Mechanizing Feng-Ying Quantum Hoare Logic In Coq for Formal Proofs of Programs with Quantum and Classical Variables

MUSTAFA SAMIR KHALIL, Instituto Superior Técnico, Portugal

Hoare logic is a powerful tool for software reliability. It has been used to prove the correctness of many programs and protocols, covering many types of programs, like deterministic, non-deterministic, recursive, parallel, concurrent... and it is a suitable candidate to provide safe quantum programs, as other ways of testing and debugging could have a high cost (time and hardware resources), or not reliable in the quantum case. An overview of Quantum computing, Quantum Hoare Logic and its mathematical foundations are discussed and compared with some examples and real-world applications of quantum algorithms, and we implemented a mechanization of the logic in Feng and Ying's [1] using Coq The Theorems prover.

CCS Concepts: • **Quantum Computing**; • **Formal Verification**; • **Programming languages**; • **Quantum Hoare Logic**;

1 INTRODUCTION

The development of a new computing paradigm based on Quantum physics, with completely different non-deterministic programming paradigms, brought to light issues, such as probabilities and non-determinism, that challenge our intuition. Developing programs using this approach is different from developing classical programs, and while Quantum computing is still in its infancy, verifying the correctness of Quantum programs is necessary. The traditional ways of guaranteeing safety, like testing and debugging, could have a high cost when applied to Quantum programs, and it might not be the best way to achieve this purpose due to Uncertainty and Non-determinism. A safer alternative is to prove mathematically that the program is correct. For example, the famous computer scientist Tony Hoare proposed, based on the ideas of Robert W. Floyd, a system that formalizes the correctness of programs. This system is known as *Hoare Logic* and it is about determining if some post-condition holds after executing a program starting from given pre-condition. The program, pre-condition, and post-condition form a configuration, called *Hoare triple*, that is usually written as:

$$\{Precondition\}\ Program\ \{Postcondition\}$$

Informally, the meaning of the triple above is: if the $Program$ starts in a state that satisfies the $Precondition$ and if it terminates, it terminates in a state that satisfies the $Postcondition$. The limitations of tests and debugging, together with the increasing importance of Quantum Computing, has encouraged the formal methods research community to propose Hoare logics that can be used to reason about quantum programs. For example, Chadha, Mateus, and Sernadas [2], Yoshihiko Kakutani [3], and Mingsheng Ying [4] developed different variants of Hoare logic for quantum programs. More recently, Feng and Ying extended the qPD logic in [4] to include classical variables, and also extended the definition of a states and assertions to suit the new configuration [1]. The language of their logic is very simple, expressive, and their logic is proven to be sound and complete.

2 OBJECTIVES AND CONTRIBUTIONS

Our main contribution in this thesis is the implementation of a software tool to mechanize proofs for programs with quantum and classical variables, written using a modified model of Feng-Ying Hoare logic [1]. We used the theorems prover *Coq* to model the language, the operational semantics and the proof system in 4.1. We started proving (formally) the soundness and the correctness of the logic, and the correctness of some examples from the real world.

Author's address: Mustafa Samir Khalil, mustafa.samir.khalil@tecnico.ulisboa.pt, Instituto Superior Técnico, Lisboa, Portugal.

3 QUANTUM COMPUTING: AN OVERVIEW

Von Neumann's Formalism of Quantum mechanics [5] associates any quantum system (with a finite degree of freedom) to a Hilbert space with finite dimension. when the space has two dimensions, we call it a qubit. A system could exist in a a pure state when it is sufficient to describe with one vector in the space, or in a mixed state when it is described with many vectors. A qubit $a\,|0\rangle + b\,|1\rangle$ is represented by the vector $\begin{pmatrix} a \\ b \end{pmatrix}$, where

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ and } |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

and $|a|^2, |b|^2$ represent the probabilities of resulting 0, 1 respectively when the qubit is measured.

Example 1. *The state $|+\rangle$ is a superposition between $|0\rangle$ and $|1\rangle$, as follows: $|+\rangle = \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle$ and The state $|-\rangle$ is a superposition between $|0\rangle$ and $|1\rangle$: $|-\rangle = \frac{1}{\sqrt{2}}\,|0\rangle - \frac{1}{\sqrt{2}}\,|1\rangle$*

In a system with many qubits, we cannot describe the state of a qubit separated from others, we call this phenomenon "Entanglement". to represent entangling qubits, we use "Tensor Product" (or Krockecker product) $\otimes$.

Remark 1. *We can write $|00\rangle$ instead $|0\rangle \otimes |0\rangle$ to represent a tensor product of two vectors.*

Example 2. *the state $\frac{1}{\sqrt{2}}\,|00\rangle + \frac{1}{\sqrt{2}}\,|11\rangle$ indicates that qubits are in the same state with equal probabilities.*

Transformations in the system are described with Unitary operators that satisfies the relation $U.U^\dagger = I$, where $U^\dagger = \overline{U}^T$, or the Moore-Penrose inverse.

Example 3. *The Hadamard gate is a unitary transformation, described with the matrix:*

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

where:

$$H\,|0\rangle = |+\rangle \text{ and } H\,|1\rangle = |-\rangle$$

The state of a quantum system could be represented using partial density operators $\mathcal{D}_{\mathcal{H}}$. If the system is in a mixed state ψ_i with a probability p_i, then its density operator is given by:

$$\rho = \sum_i p_i\,|\psi_i\rangle\,\langle\psi_i|$$

Upon applying a unitary transformation U, the density operator of the system updates to $U\rho U^\dagger$. In order to retrieve the state of the system at a time, a measurement could be applied, and it leads to a complete collapse of the system. Upon measurement, if the measurment result were i, then the density operator of the system changes to $\frac{M_i \rho M_i^\dagger}{trace(M_i \rho M_i^\dagger)}$, where M_i is a measurement operator, and the set of all measurment operator forms a projection space to Hilbert space, i.e, it satisfies:

$$\textstyle\sum_{i\in I} M_i {M_i}^\dagger = I_{\mathcal{H}}.$$

Quantum computing is a computational approach with the previously described foundations (initialization, transformations and measurement). Deutsch has suggested the circuit model of quantum programs [6]. In this model, quantum circuits are implemented using wires, gates, controlled gates, and measurement gates. Wires are of two types: quantum wires, which carry qubits, and classical wires for classical bits. A quantum machine is responsible, in particular, for executing the circuit, measuring the results then dispatching it to a classical computer. Knill in [7] has shown that this model is not able to execute some quantum algorithms.

4 FORMAL VERIFICATION OF QUANTUM PROGRAMS

The limitations of testing and debugging with Quantum programs, encourages using static analysis methods to assert the correctness of a quantum program. Hoare Logic, one of those ways, was considered by Chadha, Mateus, and Sernadas [2], Yoshihiko Kakutani [3], and Mingsheng Ying [4] and More recently, Feng and Ying [1] extended the qPD logic in [4] to include classical variables.

4.1 Feng-Ying Hoare Logic

Feng and Ying [1] addressed the problem of handling classical variables along with quantum variables. A simple "While" language was developed over an extension of qPD, and the operational semantics was extended to use a new definition of a *"State"*. Like states, Feng and Ying also redefined assertions to combine variables and quantum statements. A program S according to Feng and Ying [1] is using the following syntax:

$$S ::= \textbf{skip} \mid \textbf{abort} \mid \overline{q} := 0 \mid \overline{q}{*} = U \mid S_1; S_2 \mid x \ := \ e \mid x \ :=_{\$} \ g$$

$$\mid x \ := meas\ \mathcal{M}[\overline{q}] \mid \textbf{while}\ M[\overline{q}]\ \textbf{do}\ S\ \textbf{end}$$

The instruction $x \ :=_{\$} \ g$ means that the classical variable x is assigned to a probabilistic distribution of values g. The following tables compares between Classical, Quantum and Quantum-Classical programs in terms of State and Assertion's definitions and how (or to what degree) a state could satisfy an assertion.

	Classical	**Quantum**	**Quantum-Classical**
State	$\sigma \in \Sigma$	$\rho \in \mathcal{D}_{\mathcal{H}}$	$\Delta \in \Sigma \rightarrow \mathcal{D}(\mathcal{H})$
Assertion	$p \in \Sigma \rightarrow \{0,1\}$	$M \in \mathcal{P}(\mathcal{H})$	$\Theta \in \Sigma \rightarrow \mathcal{P}(\mathcal{H})$
Satisfaction	$\sigma \vert = p$	$tr(M\rho)$	$\sum_{\sigma\in\Sigma} tr(\Delta(\sigma)\Theta(\sigma))$

Table 1. Comparison between different types of languages [1]

Definition 1. *The formula $\{P\}S\{Q\}$ is totally correct, and we write $\models_{tot}$ $\{P\}S\{Q\}$, if for any $V, \Delta : qv(P,S,Q) \nsubseteq V$ and $\Delta \in \mathcal{S}_V$:*

$$Exp(\Delta \models P) \leq Exp([[S]](\Delta) \models Q)$$

where $qv(A)$ is the set of quantum variables of A.

Consider Θ a quantum-classical assertion, and $qVar(\Theta)$ is the set of quantum variables in the assertion Θ. using the definition in 1, Feng and Ying developed a proof system from the following rules:

$$Skip: \ \{\Theta\}skip\{\Theta\}$$

$$Assn: \ \Theta[e/x]\}x := e\{\Theta\}$$

$$Rassn: \sum_{d\in D_{type(x)}} g(d).\Theta[d/x]\}x :=_{\$} g\{\Theta\}$$

$$Init: \frac{q \in qVar(\Theta)}{\{\sum_{i=0}^{d_q-1} |0\rangle_q \langle i| \Theta |i\rangle_q \langle 0|\} q := 0 \{\Theta\}}$$

$$Unit: \frac{\overline{q} \subseteq qVar(\Theta)}{\{U^\dagger \Theta[i/x] U\} \overline{q} = U\overline{q} \{\Theta\}}$$

$$Meas: \frac{q \in qVar(\Theta)}{\{\sum_{i \in J} M_i^\dagger \Theta[i/x] M_i\} x =: meas\ \mathcal{M}[\overline{q}] \{\Theta\}}$$

$$Seq: \frac{\{\Theta\} S_1 \{\Theta'\}\ and\ \{\Theta'\} S_2 \{\Theta''\}}{\{\Theta\} S_1; S_2 \{\Theta''\}}$$

$$While: \frac{\forall m, \{P_m\} c_m \{Q\}}{\{\sum_m M_m^\dagger P M_m\} measure\ M[\overline{q}] : \overline{c} \{Q\}}$$

$$Imp: \frac{\Theta \leq \Theta' \wedge \{\Theta'\} S \{\Psi'\} \wedge \Psi' \leq \Psi}{\{\Theta\} S \{\Psi\}}$$

The term $\Theta[e/x]$ represents a state update where the variable x is substituted with the value e in all occurrences in Θ.
Feng and Ying [1] proved that the system in the previous table is sound and complete with respect to partial and total correctness of quantum classical programs, using the notation of *Weakest Liberal Precondition*, or *wlp* that is inspired from the similar concept for probabilistic programs in [8].

5 MECHANIZATION IN COQ

Our thesis's work was a contribution towards creating a software tool that implements the logic of Feng-Ying in [1], with some changes in Syntax, State and Assertion's definitions.

5.1 Theoretical Part

We slightly changed the language of Feng-Ying to the following syntax:

$S ::= \textbf{skip} \mid x := e \mid x := \textbf{meas}\ n \mid \textbf{new\_qubit} \mid \textbf{q}\ n *= U \mid S_0; S_1 \mid \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_0\ \textbf{end} \mid \textbf{while}\ b\ \textbf{do}\ S\ \textbf{end}$

The following example shows how the syntax could be interpreted:

Example 4.

$$\textbf{\textit{new\_qubit}}; \textbf{\textit{new\_qubit}}; \textbf{\textit{q}}\ 0 *= H; x := \textbf{\textit{meas}}\ 0;$$

$$\textbf{\textit{if}}\ x == 0\ \textbf{\textit{then q}}\ 1 *= X\ \textbf{\textit{else skip end}}; y := \textbf{\textit{meas}}\ 1$$

This program uses two qubits in the $|0\rangle$ state, applies the Hadamard gate H to the first one, then measures it, and if the outcome of the measurement is 0, then it applies the Not gate X to the second qubit, otherwise it skips, then it measures the second qubit, stores the result in y and ends the program.

We decided to use indices to represent qubits, to ease the calculation of Density operators when we reason about assertions with different quantum variables or when we apply gates with two entries like Controlled Not *CNOT*.

In our language, we considered two types of variables, Classical variables with boolean or natural values, hence the domain of those variables is $D = D_{Boolean} \cup D_{Natural}$, and quantum variables, or qubits, the set of quantum variables $\overline{q}$ of a program is associated with a Hilbert Space $\mathcal{H}_{\overline{q}} = \otimes \mathcal{H}_q$, this set is always finite.

5.1.1 *Operational Semantics.*

Definition 2. *the triple* $(Prog_0 \rightarrow Prog_1, \sigma_0 \rightarrow \sigma_1, \rho_0 \rightarrow \rho_1)$ *represents a the transition of the program* $Prog_0$ *to* $Prog_1$ *and the implied change of state.*

Let *dim* be the dimension of the quantum system, i.e, the number of qubits in the system, and let E be the notation for an empty program. the following lemmas illustrate the state updating system of the language. The following rules are state updating rules:

(1) $(\textbf{skip} \rightarrow E, \sigma \rightarrow \sigma, \rho \rightarrow \rho)$
(2) $(x := e \rightarrow E, \sigma \rightarrow \{\sigma, x \rightarrow e\}, \rho \rightarrow \rho)$
(3) $(\textbf{new\_qubit} \rightarrow E, \sigma \rightarrow \sigma, \rho \rightarrow |0\rangle \otimes \rho \otimes \langle 0|)$
(4) $(\textbf{q}\ n *= U \rightarrow E, \sigma \rightarrow \sigma, \rho \rightarrow padding(U, n).\rho.padding(U, n)^\dagger)$
(5) $(x := \textbf{meas}\ n \rightarrow E, \sigma \rightarrow \{\sigma[x/0]\}, \rho \rightarrow M_0(n).\rho.M_0(n)^\dagger)$
(6) $(x := \textbf{meas}\ n \rightarrow E, \sigma \rightarrow \{\sigma[x/1]\}, \rho \rightarrow M_1(n).\rho.M_1(n)^\dagger)$
(7) if $(S_1 \rightarrow S_1', \sigma_1 \rightarrow \sigma_1', \rho_1 \rightarrow \rho_1')$ then $(S1; S2 \rightarrow S1'; S2, \sigma_1 \rightarrow \sigma_1', \rho_1 \rightarrow \rho_1')$.
(8) if $\sigma \models b$ then $((\textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2) \rightarrow S_1, \sigma \rightarrow \sigma, \rho \rightarrow \rho)$, otherwise, then $((\textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2) \rightarrow S_2, \sigma \rightarrow \sigma, \rho \rightarrow \rho)$.
(9) if

$$\sigma \models b$$

then

$$((\textbf{while}\ b\ \textbf{do}\ S\ \textbf{end}) \rightarrow (S; \textbf{while}\ b\ \textbf{do}\ S\ \textbf{end}), \sigma \rightarrow \sigma, \rho \rightarrow \rho)$$

otherwise,

$$((\textbf{while}\ b\ \textbf{do}\ S\ \textbf{end}) \rightarrow E, \sigma \rightarrow \sigma, \rho \rightarrow \rho)$$

.

Where the function *padding* is used to apply the gate U only on the intended qubit, leaving the others unchanged (by applying I), and producing a unitary transformation with the same dimensions of the space to make the multiplication operations in 4. possible. Similarly, $M_0(n)$ and $M_1(n)$ produce matrices with dimensions of $2^{dim} \times 2^{dim}$, and upon applying any of them, let us say $M_0(n)$, then the probability of giving 1 when measuring the n^{th} qubit again is 0, and vice versa for $M_1(n)$.

5.1.2 *Proof System.*

Definition 3. *Let* Δ_{cq} *be a Quantum-Classical state, and let* Ψ_{cq} *be a Quantum-Classical assertion, then we say that* Δ_{cq} *is expected to satisfy* Ψ_{cq} *with a degree:*

$$Exp(\Delta_{cq} \models \Psi_{cq}) = \sum_{\sigma \in \Delta_{cq} \wedge \sigma_1 \models \Psi_{cq}(\sigma_0)} tr(\sigma_2.(\Psi_{cq}(\sigma_1) \otimes I_{\mathcal{H}_V})) \tag{1}$$

Where $\mathcal{H}_V$ *in this context refers to the set of quantum variables in* Ψ_{cq} *but not in* Δ_{cq}.

Definition 4. *The triple*

$$\{P\}Prog\{Q\}$$

denotes a formula to express the correctness of the program Prog, P and Q are Quantum-Classical assertions, we say that $\{P\}Prog\{Q\}$ is totally correct, and we write $\models_{tot}$ $\{P\}Prog\{Q\}$ if

$$\forall \Delta \in \Sigma, Exp(\Delta \models P) \leq Exp(Prog[\Delta] \models Q) \tag{2}$$

and we say that $\{P\}Prog\{Q\}$ is partially correct, and we write $\models_{par}$ $\{P\}Prog\{Q\}$ if

$$\forall \Delta \in \Sigma, Exp(\Delta \models P) \leq Exp(Prog[\Delta] \models Q) + tr(\Delta) - tr(Prog[\Delta]) \tag{3}$$

Where $tr(\Delta) = \sum_{(\sigma,\rho)\in\Delta} tr(\rho)$. The expression $tr(Prog[\Delta]) - tr(\Delta)$ represents the probability of the termination of Prog.

Lemma 1. $\models_{tot}$ $\{P\}Prog\{Q\} \implies \models_{par}$ $\{P\}Prog\{Q\}$

There is no restriction on the sets of quantum variables in *Prog*, P or Q, therefore, they could be different. the statements above are defined for any superset V that contains all those sets.

The following table lists All the Hoare triples for the instructions of the language:

$$(Skip)\frac{}{\{P\}\textbf{skip}\{P\}}$$

$$(Asgn)\frac{}{\{[(\sigma[x/e \to \rho]),\ \forall(\sigma,\rho) \in P]\}x := e\{P\}}$$

$$(Init)\frac{}{\{\sigma, (\langle 0| \otimes I_{2^{dim-1}}).\rho.(|0\rangle \otimes I_{2^{dim-1}})),\ \forall(\sigma,\rho) \in P]\}\textbf{new\_qubit}\{P\}}$$

$$(Unit)\frac{}{\{\sigma, padding(U,n)^{\dagger}.\rho.padding(U,n)) :\ \forall(\sigma,\rho) \in P\}\mathbf{q}\ n{*}=\ U\{P\}}$$

$$(Meas)\frac{}{\{\sigma[x/0], M_0(n)^{\dagger}.\rho.M_0(n)\} \wedge \{\sigma[x/1], M_1(n)^{\dagger}.\rho.M_1(n)\}x\ := \textbf{meas q}\ n\{\sigma,\rho\}}$$

$$(If)\frac{\{b \wedge P\}\mathbf{S1}\{Q\} \wedge \{\neg b \wedge P\}\mathbf{S2}\{Q\}}{\{P\}\textbf{if}\ b\ \textbf{then}\ S1\ \textbf{else}\ S2\ \textbf{end}\ \{Q\}}$$

$$(While)\frac{\{b \wedge P\}\mathbf{S}\{P\}}{\{P\}\textbf{while}\ b\ \textbf{do}\ S\ \textbf{end}\ \{\neg b \wedge P\}}$$

$$(Imp)\frac{P \sqsubseteq P' \wedge \{P'\}\mathbf{S}\{Q'\} \wedge Q' \sqsubseteq Q}{\{P\}S\{Q\}}$$

Example 5. *The following program simulates a coin toss:*

$$\textbf{\textit{new\_qubit}}\ ;\ q * =\ H\ ;\ x\ \textbf{\textit{:=meas}}\ 0$$

We will reason about it (using Operational Semantics and Proofs system), to reach the assertion

$$\Psi = (\ x == 0\ ,\ 1\ )$$

Using Operational semantics:

$$(\textbf{\textit{new\_qubit}}\ ; ...), \{\} \to \{\}, 1 \to |0\rangle\langle 0|$$

$$(\boldsymbol{q}\ \ {}^{*}{=}\ \ \boldsymbol{H}\ ; ...), \{\} \to \{\}, 1 \to H|0\rangle\langle 0|H^{\dagger}$$

$$(x\ \textbf{\textit{:=meas}}\ 0), \{\} \to \{x \to 0\}, 1 \to (|0\rangle\langle 0|)H|0\rangle\langle 0|H^{\dagger}(|0\rangle\langle 0|)^{\dagger}$$

$$(x\ \textbf{\textit{:=meas}}\ 0), \{\} \to \{x \to 1\}, 1 \to (|1\rangle\langle 1|)H|0\rangle\langle 0|H^{\dagger}(|1\rangle\langle 1|)^{\dagger}$$

the final state Δ *is*

$$[\{x \rightarrow 0\} \rightarrow (|0\rangle\langle 0|)H|0\rangle\langle 0|H^\dagger(|0\rangle\langle 0|)^\dagger,$$
$$\{x \rightarrow 1\} \rightarrow (|1\rangle\langle 1|)H|0\rangle\langle 0|H^\dagger(|1\rangle\langle 1|)^\dagger]$$

Then we evaluate the expectation:

$$Exp(\Delta \models \Psi) = trace(((|0\rangle\langle 0|)H|0\rangle\langle 0|H^\dagger(|0\rangle\langle 0|)^\dagger)I) = 0.5$$

Using Proofs System:

$$(\ True\ ,\ 1\ )\ \boldsymbol{new\_qubit}\ (\ True\ ,\ |0\rangle\langle 0|\ )\ Init$$
$$(\ True\ ,\ |0\rangle\langle 0|\ )\ \boldsymbol{q}\ \ {}^*\!\!=\ \ \boldsymbol{H}\ (\ True\ ,\ H|0\rangle\langle 0|H^\dagger\ )\ App$$
$$(\ True\ ,\ H|0\rangle\langle 0|H^\dagger\ )\ x\ \boldsymbol{:=meas}\ 0\ (\ x == 0\ ,\ (|0\rangle\langle 0|)H|0\rangle\langle 0|H^\dagger(|0\rangle\langle 0|)^\dagger\ )\ Meas0$$
$$(\ x == 0\ ,\ (|0\rangle\langle 0|)H|0\rangle\langle 0|H^\dagger(|0\rangle\langle 0|)^\dagger\ )\ \Longrightarrow\ (\ x == 0\ ,\ 0.5|0\rangle\langle 0|\ )\ Weaker$$

5.2 Implementation Part

We chose Coq to implement our project because of the rich documentation (especially in the Software Foundations book [9]), and to reuse the work of QWIRE in [10]. We started with implementing the syntax in 5.1 in the file *Syntax.v*, then we implemented the concepts of Quantum-Classical state and assertion in *State.v* and *Assertion.v*. The operational semantics in 5.1.1 were implemented in *Semantics.v*, and we proved in *Logic* the partial correctness of the Proof system rules in 5.1.2, then we used it to start proving the soundness and completeness of the system in *Soundness.v*. We used a series of intermediate definitions and theorems from QWIRE [10] in *Utils.v* to help in the construction of the project. Some tactics failed to simplify quantum expressions, so we developed a Python script *MatricesConverter.py* to generate Lemmas to make those tactics work. Later, we started proving some examples of quantum algorithms and programs using our System.

The following piece of code is the implementation of the Coin toss in 5:

```
Definition Prog : com :=
  <{ new_qubit;
     q 0 *= GH;
     X :=meas 0
    }>.
```

The assertion in 5 could be defined as follows:

```
Definition post : Assertion 1 := ((_ !-> 0%nat),
        (<{ X == (0 % nat)}>, 0.5 * 0⟩⟨0)).
```

We can prove the Operational semantics version with the following theorem:

```
Theorem operational_sem:
  ceval 0 1 Prog [(( _ !-> 0%nat), I 1)]
  [(( X !-> 0%nat; _ !-> 0%nat), 0.5 * 0⟩⟨0 );
   (( X !-> 1%nat; _ !-> 0%nat), 0.5 * 1⟩⟨1 )].
```

To prove the *post* using the Proof system, first we need to define the pre-condition:

```
Definition pre : Assertion 0 := ((_ !-> 0%nat), (BTrue, I 1)).
```

Then we prove the theorem:

```
Theorem prog_correct: hoare_triple pre Prog post.
```

To prove the previous theorem, we use the theorems of *Logic.v*:

```
Theorem fy_skip: forall n (P: Assertion n),
   hoare_triple P <{skip}> P.

Theorem fy_sequence: forall np nq nr (P: Assertion np)
  (Q: Assertion nq) (R: Assertion nr) c1 c2,
    hoare_triple P c1 Q ->
    hoare_triple Q c2 R ->
    hoare_triple P <{ c1;c2 }> R.

Theorem fy_assign: forall n x e (P: Assertion n),
    hoare_triple (asgn_sub P x e) <{ x := e }> P.

Theorem fy_if: forall (n: nat) (b: bool_exp) (c1 c2: com)
    (P Q: Assertion n),
    hoare_triple (pre_if_assertion_boolean P b) c1 Q ->
    hoare_triple (pre_if_assertion_boolean P (BNot b)) c2 Q ->
    hoare_triple P <{ if b then c1 else c2 end }> Q.

Theorem fy_init: forall n (P: Assertion n),
    hoare_triple (init_sub n P) <{ new_qubit }> P.

Theorem fy_apply: forall n m G (P: Assertion n),
    hoare_triple (apply_sub n m (geval G) P) <{ q m *= G }> P.

Theorem fy_measure: forall n x m (P: Assertion n),
    hoare_triple (meas_sub P x 1%nat m) <{ x :=meas m }> P /\
    hoare_triple (meas_sub P x 0%nat m) <{ x :=meas m }> P.

Theorem fy_while: forall n b c (P: Assertion n),
    hoare_triple (pre_if_assertion_boolean P b) c P ->
    hoare_triple P <{ while b do c end }>  (pre_if_assertion_boolean P (BNot b)).

Theorem fy_weakness: forall n c (P Q P' Q': Assertion n),
    hoare_triple P c Q ->
    (forall ns (st: State ns)),
           weaker ns n n st P' P) ->
    (forall ns (st: State ns)),
          weaker ns n n st Q Q') ->
    hoare_triple P' c Q'.

Theorem fy_imp_pre: forall n c (P Q P': Assertion n),
    hoare_triple P c Q ->
    (DensityOf P) = (DensityOf P') ->
    classicalPropsImp n n P' P ->
    hoare_triple P' c Q.

Theorem fy_imp_post: forall n c (P Q Q': Assertion n),
    hoare_triple P c Q ->
    (DensityOf Q) = (DensityOf Q') ->
    classicalPropsImp n n Q Q' ->
    hoare_triple P c Q'.
```

6 CONCLUSIONS

We presented a contribution towards facilitating the development verifiable quantum programs with quantum and classical variables, we also implemented the concepts in [1], and started formally proving the soundness and completeness of the logic in 4.1. For the future, we have a clear road-map to continue developing this project towards making it easier to be used by developers.

7 FUTURE WORK

For the future, The first priority is to address some limitations, like the following:

- The inability to reason about assertions with Real numbers: Our arithmetic expressions contains only boolean and natural values, and it was not possible to extend the expressions to contain real values, due to the axiomatic definition of Real numbers, which implied the impossibility to cast Real numbers comparison relations to boolean.
- Proving some programs took too many lines and longer time than desired.
- Lack of Expressiveness: our current syntax does not allow to initialize many qubits at once, measuring many qubits, and applying gates on multiple qubits. Also, it does not allow to apply *CNOT* gates on nonadjacent qubits, or even on qubits with reverse order.
- We were not able to complete the proofs for the *While* rule, nor the soundness and completeness of the system.
- Some intermediate theorems are unproven (formally): to facilitate proving the main theorems, we used some intermediate theorems. Those theorems were left unproven (with the keyword *Admitted* in Coq). The reason was that Some of the tactics that we were using to simplify quantum expressions were failing. To solve this issue, we developed a script in Python to do this job, the script converts any quantum expression into a matricial form, which allows using the *lma* tactic from QWIRE [10].

Later, the project could be enhanced with:

- Finishing the formal proof of Soundness/Completeness of the logic.

- Implementing the discarding mechanism: to avoid the use of measured quantum variables.
- Adding more arithmetic, boolean and quantum expressions: to facilitate the implementation of some quantum algorithms.
- Automating the proofs: Proving some of the programs using the proof system and our tool took relatively long time and many lines of code, which could jeopardise the objective of our work to make assuring programs' correctness easier for developers than waiting in line for execution in some cloud quantum service. Therefore, we suggest developing our tool to automate proofs, or at least making the process easier for the developer.
- Interoperability with External Quantum Computation Platforms

REFERENCES

[1] Feng, Y.; Ying, M. Quantum Hoare logic with classical variables. *arXiv preprint arXiv:2008.06812* **2020**,

[2] Chadha, R.; Mateus, P.; Sernadas, A. Reasoning about imperative quantum programs. *Electronic Notes in Theoretical Computer Science* **2006**, *158*, 19–39.

[3] Kakutani, Y. A logic for formal verification of quantum programs. Annual Asian Computing Science Conference. 2009; pp 79–93.

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

[5] Von Neumann, J. *Mathematical foundations of quantum mechanics*; Princeton university press, 2018.

[6] Deutsch, D. E. Quantum computational networks. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* **1989**, *425*, 73–90.

[7] Knill, E. *Conventions for quantum pseudocode*; 1996.

[8] Morgan, C.; McIver, A.; Seidel, K. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems (TOPLAS)* **1996**, *18*, 325–353.

[9] Pierce, B. C.; de Amorim, A. A.; Casinghino, C.; Gaboardi, M.; Greenberg, M.; Hriţcu, C.; Sjöberg, V.; Yorgey, B. In *Logical Foundations*; Pierce, B. C., Ed.; Software Foundations; Electronic textbook, 2021; Vol. 1; Version 6.1, `http://softwarefoundations.cis.upenn.edu}\relax`\mciteBstWouldAddEndPuncttrue \mciteSetBstMidEndSepPunct.

[10] Paykin, J.; Rand, R.; Zdancewic, S. QWIRE: a core language for quantum circuits. *ACM SIGPLAN Notices* **2017**, *52*, 846–858.