}$   
 $\{(2 \leq x \leq N - 1)$   
 $\wedge (y = \gcd(x^{r/2} + 1, N) \wedge y \neq N \wedge y \neq 1)[\gcd(x^{z/2} + 1, N)/y]\}$  by rule [Conseq]  
 $\implies \{(y = \gcd(x^{r/2} + 1, N) \wedge y \neq N \wedge y \neq 1)[\gcd(x^{z/2} + 1, N)/y]\}$   
 12:  $y := \gcd(x^{z/2} + 1, N);$   
 $\Leftarrow \{(y = \gcd(x^{r/2} + 1, N) \wedge y \neq N \wedge y \neq 1)\}$  by rule [Assgn]  
 $\{(2 \leq x \leq N - 1) \wedge (y = \gcd(x^{r/2} + 1, N) \wedge y \neq N \wedge y \neq 1)\}$  by rule [Qframe]  
 $\{(2 \leq x \leq N - 1) \wedge ((y = \gcd(x, N) \wedge y \neq N)$   
 $\vee (y = \gcd(x^{r/2} - 1, N) \wedge y \neq N \wedge y \neq 1)$   
 $\vee (y = \gcd(x^{r/2} + 1, N) \wedge y \neq N \wedge y \neq 1))\}$  by rule [Conseq]  
 13: **fi**  
 $\{(2 \leq x \leq N - 1) \wedge ((y = \gcd(x, N) \wedge y \neq N)$   
 $\vee (y = \gcd(x^{r/2} - 1, N) \wedge y \neq N \wedge y \neq 1)$   
 $\vee (y = \gcd(x^{r/2} + 1, N) \wedge y \neq N \wedge y \neq 1))\}$  by rule [Cond]  
 14: **else**  
 $\{(2 \leq x \leq N - 1)[\text{random}(2, N - 1)/x]\}$  by rule [Conseq]  
 15:  $x := \text{random}(2, N - 1);$   
 $\{(2 \leq x \leq N - 1)\}$  by rule [Assgn]  
 $\{(2 \leq x \leq N - 1) \wedge (y = \gcd(x, N))[\gcd(x, N)/y]\}$  by rule [Conseq]  
 $\implies \{(y = \gcd(x, N))[\gcd(x, N)/y]\}$   
 16:  $y := \gcd(x, N);$   
 $\Leftarrow \{(y = \gcd(x, N))\}$  by rule [Assgn]  
 $\{(2 \leq x \leq N - 1) \wedge (y = \gcd(x, N))\}$  by rule [Qframe]  
 $\{(2 \leq x \leq N - 1) \wedge ((y = \gcd(x, N) \wedge y \neq N)$   
 $\vee (y = \gcd(x^{r/2} - 1, N) \wedge y \neq N \wedge y \neq 1)$   
 $\vee (y = \gcd(x^{r/2} + 1, N) \wedge y \neq N \wedge y \neq 1))\}$  by rule [Conseq]  
 17: **fi**  
 $\{(2 \leq x \leq N - 1) \wedge ((y = \gcd(x, N) \wedge y \neq N)$   
 $\vee (y = \gcd(x^{r/2} - 1, N) \wedge y \neq N \wedge y \neq 1)$   
 $\vee (y = \gcd(x^{r/2} + 1, N) \wedge y \neq N \wedge y \neq 1))\}$  by rule [Cond]  
 18: **od**  
 $\{y|N \wedge y \neq N \wedge y \neq 1\}$  by rule [While] and [Conseq]  
 19: **fi**  
 $\{y|N \wedge y \neq N \wedge y \neq 1\}$  by rule [Cond]

Received XXX; revised XXX; accepted XXX