TÉCNICO LISBOA

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

Mustafa Samir Khalil

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Dr. João Fernando Peixoto Ferreira

Examination Committee

Chairperson: Prof. Dr. José Luís Brinquete Borbinha
Supervisor: Prof. Dr. João Fernando Peixoto Ferreira
Member of the Committee: Prof. Dr. Luís Soares Barbosa

October 2021

Acknowledgments

I would like to thank the Global Platform for Syrian Students for their generous support for me to finish my studies. especially Dr. Helena Barroco, I would also like to dedicate this work to the memory of Dr. Jorge Sampaio, the former president of the Portuguese republic and the founder of the Global Platform for Syrian Students. I would like to also to thank my parents for being there for me all this time, and my brothers Dr. Ali and Dr. Ehab for their friendship. My appreciation goes to my supervisor, Dr. João Fernando Ferreira for his support in this work along this year, and to Dr. Professor Luís Soares Barbosa for his valuable comments to earlier versions of this work. Last but not least, to all my professors, friends and colleagues that helped me grow as a person and were always there for me during the good and bad times in my life.

Abstract

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. In this thesis, we present 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 [1] using Coq The Theorems prover.

Keywords

Quantum Computing, Formal Verification, Quantum Hoare Logic and Programming languages

Resumo

A lógica de Hoare é uma ferramenta poderosa para a verificação formal de software. Tem sido usado para provar a correção de muitos programas e protocolos, com varios tipos de programas, como determinísticos, não determinísticos, recursivos, paralelos, concorrentes ... e é um candidato adequado para asegurar a correção dos programas quânticos, como outras formas de teste e depuração tem um custo alto, ou não são confiáveis, no caso quântico. Neste tese vamos fazer uma curta introdução para computação quântica, Quantum Hoare Logic e seus fundamentos matemáticos que serão discutidos e comparados com alguns exemplos e aplicações do mundo real de algoritmos quânticos, e vamos implementar uma mecanização da lógica em Feng et al. [1] usando o provador de teoremas Coq.

Palavras Chave

Computação quântica, verificação formal, lógica Quantum Hoare e linguagens de programação

Contents

List of Figures

List of Tables

Listings

1

Introduction

Contents

Although quantum computing has not been yet used to solve problems that could not be solved with a feasible complexity using classical computing, it has been constantly advancing in the last decades. The increasing interest in quantum computing highlights the necessity to guarantee the correctness of quantum programs. It has been shown that traditional ways of testing and debugging software are not adequate to be used with quantum programs, due to an interesting feature of quantum systems, which is: *measuring a system state destroys it, which leads to aborting the computational process* [2]. An alternative, and much stronger, way to guarantee correctness is to use formal methods that guarantee that quantum programs achieve what they are supposed to. However, the notion of correctness can pose some challenges.

1.1 The Challenge of Correctness

Our human intuition and perception can handle, relatively easily, the principles of classical physics, like Newtonian mechanics, because of its deterministic nature that can trace natural phenomena back to their root causes. Since the dawn of modern computing in 1946 with the development of the first programmable electronic computer, ENIAC, writing (and verifying) algorithms was not trivial, but the ideas were not very hard to understand, as they were based on classical physics. Later on, 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.

Programmers still face some challenges with non-deterministic programming, mainly due to the difficulty to reason about those programs. Reasoning about non-deterministic programs may involve reasoning about probability distributions, in which case Bayesian networks could be helpful to reason about the probabilities distribution of the program state. Programmers can also use some conventional software engineering testing techniques to gain confidence in that a program that they wrote is doing what is meant to be doing: useful techniques include unit testing, integration testing (when several components are involved), and debugging. For example, it is possible to write a test, running it multiple times, and then measure the likelihood that the probability distribution of the results coincide with the desired one. However, those techniques might not be the best criteria to judge if a non-deterministic program is correct, since testing may cover only a limited space of cases.

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 $Postcondition$. For example: the triple $\{x = 1\}x := x + 1\{x = 2\}$ is valid, but the triple $\{x < 1\}x := x + 1\{x > 2\}$ is invalid. A Hoare logic for probabilistic programs was first proposed Lyle Ramshaw in [3]. In this work, the triple

$$\{P[x < 1] = \frac{1}{3})\}\ x := x + 1\ \{P[x < 2] = \frac{1}{3}\}$$

is read as "if x is less than 1 with a probability of $\frac{1}{3}$, then by adding 1 to x, it will be less than 2 with the same probability". Probabilistic programs extend conventional imperative programs with probabilistic instructions, like probabilistic assignment, which can manipulate the probability distributions. For instance, the assignment $x := (0 : \frac{1}{2}, 2 : \frac{1}{4}, -2 : \frac{1}{4})$ assigns x to 0 with a probability of 0.5, to 2 with a probability of 0.25, and to -2 with a probability of 0.25. Considering this assignment, the following triple holds:

$$\{P[true] = 1\}\ x := (0 : \frac{1}{2}, 2 : \frac{1}{4}, -2 : \frac{1}{4})\ \{P[x <= 0] = \frac{3}{4}\}$$

As the statement $x \leq 0$ holds for $x = 0$ or $x = -2$, hence

$$P[x <= 0] = P[x = 0] + P[x = -2] = \frac{1}{2} + \frac{1}{4} = \frac{3}{4}$$

1.2 Formal Verification of Quantum Programs

In addition to the uncertainty of how useful using tests and debugging could be to assure the correctness of a quantum program, the cost for such methods is high, at least in the meantime, where we live in the Noisy intermediate-scale quantum (NISQ) era, [4]. This means that the current quantum computation power can put between 50 to 300 qubits all together to execute a process, but without guaranteeing fault-tolerance or exceeding the performance of classical computers.

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 [5], Yoshihiko Kakutani [6], and Mingsheng Ying [7] developed different variants of Hoare logic for quantum programs. From those variants, Chadha, Mateus, and Sernadas's EEQPL is the least expressive and, even though it is proven to be sound, its completeness is still unknown (in contrast, Ying's qPD was proven to be

sound and relatively complete). However, EEQPL is the only variant that considers quantum and classical variables together in a program. This is useful because it could save quantum resources from representing those variables, and it satisfies the quantum circuit model, suggested by Deutsch [8]. More recently, Feng and Ying extended the qPD logic 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, similarly to qPD, it is proven to be sound and relatively complete. As future work, Feng and Ying suggested mechanizing this logic using a sofware tool. In this project, we propose to contribute towards this direction.

1.3 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 Section 4.6. We started proving (formally) the soundness and the correctness of the logic, and the correctness of some examples from the real world.

1.4 Organization of the Document

This thesis is organized as follows. In Chapter 2, we present a brief introduction to Quantum computing, mathematical preliminaries, some quantum algorithms, and some useful tools to develop quantum programs. In Chapter 3, we present a survey of Hoare logics that are used to reason about quantum programs, and a comparison between them in expressiveness, soundness and completeness. In Chapter 4 we show our adapted model from Feng-Ying Language's syntax, semantics and proof system. Chapter 5 contains our contributions, mainly the mechanization of the logic in Chapter 4. Chapter 6, we list the problems with our solution, and some suggestions for future to improve it.

2

Quantum Computation: An overview

Contents

The complexity of Quantum mechanics has introduced a new non-deterministic approach to writing programs, in a different way from what we used to do with Classical computers. The concept of quantum programs relies basically on two fundamental phenomenons, Superposition and Entanglement.

2.1 A Brief History of Quantum Computing

The main idea behind Quantum is exploiting the superposition of quantum-entangled information units, or qubits [9]. Quantum entanglement [10] happens when more than one particle is formed in a shared space with a small distance in such a manner that the quantum state of each particle of the group must be described along with the states of the rest of the particles, even the most distant ones [11]. In 1934, the EPR paper by Albert Einstein, Boris Podolsky, and Nathan Rosen [12] concluded the incompleteness of the Quantum-mechanical description of physical reality given by wave functions. Later, Erwin Schrödinger, In his response to "EPR paradox" paper, used the word "entanglement" (*Verschränkung* in German) to describe the correlations between two particles that interact and then separate. Einstein and Schrödinger were both not completely along with the concept of entanglement, as it violates the theory of relativity, More specifically, the Information transfer speed limit. In 1964, John Stewart Bell argued the principle of the locality, one of the key principles of EPR paradox was mathematically inconsistent with the predictions of quantum theory using Bell's Inequality in [13]. Bell's work (namely Bell's Inequality) was used later in 1984 to develop the "Quantum key distribution Protocol" by Erkert [11] to prove its security.

In 1994, Peter Shor from MIT's Bell labs published the well-known quantum Natural Numbers Factoring algorithm [14] that provides an exponential acceleration in finding the factors of a natural number compared to the most effective classical factoring algorithm, exposing the famous RSA encryption protocol to danger. Quantum computers are composed of circuits. A circuit in a Quantum computer is mainly composed of qubits (quantum bits), and quantum gates. Quantum gates can perform operations on entangled qubits and change its state. Quantum algorithms are constructed by interconnecting and manipulating qubits and Qauntum gates to achieve a defined goal. To perform reliable operations on qubits, they need to be isolated from any external influence long enough to execute the whole program besides the error correction operations before collapsing, which is a real challenge to offer. To improve the efficiency of quantum computers, we could make the gates faster, the error correction mechanisms more effective, and increase the numbers of gates that complete their operation correctly.

When a quantum computer can solve a problem that no classical computer could solve in any feasible time (regardless of the type or usefulness of the problem), we say that "Quantum supremacy" is reached. In 2019, Sycamore, a processor with 53 programmable superconducting qubits, managed to solve a problem with temporal complexity of 10 thousand years in 200 seconds [15]. In December 2020, a

team of researchers in Beijing National Research Center claimed that their system, Jiuzhang, produced results showing that their quantum computer performed, in minutes, calculations that could take more than 2 billion years of effort by a powerful supercomputer [1].

2.2 Mathematical Preliminaries

In this section, we introduce the mathematical preliminaries required to support the remainder of the work.

2.2.1 Hilbert space

Complex numbers play an essential role in quantum computing, specifically in representing the state of a quantum system. The set of complex numbers is denoted by $\mathbb{C}$ and defined as:

$$\mathbb{C} = \{z : z = a + bi,\ a, b\ \in \mathbb{R}\}$$

The state of a quantum system is represented using Hilbert Spaces.

Definition 1. *A Hilbert Space $\mathcal{H}$ is a complex vector space with two operations:*

1. *Addition:* $+ : \mathcal{H} \times \mathcal{H} \to \mathcal{H}$
2. *Scalar Multiplication:* $. : \mathcal{H} \times \mathbb{C} \to \mathcal{H}$

We will use the bra-ket notation to represent states in a Hilbert space. In more detail, it represents a linear mapping $|.\rangle : S \to \mathbb{C}$ where S is a complex plane, it was used by Paul Dirac in 1939 [16] to facilitate the calculations in Quantum physics. Hilbert spaces satisfy the following properties:

1. $(\mathcal{H}, +)$ is an Abelian group with a zero element, that is the zero vector.
2. $1. |\psi\rangle = |\psi\rangle$
3. $\lambda(\mu |\psi\rangle) = \lambda\mu |\psi\rangle$
4. $(\lambda + \mu) |\psi\rangle = \lambda |\psi\rangle + \mu |\psi\rangle$
5. $(|\phi\rangle + |\psi\rangle)\lambda = \lambda |\psi\rangle + \lambda |\phi\rangle$
6. $(|\phi\rangle + |\psi\rangle) + |\alpha\rangle = |\phi\rangle + (|\psi\rangle + |\alpha\rangle)$

Hilbert spaces are also provided with an inner product, defined as follows:

[1] https://www.sciencenews.org/article/new-light-based-quantum-computer-jiuzhang-supremacy

Definition 2. *The inner product* $\langle .|. \rangle : \mathcal{H} \times \mathcal{H} \to \mathbb{C}$ *is a map defined as:*

1. $\langle \psi|\psi \rangle \geq 0$
2. $\langle \psi|\psi \rangle = 0$ *if and only if* $|\psi\rangle = 0$
3. $\langle \psi|\phi \rangle = \overline{\langle \phi|\psi \rangle}$*, where* $\overline{a}$ *stands for complex conjugate of* a*.*
4. $\langle \psi|(\lambda_1 |\phi_2\rangle + \lambda_2 |\phi_2\rangle)\rangle = \lambda_1 \langle \psi|\phi_2\rangle + \lambda_2 \langle \psi|\phi_2\rangle$

For any $\psi, \phi, \phi_1, \phi_2 \in \mathcal{H}$ *and* $\lambda_1, \lambda_2 \in \mathbb{C}$

Using this definition, we also define the norm of $|\psi\rangle$, denoted by $||\psi||$, as:

$$||\psi|| = \sqrt{\langle \psi|\psi \rangle}$$

The vector ψ is called a unit vector if $||\psi|| = 1$. And also, we say that two vectors ψ and ϕ are orthogonal if $\langle \psi|\phi \rangle = 0$.

Example 1. *Let* $|\psi\rangle = (\frac{1}{\sqrt{3}} \ \ \frac{\sqrt{2}}{\sqrt{3}})^T$*; then* $\langle \psi|\psi \rangle = \frac{1}{3} + \frac{2}{3} = 1$*.*

Definition 3. *A family* $\{|\psi_i\rangle\}_{i \in I}$ *of unit vectors forms an orthonormal basis of* $\mathcal{H}$ *iff*

1. $\forall i, j, i \neq j$*:* ψ_i, ψ_j *are orthogonals.*
2. $|\psi\rangle = \sum_{i \in I} \langle \psi_i|\psi \rangle |\psi_i\rangle$ *for all* $|\psi\rangle \in \mathcal{H}$

A Hilbert space is defined as a complete inner product space. i.e, an inner product space in which each Cauchy sequence of vectors has a limit.

A pure state of the system is described by a unit vector in its state subspace. For example, the state space of a system of one qubit is a 2-dimensional Hilbert Space of the form

$$\mathcal{H}_2 = \{a|0\rangle + b|1\rangle : a, b \in \mathbb{C}\}$$

and the 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. Note that $|a|^2 = a.\overline{a}$ is the norm of the complex number a.

Remark 1. *It is common to write* $|+\rangle$ *instead of* $\frac{|0\rangle + |1\rangle}{\sqrt{2}}$ *and* $|-\rangle$ *instead of* $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$

2.2.2 Tensor Product

Tensor product is used to represent the state space of a multi-qubit system.

Definition 4. *The Tensor Product of two Hilbert Spaces is a Hilbert Space together with a bilinear mapping* $\phi : \mathcal{H}_1 \times \mathcal{H}_2 \to \mathcal{H}_1 \otimes \mathcal{H}_2$ *such that*

1. *The set of all vectors* $\phi(x, y) : x \in \mathcal{H}_1$ *and* $y \in \mathcal{H}_2$ *is a subset of* $\mathcal{H}_1 \otimes \mathcal{H}_2$.
2. $\langle x_1 \otimes x_2, y_1 \otimes y_2 \rangle_{\mathcal{H}_1 \otimes \mathcal{H}_2} = \langle x_1, y_1 \rangle_{\mathcal{H}_1} \langle x_2, y_2 \rangle_{\mathcal{H}_2}$.

Lemma 1. *The following assertions are true:*

1. *If* $\mathcal{H}_1$ *has the orthonormal basis* $\{\psi_i\}_{i \leq n}$ *and* $\mathcal{H}_2$ *has the orthonormal basis* $\{\psi_j\}_{j \leq m}$ *then* $\mathcal{H}_1 \otimes \mathcal{H}_2$ *has the basis* $\{\psi_j \otimes \psi_j\}_{i \leq n \ \& \ j \leq m}$.
2. *We can generalize the statements above over a tensor product of multiple Hilbert Spaces. Let* $\mathcal{H}_i$ *be one of those spaces with the orthonormal basis* $\{\psi_{ij_i}\}_{j \leq n_i}$*; then the set* $\mathcal{B}$ *of tensor products of all* $\mathcal{H}_i$ *is*

$$\mathcal{B} = \{\otimes_i |\psi_{ij_i}\rangle\}$$

2.2.3 Density Operators

An operator is a rule that can be applied to a function to transform it into another function, or to transform a vector to another vector in the same space. For instance, the derivation of a function D is an operator. *Observable* are variables in quantum mechanics, like position, momentum, angular momentum, and energy, where its measurement characterizes the quantum state of a particle. One of quantum mechanics' postulates states that there is an operator that corresponds to each physical observable.

Definition 5. *A linear operator* A *on a Hilbert Space* $\mathcal{H}$ *is a mapping* $\mathcal{H} \to \mathcal{H}$ *that satisfies:*

1. $A(|\psi\rangle + |\phi\rangle) = A|\psi\rangle + A|\phi\rangle$
2. $A(\lambda|\psi\rangle) = A\lambda|\psi\rangle$

for all $|\psi\rangle, |\phi\rangle \in \mathcal{H}$ *and* $\lambda \in \mathbb{C}$.

if $\{|\psi_i\rangle\}_i$ forms an orthonormal basis of $\mathcal{H}$ then the operator A can be written as a matrix: $A = (\langle\psi_i| A |\psi_j\rangle)_{ij}$ when $\mathcal{H}$ is finite-dimensional.

Definition 6. *We say that an operator* A *is bounded if for a constant* $C > 0$ *and all* $|\psi\rangle$ *the following inequality is satisfied:*

$$||A|\psi\rangle|| \leq C||\psi||$$

We write $\mathcal{L}(\mathcal{H})$ to denote all the bounded operators over $\mathcal{H}$

Definition 7. *The adjoint of an operator A, denoted as $A^\dagger$, is the only operator that satisfies*

$$\langle A|\psi\rangle, |\phi\rangle\rangle = \langle|\psi\rangle, A^\dagger|\phi\rangle\rangle$$

Definition 8. *An operator is called **Hermitian** if and only if $A = A^\dagger$.*

Definition 9. *An operator A is called **positive** if it is a linear operator that satisfies $\langle\psi|A|\psi\rangle \geq 0$ for all states $|\psi\rangle \in \mathcal{H}$.*

Lemma 2. *All positive operators are Hermetian.*

Proof. let A be a positive operator, or, $\forall\psi : \langle\psi|A|\psi\rangle \geq 0$, We can write A in the form: $A = Re(A) + iIm(A)$, where $Re(A)$ and $Im(A)$ are the real and imaginary parts of A, we can also write:

$$Re(A)^\dagger = (\frac{A + A^\dagger}{2})^\dagger = \frac{A^\dagger + A}{2} = Re(A)$$
$$Im(A)^\dagger = (\frac{A - A^\dagger}{2i})^\dagger = \frac{A^\dagger - A}{-2i} = \frac{A - A^\dagger}{2i} = Im(A)$$

For $Im(A)$, as it is Hermetian and real, then it is diagonalizable, or, it could be written in the from

$$Im(A) = \Phi.\Lambda.\Phi^\dagger = \sum_i |\Phi_i\rangle\langle\Phi_i|\lambda_i$$

where $\{\Phi_i\}_i$ is an orthonomal basis of the space, let $|x\rangle$ be a vector from this space, then we can write

$$|x\rangle = \sum_i x_i|\Phi_i\rangle$$

We will prove that $\langle x|Im(A)|x\rangle$ is a real number, which is followed from:

$$\begin{aligned}\langle x|Im(A)|x\rangle &= (\sum_i \overline{x_i}\langle\Phi_i|)(\sum_i \lambda_i|\Phi_i\rangle\langle\Phi_i|)(\sum_i x_i|\Phi_i\rangle) \\ &= \sum_i\sum_j \lambda_i x_i\overline{x_j}\langle\Phi_j|\Phi_i\rangle \\ &= \sum_i \lambda_i x_i\overline{x_i} \\ &= \sum_i \lambda_i||x_i||^2\end{aligned}$$

Following the same way we prove that $\langle x| Re(A) |x\rangle$ is also real, and since $\langle x| A |x\rangle \geq 0$, which is also written as $\langle x| (Re(A) + iIm(A)) |x\rangle = \langle x| Re(A) |x\rangle + i \langle x| Im(A) |x\rangle \geq 0$, the part $\langle x| Im(A) |x\rangle$ could only be a zero (it could not be pure imaginary as it was proven to be real), then $A = Re(A)$ which is Hermetian. □

Definition 10. *An operator A is called s trace class operator if the sequence*

$$\{\langle \psi_i| A |\psi_i\rangle\}_{i\in I}$$

is summable , i.e, $\sum_{i\in I} \langle \psi_i| A |\psi_i\rangle < \infty$ for any orthonormal basis $\{|\psi_i\rangle\}$ of $\mathcal{H}$.

Definition 11. *The trace of a trace class operator A is given by*

$$tr(A) = \sum_{i\in I} \langle \psi_i| A |\psi_i\rangle$$

Lemma 3. *$tr(A)$ is independent of the choice of the orthonormal basis, or, if $\{|\psi_i\rangle\}$ were an orthonormal basis, and $\{|\phi_i\rangle\}$ were an orthonormal basis too, then*

$$tr(A) = \sum_{i\in I} \langle \psi_i| A |\psi_i\rangle = \sum_{i\in I} \langle \phi_i| A |\phi_i\rangle$$

Definition 12. *A Quantum system could exist in a mixed state, or, when the system is observed, the result could be $|q_i\rangle$ with a probability p_i. The operator*

$$\mathcal{P} = \sum_i p_i |q_i\rangle \langle q_i|$$

is called a Density Operator.

2.2.4 Unitary Transformation

Unitary Transformations model operations done via *quantum gates* on qubits to manipulate the probabilistic distribution of measurement outcomes, in order to reach a desirable result out of the circuit that contains them.

Definition 13. *An operator U on $\mathcal{H}$ is called a unitary transformation if*

$$U^\dagger U = I_{\mathcal{H}}$$

where $U^\dagger = \overline{U}^T$ is the Moore-Penrose inverse, and $I_{\mathcal{H}}$ is the identity operator on H.

Lemma 4. *The following two assertions are true:*

1. *If a quantum system is in two states $|\psi_0\rangle$ and $|\psi_1\rangle$ at two times t_0, t_1 then the two states are related with a unitary transformation, i.e, $|\psi_0\rangle = U |\psi_1\rangle$ where U is a unitary transformation.*

2. *Similarly, if a quantum system is in two density operators ρ_0 and ρ_1 at two times t_0, t_1 then $\rho_1 = U\rho_0 U^\dagger$.*

2.2.5 Superposition and Entanglement

We say that a qubit q is in a superposition state when we can represent its state as linear combination of the basis of a Hilbert space, i.e, $q = \alpha |0\rangle + \beta |1\rangle$.

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

In a system with many qubits, if we cannot describe the state of a qubit separated from others, we call this phenomenon "Entanglement".

Example 3. *the state $\frac{1}{\sqrt{2}} |00\rangle + \frac{1}{\sqrt{2}} |11\rangle$ indicates that qubits , when observed, will be in the same state with equal probabilities.*

2.2.6 Measurement

Once the state of a quantum system is measured once, it is going to be destroyed. Let the possible outcomes of a measurement be $\{m_i\}_{i \in I}$. Then for every outcome, there is an operator M_i called the measurement operator, satisfying:

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

Lemma 5. *The following assertions are true:*

1. *If the system was in a pure state $|\psi\rangle$, then the probability of the outcome m_i appearing is $p(m_i) = \langle\psi| M_i {M_i}^\dagger |\psi\rangle$.*
2. *If the system state was described with density operators, then if the state before the measurement was ρ then the probability of the outcome m_i is $p(m_i) = tr({M_i}^\dagger M_i \rho)$ and the state after the measurement will be $\rho_i = \frac{M_i \rho {M_i}^\dagger}{\sqrt{p(m_i)}}$.*
3. *The possible outcomes of one qubit are $\{m_0, m_1\}$.*

Example 4. *In a system of one qubit, assume that the measurement is done on basis $|0\rangle, |1\rangle$, then $M_0 = |0\rangle\langle 0|$ and $M_1 = |1\rangle\langle 1|$ are measurement operators for the system, because:*

$$\begin{aligned} M_0.M_0^\dagger + M_1.M_1^\dagger &= |0\rangle\langle 0| \left(|0\rangle\langle 0|\right)^\dagger + |1\rangle\langle 1| \left(|1\rangle\langle 1|\right)^\dagger \\ &= \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = I \end{aligned}$$

2.3 Quantum Circuit Model

Deutsch has suggested the circuit model of quantum programs [8]. 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 [17] has shown that this model is not able to execute some quantum algorithms.

2.3.1 Quantum gates

Quantum gates are Unitary transformations that take qubits as inputs and output the resulted quantum state. To help visually understand those transformations, we will show them on Bloch Sphere. The most common quantum gates are:

1. Hadamard Gate **H**: a single-qubit gate that maps $|0\rangle$ to $\frac{|0\rangle+|1\rangle}{2}$ or $|+\rangle$ and $|1\rangle$ to $\frac{|0\rangle-|1\rangle}{2}$ or $|-\rangle$.

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

   It is easy to prove that H is unitary.

2. Pauli Gates: three single-qubit gates, **X**, **Y** and **Z**.

   (a) X-Gate: works as the logical Not-Gate, as it maps $|0\rangle$ to $|1\rangle$ and $|1\rangle$ to $|0\rangle$. Its matrix is

$$X = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$$

   (b) Y-Gate: It maps $|0\rangle$ to $i\,|1\rangle$ and $|1\rangle$ to $-i\,|0\rangle$ and its matrix is

$$Y = \begin{pmatrix}0 & -i\\ i & 0\end{pmatrix}$$

   (c) Z-Gate: It does not change $|0\rangle$ but maps $|1\rangle$ to $-\,|1\rangle$ and its matrix is

$$Z = \begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}$$

3. Phase Shift gates: single-qubit of the form R_θ, where θ is the rotation angle; they do not change $|0\rangle$ but map $|1\rangle$ to $e^{i\theta}\,|1\rangle\,t$. Its matricial form is

$$R_\theta = \begin{pmatrix}1 & 0\\ 0 & e^{i\theta}\end{pmatrix}$$

4. Controlled Gates: these act on two or more qubits and there are many examples of such gates:

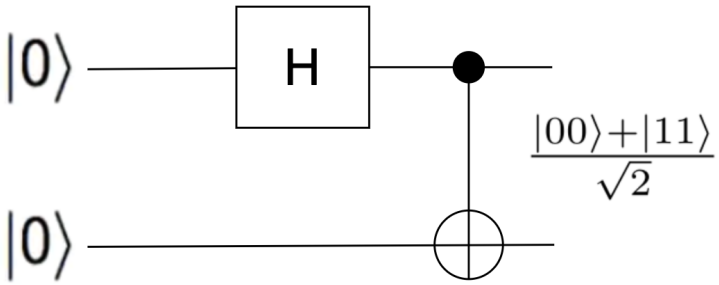

Figure 2.1: Bell State

(a) Controlled Not Gate: it maps $|a\rangle |b\rangle$ (or $|ab\rangle$) to $|a, a \oplus b\rangle$, we call $|a\rangle$ the control qubit. The matrix (for two qubits in basis $\{|0\rangle, |1\rangle\}$) is

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

(b) Swap Gate : it swaps two qubits, i.e., it maps $|xy\rangle$ to $|yx\rangle$. The matrix (for two qubits in basis $\{|0\rangle, |1\rangle\}$) is

$$SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(c) Controlled U Gate: applies the unitary transformation U on the second qubit if the value of the first qubit is $|1\rangle$. The matrix (for two qubits in basis $\{|0\rangle, |1\rangle\}$) is

$$cU = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{11} & u_{12} \\ 0 & 0 & u_{21} & u_{22} \end{pmatrix}$$

that maps $|xy\rangle$ to $|x, x \oplus Uy\rangle$

Gates can be composed in circuits. For instance, Figure Figure 2.1 shows the Bell state, which is a very important component of many algorithms that can be written as transition:

$$|00\rangle \rightarrow (\tfrac{|0\rangle+|1\rangle}{\sqrt{2}}) |0\rangle = \tfrac{1}{\sqrt{2}}(|00\rangle + |10\rangle) \rightarrow \tfrac{1}{\sqrt{2}}(|0(0 \otimes 0)\rangle + |1(1 \otimes 0)\rangle) \rightarrow \tfrac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

The final state is referred as the EPR state (Einstein, Podolsky and Rosen state).

2.3.2 Quantum Oracles

A quantum oracle O is a black box operation that is performed over two or more qubits and executes a classical function $f : \{0,1\}^n \to \{0,1\}^m$. For all $x \in \{0,1\}^n$ and $y \in \{0,1\}^m$ we have

$$O(|x\rangle \otimes |y\rangle) = |x\rangle \otimes |y \oplus f(x)\rangle$$

Oracles play an essential role in the Deutsch-Jozsa Algorithm as well as in the Grover Algorithm, as shown in the following sections.

2.4 Some Quantum Algorithms

In this section, we list some of the most famous quantum algorithms that solve some real-world problems, and we discuss the correctness of some of them.

2.4.1 Overview

Programs that take advantage of quantum properties like entanglement, superposition, unitary transformations and measurement, can provide a polynomial or an exponential boosting to solve some problems. For instance, Grover's algorithm improves searching in an unstructured dataset quadratically (it finds the solution in $O(\sqrt{N})$ iterations). In this section, we show some examples of quantum algorithms and their temporal complexity.

2.4.2 Quantum Teleportation

It is not possible to copy the state of a qubit at a time, as stated by the theorem of no-cloning:

Lemma 6. *There is no operator U that satisfies the predicate: $U|a\rangle|0\rangle = |a\rangle|a\rangle$.*

Proof. We write $|a\rangle = \alpha|0\rangle + \beta|1\rangle$, then

$$|a\rangle|0\rangle = \alpha|0\rangle|0\rangle + \beta|1\rangle|0\rangle$$

and

$$|a\rangle|a\rangle = \alpha^2|0\rangle|0\rangle + \alpha\beta|0\rangle|1\rangle + \beta\alpha|1\rangle|0\rangle + \beta^2|1\rangle|1\rangle$$

We notice that some terms are absent in the first equation, which proves the inexistence of any U. □

However, it is possible to transfer the state of a qubit to another one, using the Quantum Teleportation algorithm, proposed by Bennett et al in [18]. Figure Figure 2.2 illustrates the attempt to teleport the state

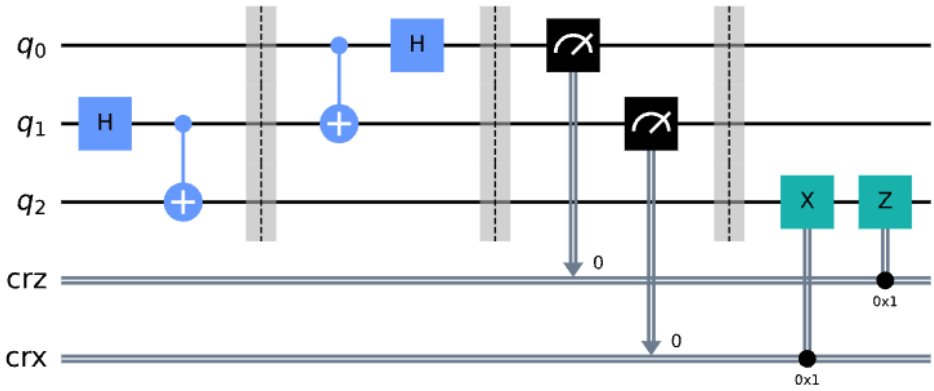

Figure 2.2: Quantum Teleportation Algorithm

of q_0 to q_2, where the qubit q_1 is a mediator.

We notice that there are four phases. The qubit q_0 is destroyed in Phase 3 (because of the effect of measurement). Phase 4 in the figure is a special case when q_1 and q_2 both measure to 1. The general case is as the following:

1. If q_0 measures to 0 and q_1 measures to 0 then do nothing.
2. If q_0 measures to 0 and q_1 measures to 1 then apply X gate.
3. If q_0 measures to 1 and q_1 measures to 0 then apply Z gate.
4. If q_0 measures to 1 and q_1 measures to 1 then apply Z gate then apply X gate.

In the end, we expect that the state of q_2 is the same as q_1 before starting Phase 1.

2.4.3 Grover's Algorithm

Assume that given a set of N items, one of them, ω, has a property that makes it distinguished. The problem is to find ω, which could be done with classical computation with an exhaustive search of a complexity $O(N)$. Grover has developed a quantum algorithm that achieves a quadratic speedup for this search [19]. First, we define the oracle of that represents the set

$$U_\omega \left|x\right\rangle = \begin{cases} -\left|x\right\rangle, & \text{if } x = \omega \\ \left|x\right\rangle, & \text{otherwise} \end{cases}$$

This makes it a diagonal matrix of the form $diag(\{(-1)^{f(2^i-1)}\}_{0\leq i<n})$.

Second, the algorithm starts with a guess, by applying a superposition on all the qubits:

$$\left|s\right\rangle = \frac{1}{\sqrt{N}} \sum_0^{n-1} \left|x\right\rangle$$

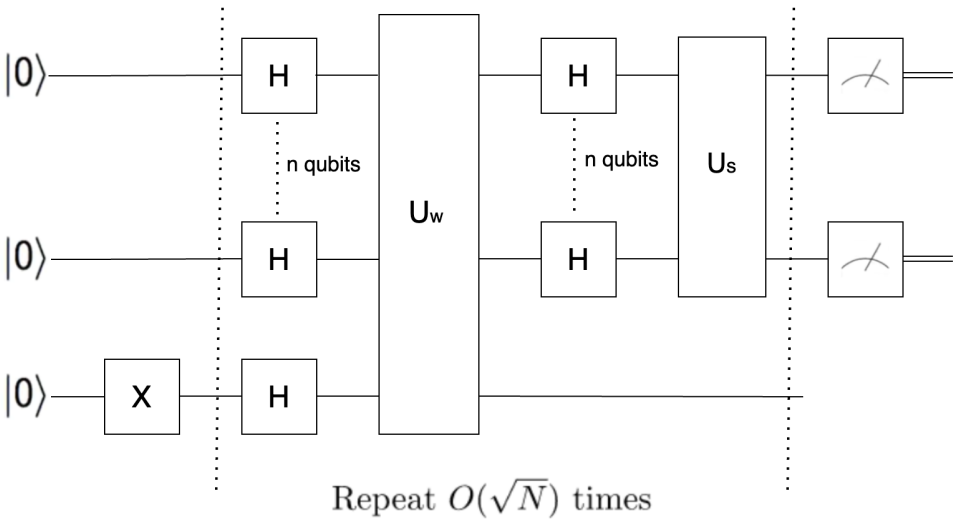

Figure 2.3: Grover Algorithm

Then, an iterative process called **Amplitude Amplification** is applied on this register, aiming to significantly enhance the probability of $|s\rangle$ landing on $|\omega\rangle$, and it iterates over three steps:

Step-1: applying uniform superposition $|s\rangle = \mathcal{H}^{\otimes n} |0\rangle^n$.

Step-2: applying the oracle which work as a reflection around $|\omega\rangle : U_\omega$: $|s\rangle = U_\omega |s\rangle$.

Step-3: applying another reflection around $|s\rangle$: $|s\rangle = U_s |x\rangle$.

After k steps, it reaches the state $|s_k\rangle = (U_s U_\omega)^k |s\rangle$. It was proven that the search will reach ω when $k \to \sqrt{N}$.

Mathematical Proof. Considering a space with the span $\{|s\rangle, |w\rangle\}$, then by the relations written above we can see that:

$$U_\omega |s\rangle = (I - 2|\omega\rangle\langle\omega|)|s\rangle = |s\rangle - \tfrac{2}{\sqrt{N}}|\omega\rangle \; U_\omega |\omega\rangle = (I - 2|\omega\rangle\langle\omega|)|\omega\rangle = 0.|s\rangle - |\omega\rangle$$

So we can define U_ω as a transformation

$$\psi \to [|s\rangle |\omega\rangle] \begin{bmatrix} -1 & -\frac{2}{\sqrt{n}} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p \\ q \end{bmatrix}$$

where ψ is linear combination of the basis $\{|s\rangle, |w\rangle\}$, i.e, $\psi = p|s\rangle + q|w\rangle$. Similarly, we can find that U_s is a also a transformation where $\psi \to \begin{bmatrix} -1 & 0 \\ \frac{2}{\sqrt{n}} & 1 \end{bmatrix} \psi$.

This makes $U_s U_\omega$, which is applied in every iteration of Grover's algorithm, equal to $\begin{bmatrix} 1 & \frac{2}{\sqrt{n}} \\ -\frac{2}{\sqrt{n}} & 1 - \frac{4}{N} \end{bmatrix}$.

Then we can diagonalize the resulted matrix to obtain:

$$U_s U_\omega = M. \begin{bmatrix} e^{2i.arcsin(\frac{1}{\sqrt{N}})} & 0 \\ 0 & e^{-2i.arcsin(\frac{1}{\sqrt{N}})} \end{bmatrix} .M^{-1}$$

where

$$M = \begin{bmatrix} -i & i \\ e^{i.arcsin(\frac{1}{\sqrt{N}})} & e^{-i.arcsin(\frac{1}{\sqrt{N}})} \end{bmatrix}$$

The probability of reaching $|\omega\rangle$ after k iterations is given by

$$\begin{aligned} ||\langle\omega|\psi_k\rangle||^2 &= || \langle\omega| \begin{bmatrix} |s\rangle & |\omega\rangle \end{bmatrix} (U_s U_\omega)^k \begin{bmatrix} p \\ q \end{bmatrix} ||^2 \\ &= || \begin{bmatrix} \langle\omega|s\rangle & \langle\omega|\omega\rangle \end{bmatrix} (U_s U_\omega)^k \begin{bmatrix} p \\ q \end{bmatrix} ||^2 \\ &= || \begin{bmatrix} \frac{1}{\sqrt{N}} & 1 \end{bmatrix} M. \begin{bmatrix} e^{i.k.arcsin(\frac{1}{\sqrt{N}})} & 0 \\ 0 & e^{-i.k.arcsin(\frac{1}{\sqrt{N}})} \end{bmatrix} .M^{-1} \begin{bmatrix} p \\ q \end{bmatrix} ||^2 \\ &= sin((2k+1)arcsin(\frac{1}{\sqrt{N}}))^2 \end{aligned}$$

which is approximately 1 when $k = \frac{\pi\sqrt{N}}{4}$, so the search will be accomplished in $O(\sqrt{N})$ iterations.

2.4.4 Deutsch-Josza Algorithm

David Deutsch and Richard Jozsa proposed a quantum algorithm to solve the following problem [20]:

> *Given a function $f : \{0,1\}^n \to \{0,1\}$, f is either constant, i.e,*
>
> $$\forall x \in \{0,1\}^n, f(x) = 0 \text{ or } f(x) = 1$$
>
> *or it is balanced, i.e.,*
>
> $$\#\{x \in \{0,1\}^n : f(x) = 0\} = \#\{x \in \{0,1\}^n : f(x) = 1\}$$
>
> *Determine whether f is constant or balanced using its oracle.*

The function f is implemented in an oracle U_f, where it receives two inputs, $x \in \{0,1\}^n$ and $y \in \{0,1\}$ and it outputs the following:

$$U_f |x\rangle |y\rangle = |x, y \oplus f(x)\rangle$$

The qubit $|y\rangle$ will be initialized to $|1\rangle$, so the the second output will be $1 - f(x)$. The algorithm is described with the quantum circuit in Figure Figure 2.4.

The measurement in the end of the circuit will output 1 if the function is balanced, and 0 if it is constant.

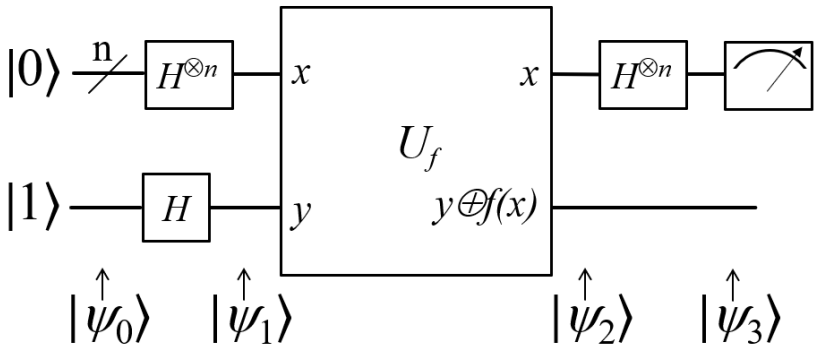

Figure 2.4: Deutsh-Josza Algorithm (Source: Wikipedia)

Correctness. We can track the state ψ of the quantum system to prove the correctness of this algorithm. Starting from ψ_0, i.e at the time of initialization, it will be:

$$\psi_0 = (\otimes_{i<n} |0\rangle) \otimes |1\rangle$$

The next state, ψ_1 comes after a Hadamard transformation over the whole system giving:

$$\begin{aligned}\psi_1 &= (\otimes_{i<n} H |0\rangle) \otimes H |1\rangle \\ &= \frac{1}{\sqrt{2}^{n+1}} \sum_{x=0}^{2^n-1} |x\rangle (|0\rangle - |1\rangle)\end{aligned}$$

Then, the oracle of f will be applied, resulting in the state ψ_2:

$$\begin{aligned}\psi_2 &= \frac{1}{\sqrt{2}^{n+1}} \sum_{x=0}^{2^n-1} |x\rangle (|f(x)\rangle - |1 \oplus f(x)\rangle) \\ &= \frac{1}{\sqrt{2}^{n+1}} \sum_{x=0}^{2^n-1} (-1)^{f(x)} |x\rangle (|0\rangle - |1\rangle)\end{aligned}$$

The last qubit is ignored, which leaves the state ψ_2 in the following form:

$$\psi_2 = \frac{1}{\sqrt{2}^{n}} \sum_{x=0}^{2^n-1} (-1)^{f(x)} |x\rangle$$

Another Hadamard transformation is applied on the system, giving the state ψ_3:

$$\psi_3 = \frac{1}{\sqrt{2}^n} \sum_{x=0}^{2^n-1} (-1)^{f(x)} (\frac{1}{\sqrt{2}^n} \sum_{y=0}^{2^n-1} (-1)^{x.y} |y\rangle)$$
$$= \frac{1}{2^n} \sum_{y=0}^{2^n-1} (\sum_{x=0}^{2^n-1} (-1)^{f(x)+x.y}) |y\rangle$$

where $x.y = \oplus_{i=0}^{n-1} x_i y_i$. The probability of measuring $|0\rangle^n$ in the end, is

$$|\tfrac{1}{2^n} \textstyle\sum_{x=0}^{2^n-1} (-1)^{f(x)}|^2$$

which is 0 if f is balanced, or 1 if it is constant, or, if the circuit measures to $|0\rangle^n$, it means that the function is constant, and balanced otherwise.

2.5 Quantum Programming Languages and Frameworks

Many players on the quantum field have introduced their own tools and frameworks to develop quantum programs. In this section, we will list some of those.

2.5.1 OpenQasm

Released by IBM in 2017, OpenQasm [2]. is an intermediate quantum instructions representation, designed as a low level processor language, analogous to assembly for classical hardware, but in this case it is used to describe quantum circuits.

In OpenQasm, it is possible to define bits, qubits, to apply a variety of gates, and to measure qubits. It also allows *if* and *while* statements. The program in Listing Listing 2.1 is an implementation of quantum teleportation in OpenQasm (the explanation of quantum teleportation is done in Section Section 2.4.2).

Listing 2.1: Quantum teleportation in OpenQasm

```
OPENQASM 3;
include "stdgates.inc";
qubit[3] q;
bit c0;
bit c1;
bit c2;
gate post q { }
```

[2] https://qiskit.github.io/openqasm/

```
reset q;
U(0.3, 0.2, 0.1) q[0];
h q[1];
cx q[1], q[2];
barrier q;
cx q[0], q[1];
h q[0];
c0 = measure q[0];
c1 = measure q[1];
if(c0==1) z q[2];
if(c1==1) x q[2];
post q[2];
c2 = measure q[2];
```

2.5.2 Qiskit

Qiskit [3] is an SDK designed by IBM to design quantum circuits and transformations in Python, and then run them either locally (using a simulator) or on IBM Quantum experience backend (via a web service). Qiskit allows to design a circuit with quantum and classical wires, applying gates sequentially, and measuring states. We can also use the syntax of Python to make conditional statements or loops. Listing Listing 2.2 shows the algorithm of quantum teleportation in Qiskit.

Listing 2.2: Quantum teleportation in Qiskit

```
from qiskit import QuantumCircuit, Aer, execute

qc = QuantumCircuit(3, 3)

qc.h(0)
qc.cx(0, 1)
qc.cx(1, 2)
qc.measure([0,1,2], [0,1,2])
backend = Aer.get_backend("qasm_simulator")
```

[3] https://qiskit.org/documentation/

```
job = execute(qc, backend)
result = job.result()
print(result.get_counts(qc))
```

The framework offers some useful functionalities like drawing circuits, calculating states like density matrix or vector state, and visualizing the results of the execution.

2.5.3 Q#

Developed by Microsoft for its Quantum Development Kit (QDK) [4] , Q# is a high level, domain-specific language to design quantum programs and focus on the algorithmic level. In addition to supporting qubits and gates, it allows *for* loops, defining functions (including lambda functions), modules and many classical data types like Strings, Integers, Doubles and Booleans. The example in Listing Listing 2.3 is the code for quantum teleportation in Q#.

Listing 2.3: Quantum teleportation in Q#

```
operation Teleport (msg : Qubit, target : Qubit) : Unit {
    use register = Qubit();

    H(register);
    CNOT(register, target);

    CNOT(msg, register);
    H(msg);

    if (IsResultOne(MResetZ(register))) { X(target); }
}
```

2.5.4 Cirq

Google has developed a Python library [5] to write, manipulate and optimize quantum circuits, and it was provided with a mechanism to mitigate quantum noise in current quantum computers. Like the previous languages, it is possible to initialize qubits, applying gates, measuring and executing conditional and loop

[4] https://docs.microsoft.com/en-us/azure/quantum/overview-what-is-qsharp-and-qdk
[5] https://quantumai.google/cirq

statements (using Python's syntax), In addition to the ability to execute the code on Google Quantum Backend. The code in Listing Listing 2.4 is a Cirq implementation of quantum teleportation.

Listing 2.4: Quantum teleportation in Cirq

```
import cirq
circuit = cirq.Circuit()

m = cirq.NamedQubit("m")
a = cirq.NamedQubit("a")
b = cirq.NamedQubit("b")

circuit.append(gate(b))

circuit.append([cirq.H(a), cirq.CNOT(a, b)])

circuit.append([cirq.CNOT(m, a), cirq.H(m), cirq.measure(m, a)])

circuit.append([cirq.CNOT(a, b), cirq.CZ(m, b)])
```

2.5.5 Silq

Silq [6], developed by SRI Lab in ETH Zurich, is a high level quantum programming langauge that provides a string static type system [21] and can run the traditional quantum operations, conditions and loops. The Type system of Silq uses Uncomputation to help preventing some errors like consuming used variables or implicit measurements.

Listing 2.5: Quantum teleportation in Silq

```
def main() {
  return Teleportation();
}
def Teleportation(){
    m := 0:B;
```

[6] https://silq.ethz.ch/overview

```
    a := 0:B;
    b := 0:B;
    m := 1/3;

    a := H(a);
    b := cx(a, b);
    a := cx(m,a);
    m := H(m);

    m := measure(m);
    a := measure(a);

    if m==1 {
        if a==1{
          b := X(b);
          b := Z(b);
        }
        else{
          b := X(b);
        }
    }
    else {
        if a==1 {
          b := X(b);
        }
    }
    b := measure(b);
}
```

3

Survey: Hoare Logics for Quantum Programs

Contents

In this chapter, we list four different examples of Hoare logics that can be used to reason about quantum programs, and we compare them in terms of language expressiveness, soundness, completeness, and the ability to apply them to programs that require reasoning about classical states.

3.1 Chadha, Mateus and Sernadas's EEQPL

In [5], Chadha, Mateus and Sernadas extend their EEPL (a complete logic to reason on probabilistic programs) to EEQPL, with the ability to reason about quantum programs. The language contains classical and quantum instructions, with the following syntax:

$$c := skip \mid b := b \mid n := e \mid I \mid H : q \mid H : qn \mid \sigma_x : q \mid \sigma_x : q(e,e) \mid S(e,b) : q \mid S(e,e) : qn \mid UU$$

$$\mid qif\ q\ then\ U\ else\ U \mid qcase\ qn\ : 0 : U, ..., n-1 : U \mid b :^m= qn \mid n :^m= qn \mid c; c$$

$$if\ b\ then\ c\ else\ c \mid case\ n\ :\ 0 : c, ..., n-1 : c \mid n\ repeat\ c$$

As we can see, the language uses four types of data: bits b, natural numbers n, qubits q, and qunits qn. *Qunits* are the natural numbers' equivalent for qubits. The quantum operations are H for Hadamard Gate and it could be applied on qubits or qunits, σ_x is the NOT gate, S is a Phase shifting gate with two arguments, and $:^m=$ refers to measuring operation. Expressions e are defined by the authors to include Real numbers, Complex numbers, and bra-kets expressions. Note that the language lacks *while* loops.

3.1.1 EEQPL rules

Here are some of the rules for EEQPL:

$$Skip\frac{}{\{P\}skip\{P\}}$$

$$Asgn\frac{}{\{P[\mathbf{b} \to b]\}\mathbf{b} := b\{P\}}$$

$$Unit\frac{}{\{P[\langle u_1|u_2\rangle \to \langle U.u_1|u_2\rangle]\}U\{P\}}$$

$$Seq\frac{\{P\}c_1\{Q\}\ \ \{Q\}c_2\{R\}}{\{P\}c1; c2\{R\}}$$

$$Cons\frac{P' \to P\ \ \{P\}c\{Q\}\ \ Q \to Q'}{\{P'\}c\{Q'\}}$$

$$Meas\frac{}{\{P[E(r) \to m_1^{b,q}E(r) + m_0^{b,q}E(r)]\}b :^m= q\{P\}}$$

$$If\frac{\{P_1\}c1;\mathbf{b}:=true\{Q_1\}\ \ \{P_2\}c2;\mathbf{b}:=false\{Q_2\}}{\{(P_1/b\wedge(P_2/\neg b))\}if\ b\ then\ c1\ else\ c2\{(Q_1/b\wedge(Q_2/\neg b))\}}$$

Note that P/b is the logical value of P after substituting b with $true$, and $E(r)$ is the real expectation of satisfaction of P upon measurement.

The logic is proven to be sound, but not complete [5], and the authors used the rules to prove the correctness of the Deutsch Algorithm.

3.2 Yoshihiko Kakutani's QHL

In [22], Selinger designs a simple, well-defined functional quantum language called *QPL*, with the following Syntax:

$$c:=\textbf{skip}\mid c\ ;\ c\mid\textbf{bit}\ b\mid\textbf{qbit}\ q\mid\textbf{discard}\ q\mid b:=0\mid b:=1\mid$$

$$\overline{q}*=\ U\mid\textbf{if}\ b\ \textbf{then}\ c\ \textbf{else}\ c\mid\textbf{while}\ b\ \textbf{do}\ c\mid\textbf{measure}\ q\ \textbf{then}\ c\ \textbf{else}\ c$$

Note that $discard$ could be applied on bits or qubits.

The language does not have a global state, but instead, Kakutani's [6] uses its denotational semantics as functions of matrices to define a Quantum Hoare Logic *QHL* over *QPL*, but only with measurement and while loops and no recursive functions.

Assertions in QHL could be First Order Logic predicates over real numbers, or Density matrices, adding/discarding qubits or bits, or applying unitary transformations impact the density matrix.

3.2.1 QHL rules

Many of the following rules are standard as we can see in the following annotation:

$$Skip\frac{}{\{P\}skip\{P\}}$$

$$Seq\frac{\{P\}c_1\{Q\}\ \ \{Q\}c_2\{R\}}{\{P\}c1;c2\{R\}}$$

$$Asgn0\frac{}{\{P\}\mathbf{b}:=0\{|1\rangle\langle 0|\,P+|1\rangle\langle 1|\,P\}}$$

$$Asgn1\frac{}{\{P\}\mathbf{b}:=1\{|1\rangle\langle 0|\,P+|1\rangle\langle 1|\,P\}}$$

$$Init-b\frac{}{\{P\wedge P[t]=1\}bit\ \mathbf{b}\{P\wedge P[b=0]=1\}}$$

$$Init-q\frac{}{\{P\wedge P[t]=1\}qbit\ \mathbf{q}\{P\wedge P[b=0]=1\}}$$

$$Unit\frac{}{\{U^\dagger P\}\overline{q}*=U\{P\}}$$

$$Discard\frac{\mathbf{b} \notin vars(P)}{\{P\}discard\ \mathbf{b}\{P\}}$$

$$Cons\frac{P' \to P \ \ \{P\}c\{Q\} \ \ Q \to Q'}{\{P'\}c\{Q'\}}$$

$$If\frac{\{|1\rangle\langle 0|\, P\}c1\{Q_1\} \ \ \{|1\rangle\langle 1|\, P\}c2\{Q_2\}}{\{P\}if\ b\ then\ c1\ else\ c2\{(Q_1 + Q_2\}}$$

$$Meas\frac{\{|1\rangle\langle 0|\, P\}c1\{Q_1\} \ \ \{|1\rangle\langle 1|\, P\}c2\{Q_2\}}{\{P\}measure\ q\ then\ c1\ else\ c2\{(Q_1 + Q_2\}}$$

$$While\frac{\{|1\rangle\langle 0|\, P_n\}c\{P_{n+1}\}\forall n \in \mathbf{N} \ \ \{|1\rangle\langle 0|\, P_n | n \in \mathbf{N}\} \models Q}{\{P\}while\ b\ do\ c\{(Q\}}$$

There are more rules for the other variants of Assertions like linearity or existentials, but it is necessary to highlight on the *While* rule, as it is not very standard in this logic, because it does not reason about an invariant but it only says that the if the matrix of the last state P_n implies Q and the loop terminates then the triple is correct, which is not very useful. The paper suggests an invariant-based *While* rule with some conditions, like guaranteeing that the program is going to terminate, and that P_i cannot contain any negation, disjunction or existentials. Even with this rule, there is no claim (and yet a proof), that the logic is complete. However, it was used to prove the correctness of Deutsch and Shor's algorithms in [6], and to prove the security of Quantum Cryptography Protocols in [23].

3.3 Mingsheng Ying's qPD

Ying introduced *qPD* [7], a complete Hoare logic, that uses D'Hondt and Panangaden's quantum predicates [24] and a syntax similar to *QPL* (Selinger [22]). qPD only deals with quantum variables, and it can represent integers using infinite Hilbert space $\mathcal{H}_\infty$ and booleans with binary Hilbert space $\mathcal{H}_2$.
The syntax is defined by Ying [7] as the following:

$$S ::= \textbf{skip} \mid q := 0 \mid qn := 0 \mid \overline{q}* = U \mid S_1; S_2 \mid \textbf{measure}\ M[\overline{q}] : \overline{S} \mid \textbf{while}\ M[\overline{q}]\ \textbf{do}\ S\ \textbf{end}$$

The program state is the density operator that represents the mixed quantum state of the variables, and assertions (as defined in [24]) are positive Hermitian operators with a maximum eigenvalue less than 1.

Definition 14. *The denotational semantics of a program c indicates the states in which a program might terminate starting from a given state, and is defined as the following:*

$$[\![c]\!](\rho) = \sum\{\rho' : \langle c, \rho\rangle \longrightarrow *\langle skip, \rho'\rangle\}$$

Lemma 7. *A program c terminates when $tr(\rho) = tr([\![c]\!](\rho))$.*

The Hoare triple of the form $\{P\}c\{Q\}$ is totally satisfied, or $\models_{tot} \{P\}c\{Q\}$, iff

$$\forall \rho, tr(P\rho) \leq tr(Q[\![c]\!]\rho)$$

The same triple is partially satisfied iff

$$\forall \rho, tr(P\rho) \leq tr(Q[\![c]\!](\rho)) + tr(\rho) - tr([\![c]\!](\rho))$$

The term $tr(\rho) - tr([\![c]\!](\rho))$ represents the probability that the program terminates.

3.3.1 qPD rules

First, the set of Quantum predicates is ordered with respect to Lowner partial order, i.e, we say that $P \sqsubseteq Q$ iff $\forall \rho, tr(P\rho) \leq tr(Q\rho)$. This definition is helpful for some of the following rules:

$$Skip\frac{}{\{P\}skip\{P\}}$$

$$Seq\frac{\{P\}c_1\{Q\} \;\; \{Q\}c_2\{R\}}{\{P\}c1; c2\{R\}}$$

$$AsgnB\frac{}{\{\sum_{n\in\{0,1\}} |n\rangle \langle 0| P |0\rangle \langle n|\}q := 0\{P\}}$$

$$AsgnN\frac{}{\{\sum_{n=-\infty}^{\infty} |n\rangle \langle 0| P |0\rangle \langle n|\}qn := 0\{P\}}$$

$$AsgnN\frac{}{\{\sum_{n=-\infty}^{\infty} |n\rangle \langle 0| P |0\rangle \langle n|\}qn := 0\{P\}}$$

$$Unit\frac{}{\{U^\dagger P U\}\overline{q}* = U\{R\}}$$

$$Cons\frac{P' \sqsubseteq P \;\; \{P\}c\{Q\} \;\; Q \sqsubseteq Q'}{\{P'\}c\{Q'\}}$$

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

$$While\frac{\{Q\}c_m\{M_0^\dagger P M_0 + M_1^\dagger Q M_1\}}{\{M_0^\dagger P M_0 + M_1^\dagger Q M_1\}while\ M[\overline{q}]\ do\ c\{P\}}$$

As stated before, this system was proven (by Ying [7]) to be sound and complete, i.e any triple $\{P\}c\{Q\}$ is derivable using *qPD*. There is also a total correctness version with a different *While* rule, that was also proven to be sound and correct.

3.4 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 Section 3.3.1, 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. The new concepts of States and Assertions are going to be further discussed later in Definition 16. Briefly, a quantum-classical state is a function that maps each state of the classical variables to a respective partial density operator over the quantum variables of the system. Moreover, a quantum-classical assertion is a function between assertions over classical variables to a density operator. The table Table 3.1 compares between classical, probabilistic, quantum and quantum classical with respect to the definition of "State", "Assertion" and "Satisfaction".

	Classical	**Probabilistic**	**Quantum**	**Quantum-Classical**
State	$\sigma \in \Sigma$	$\mu \in \Sigma \to [0,1]$	$\rho \in \mathcal{D}_{\mathcal{H}}$	$\Delta \in \Sigma \to \mathcal{D}(\mathcal{H})$
Assertion	$p \in \Sigma \to \{0,1\}$	$f \in \Sigma \to [0,1]$	$M \in \mathcal{P}(\mathcal{H})$	$\Theta \in \Sigma \to \mathcal{P}(\mathcal{H})$
Satisfaction	$\sigma \mid = p$	$\sum_{\sigma \in Dom(\mu)} \mu(\sigma) f(\sigma)$	$tr(M\rho)$	$\sum_{\sigma \in \Sigma} tr(\Delta(\sigma)\Theta(\sigma))$

Table 3.1: Comparison between different types of languages [1]

In qPD, the expectation of satisfaction is the probabilty that a state satisfies an assertion, and it was given by the formula:

$$Exp(\sigma \models \theta) = tr(\sigma.\theta)$$

While in the Quantum-classical state, we generalize the previous formula as follows:

$$Exp(\Delta \models \Theta) = \sum_{\sigma \in \lceil \Delta \rceil} tr([\Theta(\sigma) \otimes I_{\mathcal{H}_V}].\Delta(\sigma))$$

where $\lceil \Delta \rceil$ is the image set of Δ, and $\mathcal{H}_V$ is the Hilbert space of the state's quantum variables.

Lemma 8. *Let Σ_V be the set of all quantum classical states over the variables set V, and $\mathcal{A}_W$ be the set of all quantum classical assertions over the variables set W. If W is a subset of V then for any state $\Delta \in \pm_V$, quantum classical assertion $\Theta \in \mathcal{A}_W$, and classical assertion p:*

1. $Exp(\Delta \models \Theta) \in [0,1]$
2. $Exp(\sum_i \lambda_i \Delta_i \models \Theta) = \sum_i \lambda_i Exp(\Delta_i \models \Theta)$
3. $Exp(\Delta \models \sum_i \lambda_i \Theta) = \sum_i \lambda_i Exp(\Delta \models \Theta_i)$
4. $Exp(\Delta_i|_p \models \Theta) = Exp(\Delta) \models p \wedge \Theta)$ *where $\Delta_i|_p$ is the quantum classical state of the set of classical states of Δ that satisfy p.*

Consider x, a classical variable, q, a quantum variable, e, an expression, g, a function that maps values to probabilities, U, a unitary transformation, and $\mathcal{M}$, the basis of the Hilbert space of quantum variables. Feng and Ying [1] define a program S using the following syntax:

$$S ::= \textbf{skip} \mid \textbf{abort} \mid \overline{q} := 0 \mid \overline{q} * = U \mid S_1; S_2 \mid x := e \mid x :=_{\$} g \mid x := meas\ \mathcal{M}[\overline{q}] \mid \textbf{while}\ M[\overline{q}]\ \textbf{do}\ S\ \textbf{end}$$

we can see that a new instruction $x :=_{\$} g$ has been introduced. It means that the classical variable x is assigned to a probabilistic distribution of values.
The authors of [1] did also define the operational and dentational semantics of each one of the previous instructions. where they denoted $[\![S]\!](\sigma)$ to be the set of all quantum classical states the a program S can produce after reducing it to the empty program E. They used the dentational semantics later to define Hoare triples as the following:

Definition 15. *The formula $\{P\}S\{Q\}$ is totally correct, and we write $\models_{tot} \{P\}S\{Q\}$, if for any $V, \Delta : qv(P, S, Q) \not\subseteq V$ and $\Delta \in \mathcal{S}_V$:*

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

3.4.1 Feng-Ying Logic rules

Consider Θ a quantum-classical assertion, and $qVar(\Theta)$ is the set of quantum variables in the assertion Θ. using the definition in Definition 15, 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 [25].

4

A Verified Quantum Programming Language with Hybrid Variables

Contents

In this chapter, we are going to present the language described in [1], its syntax, operational semantics and a proof system, with some changes that we added to it, to be implemented later in Chapter 5.

4.1 Syntax

We slightly changed the language defined by Feng and Ying in [1], to come up with the following syntax:

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

In the following, we list the differences with the language in [1]:

- We removed the command that assigns a classical variable to a probabilistic distribution, because we did not find it relevant to the problem.
- It is possible to initialize/measure only one qubit at a time.
- It is possible to apply a Unitary transformation on one qubit (or two adjacent qubit in case of a $CNOT$ gate).
- We uses indices as IDs for the quantum variables.

The main shortcomings of this language is its lack of expressiveness, as it initializes one qubit at a time and allows applying unitary transformations on a limited number of qubits, to serve a purpose of proving the concept of the possibility to reason about such programs. It would still be an interesting open point for the future to extend the language and make it more expressive. The following definitions show the expressions and gates that could be used in our language:

$$U ::= X \mid Y \mid Z \mid I \mid H \mid CNOT$$

$$e ::= X_{id} \mid n \mid e + e \mid e - e \mid e \times e \mid e/e : e \neq 0$$

$$b ::= true \mid false \mid b \wedge b \mid b \vee b \mid \neg b \mid e == e \mid e <= e \mid e < e$$

Here, X_{id} refers to an identifier to classical variable.
Example 5 shows a program written in our language.

Example 5.

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

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 ($CNOT$).

4.2 Variables and Types

In our language, we consider 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.

4.3 Quantum-Classical State

Adapted from the definition of Feng and Ying [1], the state of a program written in our language consists of elements to represent the classical part and the corresponding quantum probabilistic distribution of possible measurement outcomes. We use Σ to denote the set of all states of the classical variables in a program.

Definition 16. *A Quantum-Classical State, or shortly a cq-state, is defined as following:*

$$\Delta = \{(\sigma, \rho),\ where\ \sigma \in \Sigma, \rho \in \mathcal{D}(\mathcal{H})\} \tag{4.1}$$

where Σ is the set of all classical states, i.e, a mapping between classical variables identifiers and values, and $\mathcal{D}(\mathcal{H})$ is the set of all partial density operators. Δ should be countable.

In plain words, the state is a countable set of pairs of key-value combinations that represent current values of classical variables, and the corresponding density matrix of the quantum variables.

Looking at Example 5, we expect the program to end up with the following state:

$$\{(x \to 0, y \to 0) \to \rho_{00}, (x \to 0, y \to 1) \to \rho_{01},$$

$$(x \to 1, y \to 0) \to \rho_{10}, (x \to 1, y \to 1) \to \rho_{11}\}$$

where

$$\rho_{00} = (|0\rangle\langle 0| \otimes I)(X \otimes I)(I \otimes |0\rangle\langle 0|)(I \otimes H)\,|00\rangle\langle 00|\,(I \otimes H)^\dagger (I \otimes |0\rangle\langle 0|)^\dagger (X \otimes I)^\dagger (|0\rangle\langle 0| \otimes I)^\dagger$$

$$\rho_{01} = (|1\rangle\langle 1| \otimes I)(X \otimes I)(I \otimes |0\rangle\langle 0|)(I \otimes H)\,|00\rangle\langle 00|\,(I \otimes H)^\dagger (I \otimes |0\rangle\langle 0|)^\dagger (X \otimes I)^\dagger (|1\rangle\langle 1| \otimes I)^\dagger$$

$$\rho_{10} = (|0\rangle\langle 0| \otimes I)(I \otimes |1\rangle\langle 1|)(I \otimes H)\,|00\rangle\langle 00|\,(I \otimes H)^\dagger (I \otimes |1\rangle\langle 1|)^\dagger (|0\rangle\langle 0| \otimes I)^\dagger$$

$$\rho_{11} = (|1\rangle\langle 1| \otimes I)(I \otimes |1\rangle\langle 1|)(I \otimes H)\,|00\rangle\langle 00|\,(I \otimes H)^\dagger (I \otimes |1\rangle\langle 1|)^\dagger (|1\rangle\langle 1| \otimes I)^\dagger$$

And we will show later that the probabilistic distribution of the results is going to be:

$$\{(x \to 0, y \to 0) \to 0, (x \to 0, y \to 1) \to 0.5, (x \to 1, y \to 0) \to 0.5, (x \to 1, y \to 1) \to 0\}$$

4.4 Quantum-Classical Assertions

We recall the set $\mathcal{P}(\mathcal{H})$ to be the set of all Density operators whose eigenvalues are in the interval $[0, 1]$. Assertions in classical programs are First-order logic statements that could be tested against the program state, while in Quantum programs, In [24], D'hondt et al defined a Quantum state to be a Density operators in $\mathcal{P}_\mathcal{H}$. In Quantum-classical programs, the state of Classical variables is mapped to the corresponding state the quantum registers system before measurement. This is similar to what is usually done for probabilistic programs.

Definition 17. *A quantum-classical assertion is defined as the following:*

$$\psi : \mathcal{FOP}(\mathcal{V}) \to \mathcal{P}_\mathcal{H}$$

*where $\mathcal{FOP}(\mathcal{V})$ stands for the set of First-order logic statements on the set of classical variables V. To ease the calculations, we consider that the default value of $P(\sigma)$ is **0** for all $\sigma \in \Sigma$, if not indicated otherwise.*

Example 6. *$(X\ mod\ 2 = 0, |00\rangle\langle 00|)$ is an example of a Quantum-Classical assertion.*

Another contribution from Feng and Ying was defining the degree of satisfaction of a certain Quantum-Classical assertion by a Quantum-Classical state. We adapted in our definitions to be as following:

Definition 18. *Let Δ_{cq} be a Quantum-Classical state defined as in Definition Definition 16, and let Ψ_{cq} be a Quantum-Classical assertion as defined above. Then we say that Δ_{cq} is expected to satisfy Ψ_{cq} with a degree:*

$$Exp(\Delta_{cq} \models \Psi_{cq}) = \sum_{\sigma \in \Delta_{cq} \wedge \sigma_1 \models \Psi_{cq}(\sigma_0)} tr(\sigma_2.(\Psi_{cq}(\sigma_1) \otimes I_{\mathcal{H}_V})) \tag{4.2}$$

Here, $\mathcal{H}_V$ refers to the set of quantum variables in Ψ_{cq} but not in Δ_{cq}.

The Expectation in the previous definition only adds up states whose classical part satisfies the classical part of the assertion, because it is iterating over a countable set.

The following examples illustrate the calculation of the expectation of an assertion.

Example 7. *Considering the state* Δ

$$\{(X \to 2) \to 0.5 \left|+\right\rangle \left\langle+\right|, (X \to 4) \to 0.5 \left|-\right\rangle \left\langle-\right|\}$$

and the assertion Ψ

$$(X \leq 3, 1)$$

then the expectation of $\Delta \models \Psi$ *is given by:*

$$Exp(\Delta \models \Psi) = trace(0.5 \left|+\right\rangle \left\langle+\right| .I) = 0.5$$

This is because the proposition $X \leq 3$ *is satisfied when* $X \to 2$*.*

Definition 19. *We say that an assertion* Ψ_1 *is weaker than an assertion* Ψ_2 *iff*

$$Exp(\Delta \models \Psi_1) \leq Exp(\Delta \models \Psi_2)$$

for all Quantum-Classical states Δ*.*

Example 8. *Let* Ψ_1, Ψ_2 *be two assertions where:*

$$\Psi_1 = (x \leq 3, 0.5) \text{ and } \Psi_2 = (x \leq 5, 0.5)$$

Knowing that for any classical state σ*, we have* $\sigma \models \Psi_{11} \implies \sigma \models \Psi_{21}$ *and* $\Psi_{12} = \Psi_{22}$*, then, for all states* Δ*:*

$$Exp(\Delta \models \Psi_1) = \sum_{(\sigma,\rho)\in\Delta, \sigma\models\Psi_{11}} trace(\rho.\Psi_{12}) \leq \sum_{(\sigma,\rho)\in\Delta, \sigma\models\Psi_{21}} trace(\rho.\Psi_{22}) = Exp(\Delta \models \Psi_2)$$

Hence, Ψ_1 *is weaker than* Ψ_2*.*

This definition allows us to construct a complete partial order set of Quantum-Classical assertions as mentioned before and proved by Feng and Ying [1].

4.5 State Update and Semantics

Definition 20. *The triple* $(Prog_0 \to Prog_1, \sigma_0 \to \sigma_1, \rho_0 \to \rho_1)$ *represents the transition of the program* $Prog_0$ *to* $Prog_1$ *and the implied change of both classical and quantum parts of the 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 rules are state updating rules:

1. (**skip** $\rightarrow E, \sigma \rightarrow \sigma, \rho \rightarrow \rho$)
2. $(x := e \rightarrow E, \sigma \rightarrow \{\sigma, x \rightarrow e\}, \rho \rightarrow \rho)$
3. (**new_qubit** $\rightarrow E, \sigma \rightarrow \sigma, \rho \rightarrow |0\rangle \otimes \rho \otimes \langle 0|$)
4. (**q** $n * = U \rightarrow E, \sigma \rightarrow \sigma, \rho \rightarrow padding(U, n).\rho.padding(U, n)^\dagger$)
5. $(x :=$ **meas** $n \rightarrow E, \sigma \rightarrow \{\sigma[x/0]\}, \rho \rightarrow M_0(n).\rho.M_0(n)^\dagger)$
6. $(x :=$ **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 ((**if** b **then** S_1 **else** S_2) $\rightarrow S_1, \sigma \rightarrow \sigma, \rho \rightarrow \rho$), otherwise, then ((**if** b **then** S_1 **else** S_2) $\rightarrow S_2, \sigma \rightarrow \sigma, \rho \rightarrow \rho$).
9. if $\sigma \models b$ then ((**while** b **do** S **end**) $\rightarrow (S;$ **while** b **do** S **end**)$, \sigma \rightarrow \sigma, \rho \rightarrow \rho)$, otherwise, ((**while** b **do** S **end**) $\rightarrow E, \sigma \rightarrow \sigma, \rho \rightarrow \rho$).

The functions $padding$, M_0 and M_1 are used to manipulate the density matrix of the system, and they are defined as follows:

$$padding(U, n) = (\otimes_{i=0}^{n-1} I) \otimes U \otimes (\otimes_{i=n+dim(U)}^{dim} I) \tag{4.3}$$

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.

$$M_0(n) = (\otimes_{i=0}^{n-1} I) \otimes |0\rangle \langle 0| \otimes (\otimes_{i=n+1}^{dim} I) \tag{4.4}$$

$$M_1(n) = (\otimes_{i=0}^{n-1} I) \otimes |1\rangle \langle 1| \otimes (\otimes_{i=n+1}^{dim} I) \tag{4.5}$$

Similarly, $M_0(n)$ and $M_1(n)$ give 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 qubit n again is 0, and vice versa for $M_1(n)$.

Theorem 1. *$M_0(n)$ and $M_1(n)$ are valid measurement operators, or:*

$$\forall n \in \mathbf{N}_{dim} : M_0(n).M_0(n)^\dagger + M_1(n).M_1(n)^\dagger = I$$

The state update rules formulate an operational semantics for the language, and it could be used to evaluate the final state of an execution and judge some assertions over it.

Example 9. *In the following example, we track the Program/State updates in the program shown in Example 5:*

$$(\textbf{\textit{new q}}\ ;\ ..., \{\{\} \to 1\})$$

The program starts with a mapping that points an empty classical state to 1.

$$(\textbf{\textit{new q}};\ ..., \{\{\} \to \rho_0\})$$

where $\rho_0 = |0\rangle\langle 0|$. *Upon initializing the first qubit, we realize that the density state updates to* ρ_0.

$$(\textbf{\textit{q}}\ 0 *= \ H; ..., \{\{\} \to \rho_1\})$$

where $\rho_1 = |0\rangle \rho_0 \langle 0| = |00\rangle\langle 00|$. *We initialize the second qubit, and the density state becomes* $|00\rangle\langle 00|$.

$$(x \textbf{\textit{:=meas}}\ 0; ..., \{\{\} \to \rho_2\})$$

where $\rho_2 = (I \otimes H)\rho_1(I \otimes H)^\dagger$.
After applying the Hadamard gate H *on the first qubit, the density operator will become* ρ_2, *now we need to measure it and store the result the classical variable* x.

$$(\textbf{\textit{if}}\ x == 0\ \textbf{\textit{then q}}\ 1 *= H\ \textbf{\textit{else skip end}}; ..., \{\{x \to 0\} \to \rho_{30}, \{x \to 1\} \to \rho_{31}\})$$

Where $\rho_{30} = M_0\rho_2 M0^\dagger$, $\rho_{31} = M_1\rho_2 M1^\dagger$, $M_0 = (I \otimes |0\rangle\langle 0|)$ *and* $M_1 = (I \otimes |1\rangle\langle 1|)$.
We see the Boolean condition of the if *statement is satisfied for some elements of the list of the Quantum-Classical state, so we apply the instruction* **q** $1 *= X$ *to the state that contains* $\{x \to 0\}$ *and the* $skip$ *instruction to the other states, leaving it unchanged, so the final state after applying the statement is*

$$(y := \textbf{\textit{meas}}\ 1, \{\{x \to 0\} \to \rho_{40}, \{x \to 1\} \to \rho_{31}\})$$

where $\rho_{40} = (X \otimes I)\rho_{30}(X \otimes I)^\dagger$.
And now we measure the left qubit, and put the result into y, *the final state will look like the following:*

$$(E,\ \{\{x \to 0, y \to 0\} \to \rho_{5_{00}},$$

$$\{x \to 0, y \to 1\} \to \rho_{5_{01}},$$

$$\{x \to 1, y \to 0\} \to \rho_{5_{10}},$$

$$\{x \to 1, y \to 1\} \to \rho_{5_{11}}\}$$

Where $\rho_{5_{00}} = M_0\rho_{40}M0^\dagger$, $\rho_{5_{01}} = M_1\rho_{40}M1^\dagger$, $\rho_{5_{10}} = M_0\rho_{31}M0^\dagger$, $\rho_{5_{11}} = M_1\rho_{31}M1^\dagger$, $M_0 = (|0\rangle\langle 0| \otimes I)$ *and* $M_1 = (|1\rangle\langle 1| \otimes I)$.
After simplifying the mathematical expressions, and calculating the traces of the unnormalised density matrices, we obtain the probabilistic distribution of the possible classical states, as follows:

$$\begin{aligned} final\_state = (E,\ [\{x \to 0, y \to 0\} \to 0, \{x \to 0, y \to 1\} \to 0.5,\\ \{x \to 1, y \to 0\} \to 0.5, \{x \to 1, y \to 1\} \to 0]) \end{aligned} \tag{4.6}$$

By looking at the final state, we can conclude that the assertion $x \neq y$ is satisfied with an expectation 1, and similarly, we can evaluate the satisfaction of other assertions by testing them against the final state of the program.

Definition 21. *Starting from a Quantum-Classical state* $\Delta \in \Sigma$*, the final state after executing the program* $Prog$ *is denoted by* $Prog[\Delta]$.

For example, we can say that if $Prog$ is the program in Example 5, and $\Delta = \{\}$ then

$$Prog[\Delta] = final\_state$$

This definition is going to be used later in the definition of Hoare triple.

4.6 A Hoare Logic for Quantum-Classical Programs

The triple $\{P\}Prog\{Q\}$ denotes a formula to express the correctness of the program $Prog$.

Definition 22. *Let* P *and* Q *be Quantum-Classical assertions. We say that* $\{P\}Prog\{Q\}$ *is totally correct, and we write*

$$\models_{tot} \{P\}Prog\{Q\}$$

if

$$\forall \Delta \in \Sigma, Exp(\Delta \models P) \leq Exp(Prog[\Delta] \models Q) \tag{4.7}$$

Moreover, we say that $\{P\}Prog\{Q\}$ *is partially correct, and we write*

$$\models_{par} \{P\}Prog\{Q\}$$

if

$$\forall \Delta \in \Sigma, Exp(\Delta \models P) \leq Exp(Prog[\Delta] \models Q) + tr(\Delta) - tr(Prog[\Delta]) \tag{4.8}$$

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 9. *If* $\{P\}Prog\{Q\}$ *is totally correct, then it is partially correct, or,*

$$\models_{tot} \{P\}Prog\{Q\} \implies \models_{par} \{P\}Prog\{Q\}$$

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 proof rules for the instructions of the language:

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

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

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

$$(App)\frac{}{\{(\sigma,padding(U,n)^\dagger.\rho.padding(U,n))\}\ \textbf{q}\ n* =\ U\ \{(\sigma,\rho)\}}$$

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

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

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

$$(Weakness)\frac{P\sqsubseteq P'\wedge\{P'\}\textbf{S}\{Q'\}\wedge Q'\sqsubseteq Q}{\{P\}S\{Q\}}$$

Let S be the program defined in Example 5. In the following example we will prove the Hoare triple

$$\{True\to 1\}\ S\ \{(x\neq y)\to 1\}$$

Using a similar style as in [1], the triple can be proved as follows:

$$\{True\to 1\}\equiv\{True\to\langle 0|0\rangle\langle 0|0\rangle\}$$

$$\textbf{q}\ 0 := 0$$

$$\{True\to\rho_0\}\ \textbf{(Init)}$$

$$\textbf{q}\ 1 := 0$$

$$\{True\to\rho_1\}\ \textbf{(Init)}$$

$$\textbf{q}\ 0 * = H$$

$$\{True \to \rho_2\}\ \textbf{(App)}$$

$$x := \textbf{meas}\ q\ 0;$$

$$\{(x = 0 \to \rho_{30})\}\ \textbf{(Meas - 0 case)}$$

$$\textbf{if}\ x == 0\ \textbf{then q}\ 1 * = X\ \textbf{else skip}$$

$$\{(x = 0 \to \rho_{40})\}\ \textbf{(If)}$$

$$y := \textbf{meas}\ q\ 1$$

$$\{(x = 0\ \wedge\ y = 0) \to \rho_{5_{00}}\}\ \textbf{(Meas - 00 case)}$$

$$\{P[x = 0\ \wedge\ y = 0] = 0\}\ \textbf{Equiv}$$

$$\{(x = 0\ \wedge\ y = 1) \to \rho_{5_{01}}\}\ \textbf{(Meas - 01 case)}$$

$$\{P[x = 0\ \wedge\ y = 1] = 0.5\}\ \textbf{Equiv}$$

$$\{(x = 0 \to \rho_{31})\}\ \textbf{(Meas - 1 case)}$$

$$\textbf{if}\ x == 0\ \textbf{then q}\ 1 * = X\ \textbf{else skip}$$

$$\{(x = 0 \to \rho_{41})\}\ \textbf{(If)}$$

$$y := \textbf{meas}\ q\ 1$$

$$\{(x = 0\ \wedge\ y = 0) \to \rho_{5_{10}}\}\ \textbf{(Meas - 10 case)}$$

$$\{P[x = 1\ \wedge\ y = 0] = 0.5\}\ \textbf{Equiv}$$

$$\{(x = 0\ \wedge\ y = 1) \to \rho_{5_{11}}\}\ \textbf{(Meas - 11 case)}$$

$$\{P[x = 1\ \wedge\ y = 1] = 0\}\ \textbf{Equiv}$$

$$\{x \neq y \to 1\}\ \textbf{Imp}$$

Which concludes the proof.

4.7 A Use Case: Quantum Teleportation

We talked about Quantum Teleportation in Chapter 2, and we can notice that it is a good example of a quantum program that uses classical variables. Therefore, in this section we are going to reason about it using the logic defined in Section 4.6.

First, the following piece of code is written in our language to represent a case of Quantum Teleportation in which the qubit that we intend to teleport has the state $|1\rangle$:

Listing 4.1: Quantum teleportation in FY

```
new q;
new q;
new q;
// some manipulations to q 0
q 1 *= H;
q 1 2 *= CNOT;
q 0 1 *= CNOT;
q 0 *= H;
X0 :=meas 0;
X1 :=meas 1;
if X0 == 0 then
    if X1 == 1 then
        q 2 *= X
    else:
        skip
    end
else
    if X1 == 1 then
        q 2 *= Z;
        q 2 *= X;
    else:
        q 2 *= Z;
    end
end
```

4.7.1 Reasoning using Operational Semantics

We consider that the teleported qubit is in the state $\alpha|0\rangle + \beta|1\rangle$, where $\alpha, \beta \in \mathbf{C}$ and $|\alpha|^2 + |\beta|^2 = 1$, $|\alpha|^2$ represents the probability that the qubit will be measured to 0, and $|\beta|^2$ represents the probability of measuring 1. We expect the density matrix before executing the fourth line to be

$$\rho_0 = ((\alpha|0\rangle + \beta|1\rangle) \otimes |0\rangle \otimes |0\rangle)(\langle 0| \otimes \langle 0| \otimes (\alpha\langle 0| + \beta\langle 1|))$$

Following the operational semantics in Section 4.5, we start from the fourth line by applying a Hadamard gate on the teleported qubit, the classical state remains unchanged:

$$\rho_1 = (I \otimes H \otimes I)\rho_0(I \otimes H \otimes I)^\dagger$$

Then we move to next line, where we apply a $CNOT$ where the teleported qubit and the receiver qubit, where the receiver is the target, and the density matrix changes to:

$$\rho_2 = (I \otimes CNOT)\rho_1(I \otimes CNOT)^\dagger$$

Next will be apply $CNOT$ on the mediator and the teleported (as a target) qubits, leading to ρ_3:

$$\rho_3 = (CNOT \otimes I)\rho_2(CNOT \otimes I)^\dagger$$

In the end of Phase 2, we apply a Hadamard gate on the teleported qubit:

$$\rho_4 = (H \otimes I \otimes I)\rho_3(H \otimes I \otimes I)^\dagger$$

Then we enter Phase 3, that involves measuring the teleported qubit and the mediator, and we do this sequentially, giving us the following classical states associated with their density matrices:

$$\{(x_0 \to 0, x_1 \to 0) \to \rho_{4_{00}}, (x_0 \to 0, x_1 \to 1) \to \rho_{4_{01}},$$

$$(x_0 \to 1, x_1 \to 0) \to \rho_{4_{10}}, (x_0 \to 1, x_1 \to 1) \to \rho_{4_{11}}\}$$

where

$$\rho_{4_0} = (I \otimes I \otimes |0\rangle\langle 0|)\rho_4(I \otimes I \otimes |0\rangle\langle 0|)^\dagger$$

$$\rho_{4_1} = (I \otimes I \otimes |1\rangle\langle 1|)\rho_4(I \otimes I \otimes |1\rangle\langle 1|)^\dagger$$

$$\rho_{4_{00}} = (I \otimes |0\rangle\langle 0| \otimes I)\rho_{4_0}(I \otimes |0\rangle\langle 0| \otimes I)^\dagger$$

$$\rho_{4_{01}} = (I \otimes |0\rangle\langle 0| \otimes I)\rho_{4_1}(I \otimes |0\rangle\langle 0| \otimes I)^\dagger$$

$$\rho_{4_{10}} = (I \otimes |1\rangle\langle 1| \otimes I)\rho_{4_0}(I \otimes |1\rangle\langle 1| \otimes I)^\dagger$$

$$\rho_{4_{11}} = (I \otimes |1\rangle\langle 1| \otimes I)\rho_{4_1}(I \otimes |1\rangle\langle 1| \otimes I)^\dagger$$

The chained *if-statements* are applied to the classical states that satisfy the condition, so for instance, the $skip$ instruction is only applied to the first state $\{(x_0 \to 0, x_1 \to 0) \to ...\}$, leaving the density matrix unmodified, X is applied on q_2 for the state $\{(x_0 \to 0, x_1 \to 1) \to ...\}$ and so on. So the state afterwards is:

$$\{(x_0 \to 0, x_1 \to 0) \to \rho_{00}, (x_0 \to 0, x_1 \to 1) \to \rho_{01},$$

$$(x_0 \to 1, x_1 \to 0) \to \rho_{10}, (x_0 \to 1, x_1 \to 1) \to \rho_{11}\}$$

where

$$\rho_{00} = \rho_{4_{00}}$$

$$\rho_{01} = (X \otimes I \otimes I)\rho_{4_{01}}(X \otimes I \otimes I)^\dagger$$

$$\rho_{10} = (Z \otimes I \otimes I)\rho_{4_{10}}(Z \otimes I \otimes I)^\dagger$$

$$\rho_{11} = (X \otimes I \otimes I)(Z \otimes I \otimes I)\rho_{4_{11}}(Z \otimes I \otimes I)^\dagger(X \otimes I \otimes I)^\dagger$$

For each one of the classical states above, we can find the state of q_2, i.e the probabilities of measuring 0 and 1, by applying the formula $P[i] = \langle i| \rho |i\rangle$. We start with the first state:

$$\begin{aligned} P[q_2\ measures\ to\ 0] &= \langle 000| \rho_{00} |000\rangle \\ &= \langle 000| (I \otimes I \otimes |0\rangle\langle 0|)(I \otimes |0\rangle\langle 0| \otimes I)\rho_4(I \otimes |0\rangle\langle 0| \otimes I)^\dagger(I \otimes I \otimes |0\rangle\langle 0|)^\dagger |000\rangle \\ &= |\alpha|^2 \end{aligned}$$

and

$$\begin{aligned} P[q_2\ measures\ to\ 1] &= \langle 100| \rho_{00} |001\rangle \\ &= \langle 100| (I \otimes I \otimes |0\rangle\langle 0|)(I \otimes |0\rangle\langle 0| \otimes I)\rho_4(I \otimes |0\rangle\langle 0| \otimes I)^\dagger(I \otimes I \otimes |0\rangle\langle 0|)^\dagger |001\rangle \\ &= |\beta|^2 \end{aligned}$$

For the second case we have:

$$\begin{aligned}
P[q_2\ measures\ to\ 0] &= \langle 010|\ \rho_{01}\ |010\rangle \\
&= \langle 010|\ (X\otimes I\otimes I)(I\otimes I\otimes |1\rangle\langle 1|)(I\otimes |0\rangle\langle 0|\otimes I)\rho_4(I\otimes |0\rangle\langle 0|\otimes I)^\dagger(I\otimes I\otimes |1\rangle\langle 1|)^\dagger \\
&(X\otimes I\otimes I)^\dagger\ |010\rangle \\
&= |\alpha|^2
\end{aligned}$$

and

$$\begin{aligned}
P[q_2\ measures\ to\ 1] &= \langle 110|\ \rho_{01}\ |011\rangle \\
&= (X\otimes I\otimes I)(I\otimes I\otimes |1\rangle\langle 1|)(I\otimes |0\rangle\langle 0|\otimes I)\rho_4(I\otimes |0\rangle\langle 0|\otimes I)^\dagger(I\otimes I\otimes |1\rangle\langle 1|)^\dagger(X\otimes I\otimes I)^\dagger \\
&= |\beta|^2
\end{aligned}$$

Similarly, we can prove that q_2 keeps the distribution $|\alpha|^2, |\beta|^2$ for the third and fourth case, which is the same as q_0, hence the algorithm is correct. The detailed calculations are in the appendix.

4.7.2 Reasoning using Hoare Logic

The precondition before executing the algorithm (starting from line 4), is

$$q_0 = \alpha\,|0\rangle + \beta\,|1\rangle \wedge q_1 = |0\rangle \wedge q_2 = |0\rangle$$

and we want to prove that the postcondition

$$q_2 = \alpha\,|0\rangle + \beta\,|1\rangle$$

holds after executing the program in Figure 2.2. We can prove it as follows:

$$\{True \to \rho_0\}\ (\textbf{App})$$

$$\textbf{q}\ 1 * = H;$$

$$\{True \to \rho_1\}\ (\textbf{App})$$

$$\textbf{q}\ 1\ 2 * = CNOT;$$

$$\{True \to \rho_2\}\ (\textbf{App})$$

$$\textbf{q}\ 0\ 1 * = CNOT;$$

$$\{True \to \rho_3\}\ (\textbf{App})$$

$$\textbf{q}\ 0 * = H;$$

$$\{True \to \rho_4\}\ (\textbf{App})$$

$$x_0\ \textbf{:=meas}\ 0;$$

$$\{True[x_0/0] \to \rho_{4_0}\}\ (\textbf{Meas - 0 case})$$

$$x_1\ \textbf{:=meas}\ 1;$$

$$\{True[x_0/0, x_1/0] \to \rho_{4_{00}}\}\ (\textbf{Meas - 00 case})$$

$$\{(x_0 == 0 \wedge x_1 == 0) \to \rho_{4_{00}}\}\ (\textbf{Equiv})$$

$$\textbf{if}\ x_0 == 0\ \textbf{then}...\textbf{end}$$

$$\{(x_0 == 0 \wedge x_1 == 0) \to \rho_{00}\}\ (\textbf{If, App})$$

$$\{P[M(q_2) = 0] = \langle 000|\, \rho_{00}\, |000\rangle = |\alpha|^2$$
$$P[M(q_2) = 1] = \langle 100|\, \rho_{00}\, |001\rangle = |\beta|^2\}\ (\textbf{Equiv})$$

$$\{True[x_0/0, x_1/1] \to \rho_{4_{00}}\}\ (\textbf{Meas - 01 case})$$

$$\{(x_0 == 0 \wedge x_1 == 1) \to \rho_{4_{01}}\}\ (\textbf{Equiv})$$

$$\textbf{if}\ x_0 == 0\ \textbf{then}...\textbf{end}$$

$$\{(x_0 == 0 \wedge x_1 == 1) \to \rho_{01}\}\ (\textbf{If, App})$$

$$\{P[M(q_2) = 0] = \langle 010|\, \rho_{01}\, |010\rangle = |\alpha|^2$$
$$P[M(q_2) = 1] = \langle 110|\, \rho_{01}\, |011\rangle = |\beta|^2\}\ (\textbf{Equiv})$$

$$\{True[x_0/1] \to \rho_{4_1}\}\ (\textbf{Meas - 1 case})$$

$$x_1\ \textbf{:=meas}\ 1;$$

$$\{True[x_0/1, x_1/0] \to \rho_{4_{10}}\}(\textbf{Meas - 10 case})$$

$$\{(x_0 == 0 \wedge x_1 == 0) \to \rho_{4_{10}}\}\ (\textbf{Equiv})$$

if $x_0 == 0$ **then**...**end**

$\{(x_0 == 1 \wedge x_1 == 0) \rightarrow \rho_{10}\}$ (**If, App**)

$$\{P[M(q_2) = 0] = \langle 001| \rho_{10} |100\rangle = |\alpha|^2$$
$$P[M(q_2) = 1] = \langle 101| \rho_{10} |101\rangle = |\beta|^2\}$$ (**Equiv**)

$\{True[x_0/1, x_1/1] \rightarrow \rho_{4_{00}}\}$(**Meas - 11 case**)

$\{(x_0 == 1 \wedge x_1 == 1) \rightarrow \rho_{4_{11}}\}$(**Equiv**)

if $x_0 == 0$ **then**...**end**

$\{(x_0 == 1 \wedge x_1 == 1) \rightarrow \rho_{01}\}$ (**If, App**)

$$\{P[M(q_2) = 0] = \langle 011| \rho_{11} |011\rangle = |\alpha|^2$$
$$P[M(q_2) = 1] = \langle 111| \rho_{11} |111\rangle = |\beta|^2\}$$(**Equiv**)

5

Mechanization in Coq

Contents

The practical part of this thesis involves implementing the language in Chapter 4, using the theorem prover Coq [26]. In this section, we will show our design decisions and choices.

5.1 Logic Mechanization

The theorem prover Coq [26] is a formal proof management tool that facilitates representing mathematical definitions, concepts theorems and proofs by enabling machine checking. Coq has been used to formally verify many protocols. Like Coq, there are other formal verification tools, like Isabelle. We chose Coq to implement our project because of the rich documentation (especially in the Software Foundations book [27]), and to reuse the work of QWIRE in [28], specifically the mathematical axiomatic definition of real, complex numbers, matrices and quantum computing.
The development process constituted of following phases:

1. Implementing the syntax of the language as an imperative language, by making use of the work in [27].
2. Implementing the concept of Quantum-classical state.
3. Implementing the semantics of the language.
4. Implementing Quantum-classical assertions, and the formula of expectation.
5. Implementing the proof system defined in Chapter 4.
6. Proving the correctness of the system's rules.
7. Starting Proving the soundness and the completeness of the system.
8. Writing some examples in our language and using the proof system to reason about them in Coq.

5.2 Implementation Details

The complete development of FY, the work of this thesis, consists of:

- 13 Coq files
- 3816 lines code
- 123 Definitions, Fixpoints and Inductives.
- 274 Theorems and Lemmas.

We can break down some Coq files as follows:

5.2.1 Syntax.v

In this file we define the syntax of FY, the arithmetic and boolean expressions and its syntactic representation.
Arithmetic expressions in FY are defined inductively to include the ones defined in Section 4.1

```
Inductive arith_exp : Type :=
  | AId (x : string)
  | ANum (n : nat)
  | APlus (a1 a2 : arith_exp)
  | AMinus (a1 a2 : arith_exp)
  | AMult (a1 a2 : arith_exp)
  | ADiv (a1 a2 : arith_exp).
```

and we define some syntactical notations to make writing expressions easier:

```
Notation "x - y" := (AMinus x y) (in custom com at level 50, left associativity).
Notation "x + y" := (APlus x y) (in custom com at level 50, left associativity).
Notation "x * y" := (AMult x y) (in custom com at level 40, left associativity).
Notation "x / y" := (ADiv x y) (in custom com at level 40, left associativity).
```

Similarly we define the boolean expressions and quantum gates:

```
Inductive bool_exp : Type :=
  | BTrue
  | BFalse
  | BEq (a1 a2 : arith_exp)
  | BLe (a1 a2 : arith_exp)
  | BNot (b : bool_exp)
  | BAnd (b1 b2 : bool_exp)
  | BOr (b1 b2 : bool_exp).
Inductive gate_exp : Type :=
  | GH
  | GX
  | GY
  | GZ
  | GI
  | GCNOT.
```

And its notational syntax is defined as follows:

```
Notation "'true'" := BTrue (in custom com at level 0).
Notation "'false'" := BFalse (in custom com at level 0).
Notation "x <= y" := (BLe x y) (in custom com at level 70, no associativity).
Notation "x == y" := (BEq x y) (in custom com at level 70, no associativity).
Notation "x && y" := (BAnd x y) (in custom com at level 80, left associativity).
Notation "x || y" := (BOr x y) (in custom com at level 80, left associativity).
Notation "'~' b" := (BNot b) (in custom com at level 75, right associativity).
```

Then we defined the language instructions as the following:

```
Inductive com : Type :=
  | CSkip
  | CAsgn (x : string) (a : arith_exp)
  | CMeas (x : string) (q : nat)
  | CInit
  | CAppOne (q : nat) (U : gate_exp)
  | CAppTwo (q1 : nat) (q2 : nat) (U : gate_exp)
  | CSeq (c1 c2 : com)
  | CIf (b : bool_exp) (c1 c2 : com)
  | CWhile (b : bool_exp) (c : com).
```

And its syntactical representation as defined in Section 4.1:

```
Notation "'skip'" :=
    CSkip (in custom com at level 0) : com_scope.
Notation "x ':=' y" :=
    (CAss x y)
       (in custom com at level 0, x constr at level 0,
        y at level 40, no associativity) : com_scope.
Notation "x ':=meas' n" :=
    (CMeas x n)
       (in custom com at level 0, x constr at level 0,
        n constr at level 77, no associativity) : com_scope.
Notation "'new_qubit'" :=
    (CInit)
       (in custom com at level 0, no associativity) : com_scope.
Notation "'q' n *= U" :=
    (CAppOne n U)
```

```
       (in custom com at level 0, n constr at level 0,
       U at level 85, no associativity) : com_scope.
Notation "'q' n m *= U" :=
    (CAppTwo n m U)
       (in custom com at level 0, n constr at level 0,
        m constr at level 0, U at level 85,
        no associativity) : com_scope.
Notation "x ; y" :=
    (CSeq x y)
      (in custom com at level 90, right associativity) : com_scope.
Notation "'if' x 'then' y 'else' z 'end'" :=
    (CIf x y z)
      (in custom com at level 89, x at level 99,
       y at level 99, z at level 99) : com_scope.
Notation "'while' x 'do' y 'end'" :=
    (CWhile x y)
       (in custom com at level 89, x at level 99, y at level 99) : com_scope.
```

The following example shows how a program could be written in Coq:

```
Example Prog : com :=
  <{ new_qubit;
     new_qubit;
     q 0 *= GH;
     X :=meas 0%nat;
     if X == (0 % nat) then
        q 1 *= GX
     else
        skip
     end;
     Y :=meas 1%nat
  }>.
```

5.2.2 State.v

In this module, we defined the concept of Quantum classical state as a list of pairs (maps, matrices), and contains a variety of Fixpoints to describe how the state updates, and some theorems about them.

First, we need to define the semantical evaluation of arithmetic, boolean expressions and quantum gates in order to be used in State's Calculations:

```
Fixpoint aeval (st : total_map nat)
         (a : arith_exp) : nat :=
  match a with
  | ANum m => m
  | AId x => st x
  | <{a1 + a2}> => (aeval st a1) + (aeval st a2)
  | <{a1 - a2}> => (aeval st a1) - (aeval st a2)
  | <{a1 * a2}> => (aeval st a1) * (aeval st a2)
  | <{a1 / a2}> => (aeval st a1) / (aeval st a2)
  end.
Fixpoint beval (st : total_map nat) (b : bool_exp) : bool :=
  match b with
  | <{true}> => true
  | <{false}> => false
  | <{a1 == a2}> => (aeval st a1) =? (aeval st a2)
  | <{a1 <= a2}> => (aeval st a1) <=? (aeval st a2)
  | <{~ b1}> => negb (beval st b1)
  | <{b1 && b2}> => andb (beval st b1) (beval st b2)
  | <{b1 || b2}> => orb (beval st b1) (beval st b2)
  end.
Fixpoint geval (g : gate_exp) : Unitary _ :=
  match g with
  | GI => I 2
  | GH => H
  | GX => X
  | GZ => Z
  | GY => Y
  | GCNOT => CNOT
  end.
```

Then we define the concept of Quantum Classical State:

```
Definition State (dim: nat): Type := list ((total_map nat)*(Unitary (2^dim))).
```

Then, we defined the helper functions in Section 4.5:

```
Fixpoint padding (dim : nat) (qubit : nat) (U : Unitary 2) : Unitary (2^dim) :=
  match dim with
  | O%nat => (if qubit =? 0%nat then U else (I 2))
  | S dim' => (padding dim' qubit U) ⊗ (if qubit =? n then U else I 2)
  end.
Fixpoint GetMeasurementBasis (dim : nat) (meas_qubit : nat) (isZero : bool)
  : Unitary (2^dim) :=
  match dim with
    | 0%nat => if qubit =? dim then (if isZero then |0⟩⟨0| else |1⟩⟨1|) else (I 2)
    | S dim' => (if qubit =? dim then
         (if isZero
          then ((GetMeasurementBasis dim' meas_qubit isZero) ⊗ (|0⟩⟨0|))
         else ((GetMeasurementBasis dim' meas_qubit isZero) ⊗ (|1⟩⟨1|)))
         else (GetMeasurementBasis dim' meas_qubit isZero) ⊗ (I 2))
  end.
```

To be used later in the operational semantics, we define the operations that specifies how a state updates, which happen when there is an assignement, qubit initialization, unitary transformation and measurement.

```
Fixpoint UpdateStateAssign (n : nat) (state: State n) (x: string)
                           (a: arith_exp) : State n :=
  match state with
  | [] => []
  | st :: l => (pair (x !-> (aeval (fst st) a); fst st) (snd st))
                                    :: (UpdateStateAssign n l x a)
  end.
Fixpoint UpdateStateInit (n : nat) (state: State n) : State (n + 1%nat) :=
  match state with
  | [] => []
  | st :: l => if n =? 0%nat then
    (pair (fst st) (|0⟩⟨0|)) :: (UpdateStateInit n l)
    else
    (pair (fst st) (|0⟩ ⊗ (snd st) ⊗ ⟨0|)) :: (UpdateStateInit n l)
  end.

Fixpoint UpdateStateApply (n : nat) (state: State n) (qubit : nat) (U: gate_exp): State n:=
  match state with
```

```
  | [] => []
  | st :: l => match U with
      | GCNOT => (pair (fst st) ((padding (n - 2%nat)
      qubit (geval U)) × (snd st)
      × (padding (n - 2%nat) qubit (geval U))†) )
      :: (UpdateStateApply n l qubit U)
      | _ => (pair (fst st) ((padding (n - 1%nat) qubit (geval U))
      × (snd st) × (padding (n - 1%nat) qubit (geval U))†) )
      :: (UpdateStateApply n l qubit U)
    end
  end.
Fixpoint UpdateStateMeasure (n: nat)  (state: State n) (x : string) (qubit : nat) : State n :=
  match state with
  | [] => []
  | st :: l => (pair (x !-> 0%nat; fst st)
     ((GetMeasurementBasis (n - 1%nat) qubit true) × (snd st) ×
     (GetMeasurementBasis (n - 1%nat) qubit true)†)) ::
     (pair (x !-> 1%nat; fst st)
     ((GetMeasurementBasis (n - 1%nat) qubit false) × (snd st) ×
     (GetMeasurementBasis (n - 1%nat) qubit false)†)) ::
     (UpdateStateMeasure n l x qubit)
  end.
```

Finally, we implemented an auxiliary operation that filters a state to its elements that satisfy a Boolean expression, to be used later for if and $while$ rules.

```
Fixpoint Filter (n : nat) (state: State n) (b : bool_exp): State n :=
  match state with
  | [] => []
  | st :: l => if (beval (fst st) b)
               then (st :: (Filter n l b))
               else (Filter n l b)
  end.
```

5.2.3 Semantics.v

In this file, we implement our Operational semantics (defined in Section 4.5). The Inductive $ceval$ evaluates not only the state change, but also the space dimension, the first two arguments refers to the dimension before and after applying the instruction, and the fourth and fifth arguments are the states before and after the instruction.

```
Inductive ceval : nat -> nat -> com
    -> State 1%n -> State 1%n -> Prop :=
  | E_Skip : forall n st,
      ceval n n <{ skip }> st st
  | E_Asgn : forall n st a x,
      ceval n n <{ x := a }> st (UpdateStateAssign n st x a)
  | E_Init : forall n st,
      ceval n (n + 1%nat) <{ new_qubit }> st (UpdateStateInit n st)
  | E_AppOne : forall n st U m,
      ceval n n <{ q m *= U }> st (UpdateStateApply n st m U)
  | E_AppTwo : forall n st U m r,
      ceval n n <{ q m r *= U }> st (UpdateStateApply n st m U)
  | E_Meas : forall n st x m,
      ceval n n <{ x :=meas m }> st (UpdateStateMeasure n st x m)
  | E_Seq : forall n n' n'' c1 c2 st st' st'',
      ceval n n' c1 st st' ->
      ceval n' n'' c2 st' st'' ->
      ceval n n'' <{ c1 ; c2 }> st st''
  | E_If : forall n n' st st' st'' b c1 c2,
      ceval n n' c1 (Filter n st b) st' ->
      ceval n n' c2 (Filter n st (BNot b)) st'' ->
      ceval n n' <{ if b then c1 else c2 end }> st (st' ++ st'')
  | E_WhileTrue : forall n n' st st' st'' b c,
      ceval n n' c (Filter n st b) st' ->
      ceval n n' <{ while b do c end }> (Filter n st b) st' ->
      ceval n n' <{ while b do c end }> st' st''
  | E_WhileFalse : forall n n' st b c,
      ceval n n' <{ while b do c end }> (Filter n st (BNot b)) (Filter n st (BNot b)).
```

5.2.4 Assertion.v

In this module, we implement the definition of a quantum classical assertion, and some ulterior definitions that helps implementing the proof system later.

```
Definition Assertion (dim: nat) : Type := (total_map nat) * (bool_exp * (Unitary (2^dim))).
```

Then we implement some getters and a constructor:

```
Definition StateOf {n: nat} (a: Assertion n) : total_map nat := fst a.
Definition PropOf {n: nat} (a: Assertion n): bool_exp := fst (snd a).
Definition DensityOf {n: nat} (a: Assertion n) := snd ( snd a).
Definition AssertionOf (n: nat) (st: total_map nat)
           (prop: bool_exp) (U: Unitary (2^n)) : Assertion n := (st, (prop, U)).
```

Then we implemented the definition of *Expectation of satisfaction* in Definition 18

```
Fixpoint Expectation (dims dima : nat)
     (state: State dims)
     (a: Assertion dima) : R :=
    match state with
    | [] => 0%R
    | st :: l =>
        if beval (mergeMaps (fst st) (StateOf a)) (PropOf a) then
             Rplus (fst (trace ((complement ns na (snd st))
             × (complement na ns (DensityOf a)))))
             (Expectation dims dima l a)
        else (Expectation dims dima l a)
    end.
```

where the function *complement* is a helper function that applies a Kronecker product of Identity operators, in order to equalize the dimensions and make the multiplication operation possible.
In addition to the previous definitions, we added some concepts to help in the next module. *apply_sub* simulates how an Pre-condition would be before applying a unitary transformation, and similarly *init_sub*,*asgn_sub* and *meas_sub* with respect to initialization, assignment and measurement.

```
Definition init_sub (n: nat) (P : Assertion n) : Assertion (n - 1) :=
    pair (StateOf P) (pair (PropOf P) (pre_init n (DensityOf P))).
Definition apply_sub n (U: Unitary (2^n)) (P : Assertion n) : Assertion n :=
  pair (StateOf P) (pair (PropOf P) ((padding n m U)†
```

```
      × (DensityOf P) × (padding n m U))).
Definition asgn_sub {n} (P: Assertion n) (x: string)
 (e: nat) : Assertion n := (pair (x !-> e; _ !-> 0%nat)
 (pair (PropOf P) (DensityOf P))).
Definition meas_sub {n} (P: Assertion n) (x: string) (v: nat)
  (m : nat) : Assertion n :=
  ( (x !-> v; StateOf P) , ( (PropOf P),
  (GetMeasurementBasis (n - 1%nat) m (v =? 0%nat))
  × (DensityOf P)
  × (GetMeasurementBasis (n - 1%nat) m (v =? 0%nat))†)).
```

where *pre_init* implements the function:

$$pre\_init(\rho, dim) = (\langle 0| \otimes I_{2^{dim-1}}).\rho.(|0\rangle \otimes I_{2^{dim-1}})])$$

We also implemented the weakness relation between two assertions as follows:

```
Definition weaker (ns na1 na2 : nat)
    (state: State ns)
    (assert1: Assertion na1)
    (assert2: Assertion na2) : Prop :=
      (Expectation ns na1 state assert1)
      <= (Expectation ns na2 state assert2).
```

And the concept when the classical part of the first assertion implies the classical part of the second one, given that they both have equal density matrix:

```
Definition classicalPropsImp (np nq: nat)(P : Assertion np)
  (Q : Assertion nq) : Prop := forall st,
  (DensityOf P) = (DensityOf Q) ->
  beval (mergeMaps st (StateOf P)) (PropOf P) = true ->
  beval (mergeMaps st (StateOf Q)) (PropOf Q) = true.
```

where the function *mergeMaps* is a utilitary function that merges two total maps.

To implement *If* rule in Section 4.6, we need to have a concept to describe $b \wedge P$ where b is a boolean expression. for this purpose, we implemented the following definition:

```
Definition pre_if_assertion_boolean {n} (P: Assertion n) (b: bool_exp)
 : Assertion n := (pair (StateOf P)
 (pair (BAnd b (PropOf P)) (DensityOf P))).
```

5.2.5 Logic.v

In this file, we implemented Hoare triple as specified in Section 4.6, and the theorems that describe the proof system in Section 4.6.

```
Definition hoare_triple {dimp dimq: nat}
    (P : Assertion dimp) (c : com) (Q : Assertion dimq): Prop :=
    forall dims1 dims12
    (st1: State dims1)
    (st2: State dims2),
    (ceval dimp dimp c st1 st2) ->
    (Expectation dims dimp st1 P) <= (Expectation dims2 dimq st2 Q).
```

The following Theorems are the rules of the proof system in Section 4.6:

```
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'.
```

5.2.6 Soundness.v

We started (but did not finish) proving proving that the logic is sound and complete formally in the file *Soundness.v*, as Feng and Ying [1] stated, the logic in Section 4.6 is sound and complete.
First we start by formalizing the Derivability:

```
Inductive derivable:
    nat -> nat ->
```

```
  Assertion 1 -> com -> Assertion 1 -> Type :=
| H_Skip : forall n (P: Assertion n),
  derivable n n P <{ skip }> P
| H_Init : forall n (P: Assertion n),
  derivable n (n + 1)%nat (init_sub n P) <{ new_qubit }> P
| H_App : forall n G (P: Assertion n) m,
  derivable n n (apply_sub n m (geval G) P) <{ q m *= G }> P
| H_Asgn : forall n (P: Assertion n) x e,
  derivable n n (AssertPreAsgn P x e) <{ x := e }> P
| H_Meas_0 : forall n m (P: Assertion n) x,
  derivable n n (AssertPreMeas P x 0 m) <{ x :=meas m }> P
| H_Meas_1 : forall n m (P: Assertion n) x,
  derivable n n (AssertPreMeas P x 1 m) <{ x :=meas m }> P
| H_Seq : forall np nq nr (P: Assertion np) c
                (Q: Assertion nq) d (R: Assertion nr),
  derivable np nq P c Q ->
  derivable nq nr Q d R ->
  derivable np nr P <{ c;d }> R
| H_If : forall np nq (P: Assertion np)
               (Q: Assertion nq) b c1 c2,
  derivable np nq (AssertPreIfTrue P b) c1 Q ->
  derivable np nq (AssertPreIfTrue P (BNot b)) c2 Q ->
  derivable np nq P <{if b then c1 else c2 end}> Q
| H_While : forall n (P: Assertion n) b c,
  derivable n n (AssertPreIfTrue P b) c P ->
  derivable n n P <{while b do c end}> (AssertPreIfTrue P (BNot b))
| H_Weakness : forall np nq np' nq'
      (P: Assertion np) (Q: Assertion nq)
      (P': Assertion np') (Q': Assertion nq') c,
  derivable np nq P c Q ->
  (forall ns (st: State ns), weaker ns np np' st P' P) ->
  (forall ns (st: State ns), weaker ns nq nq' st Q Q') ->
  derivable np' nq' P' c Q'.
```

A logic is sound when every derivable rule is valid, While it is complete when every valid rule is derivable. we formalize it as follows:

```
Theorem logic_sound : forall np nq (P: Assertion np) c (Q: Assertion nq),
  derivable np nq P c Q -> valid np nq P c Q.

Theorem logic_complete: forall np nq (P: Assertion np) c (Q: Assertion nq),
  valid np nq P c Q -> derivable np nq P c Q.
```

5.2.7 Utils.v

In this file, we included all the lemmas, facts, axioms and theorems that were used in the proofs, and are related to Maps, Real numbers, Complex numbers, matrices, and Quantum computing foundations. We took most of the elements from the series Software foundation [27] and from Robert Rand et al's work in [28]. We added some more theorems that proves some properties of traces like distributive and cyclic properties, and some useful facts about maps. For instance:

```
Lemma matrices_distributive: forall {n} (m1 m2 m3: Matrix n n),
       m1 × (m2 + m3) = m1 × m2 + m1 × m3.
Lemma equal_traces_comm: forall {n} (p1 p2 p3: Matrix n n),
    trace (m1 × m2) = trace (m2 × m1).
```

5.2.8 MatricesConverter.py

We developed a Python script that solves a quantum expression (with kets, bras and gates), into a matricial form that facilitates calculations later. The scripts require a manual supervision by the developer who inputs the expression and the name of the lemma.

Example 10. *If the developer entered the expression:*

$$((I2 \otimes CNOT) \times (CNOT \otimes I2) \times (H \otimes I2 \otimes I2) \times (|\,0\rangle \otimes (|\,0\rangle \otimes (|\,0\rangle\langle 0\,|) \otimes\langle 0\,|)\langle 0\,|) \times (H \otimes I2 \otimes I2)\dagger \times (CNOT \otimes I2)\dagger \times (I2 \otimes CNOT)\dagger)$$

and decided to name it $H4$, then the script will generate the following:

```
Lemma H4:
(I 2 ⊗ CNOT ⊗ I 2 ⊗ I 2 × (CNOT ⊗ I 2 ⊗ I 2 ⊗ I 2)
× (H ⊗ I 2 ⊗ I 2 ⊗ I 2) × (| 0 ⟩ ⊗ (| 0 ⟩ ⊗ |0⟩⟨0| ⊗ ⟨0|) ⊗ ⟨0|)
× (H ⊗ I 2 ⊗ I 2 ⊗ I 2) † × (CNOT ⊗ I 2 ⊗ I 2 ⊗ I 2) †
× (I 2 ⊗ CNOT ⊗ I 2 ⊗ I 2) †)
 = l2M [[1/2;0;0;0;0;0;0;1/2];
```

```
[0;0;0;0;0;0;0;0];
[0;0;0;0;0;0;0;0];
[0;0;0;0;0;0;0;0];
[0;0;0;0;0;0;0;0];
[0;0;0;0;0;0;0;0];
[0;0;0;0;0;0;0;0];
[1/2;0;0;0;0;0;0;1/2]].
Proof.
(* BY PYTHON SCRIPT *)
Admitted.
```

5.2.9 Examples

In our work, we tried to prove some known algorithms. In the following we will show a small example of how can we reason about *Coin toss* program using our model:

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

We can specify this program using directly the operational semantics as follows:

```
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| )].
```

If we want to reason about the statement $x = 0$ using Hoare logic, we can define assertions *pre* and *post* are defined as follows:

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

Then it is sufficient to prove the following theorem to assure that the tossing a coin will give ”heads” with a probability 0.5:

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

We also implemented and proved a series of intermediate theorems and lemmas, to help us proving the theorems above. Nonetheless, there is still some theorems (mostly related to mathematical facts) that were left unproven, although we were able to prove them informally.

The work also contains more examples to reason about like Coin Tossing, GHZ, Quantum Teleportation algorithm, Deutsch-Josza Algorithm, and Grover algorithm, but some of them are incomplete. The code is available in https://github.com/sr-lab/quantum-hoare-logic. The following table illustrates some statistics about the project.

Table 5.1: A summary of files content in FY

File	LoC	Theorems	Defintions
Syntax.v	103	0	4
Semantics.v	40	0	1
State.v	112	0	13
Logic.v	868	46	6
Assertion.v	88	2	15
Soundness.v	71	2	2
Utils.v	1674	175	67
Examples.v	234	20	8
CoinToss.v	88	2	4
GHZ.v	304	0	1
Teleportation.v	64	0	1
Grover.v	80	0	1
Totals	3816	274	123

6

Conclusion

Contents

In this section, we are going to list the results, the challenges that face our solution, and how could they be handled in the future.

6.1 Results

Our thesis's work presented a contribution towards facilitating the development verifiable quantum programs with quantum and classical variables. We implemented in Coq the concepts presented in [1], and started formally proving the soundness and completeness of the logic in Section 4.6. For the future, we have a clear road-map (Section 6.3) to continue developing this project towards making it easier to be used by developers.

6.2 Limitations of the Current Solution

There are some issues in some parts of our solution, like the Syntax or the Proof system's implementation, in the following we list some of those issues:

- 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 [28].

In the next section, we are going to list some of the suggestions to address those issues.

6.3 Suggestions and Future work

There are many directions that could be taken to continue this work in the future, but with different priorities. The most important direction in the short term is to prove the theorems that are left without a proof. In the following, we list some of additions that would enhance the work for this thesis.

6.3.1 Soundness and Completeness

In [1], Feng and Ying used the concept of Weakest Liberal precondition to prove the partial and total correctness of their logic. Hence, the logic defined in Section 4.6 is sound and complete. As future work, we can formalize the proof using the theorems that we already have proven.

6.3.2 Automating proofs

Proving some of the programs using the proof system in Section 4.6 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.

6.3.3 Discard Operation

Upon measuring a qubit, it collapses into one of the states of the space's basis (either $|0\rangle$ or $|1\rangle$), and then it will never leave it again, hence, this qubit is collapsed and will not change its state if it is measured again. This raises the necessity of implementing a mechanism to discard already measured qubits in the upcoming operations.

6.3.4 The Dimensional Explosion Problem

When a system contains 1 qubit, then the matrices in states (or assertions) are Unitarian with 2×2 dimensions. When we add another qubit, it becomes 4×4, and 8×8 when there are 3 qubits, hence, the dimensions in a system with n qubits are $2^n \times 2^n$. using a vector states to represent the quantum state instead of Density matrices will only reduce the spacial complexity by half. In Utils.v, matrices are defined as functions, where entries are 0 by default. This definition saves a significant amount of space, given that those matrices tend to be sparse. However, the multiplication operations and Kronecker products still may take a long time to be executed. To tackle this problem, Liu et al in [29] used NumPy to help executing matrices operation along with Isabelle.

6.3.5 Enhancing the Arithmetic, Boolean and Matricial Expressiveness

The language supports a variety of expressions like addition, subtraction, multiplication, division, conjunction, disjunction and negation. Also, it is possible to represent some mathematical order relations. However, some algorithms (like Shor) require additional expressions like arithmetic modulation and congruence. One another limitation in the language the ability to apply gates on non-adjacent qubits, i.e, given a system with 3 qubits, it is impossible to apply a $CNOT$ gate to the first and the third qubits, or to the second as a control qubit, and the first as a target. In order to support such operations, we can implement the Swap gate as follows:

$$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and then, the following equation holds:

$$SWAP\,|x\rangle\,|y\rangle = |y\rangle\,|x\rangle$$

which implies the following equation:

$$CNOTq_0q_2 = (SWAP \otimes I)(I \otimes CNOT)(I \otimes SWAP)q_0q_2$$

There is another limitation in the language, which is the inability to initialize or measure multiple qubits at once. In [1], Feng and Ying suggests some syntactic sugar to be implemented in the language regarding this issue.

6.3.6 Interoperability with External Quantum Computation Platforms

The language in this thesis is embedded in Coq, while it is mostly common to use the tools and languages in Chapter 2 to develop quantum programs. Therefore, it would be useful to develop a layer between FY and one of those tools. This layer will help developers to verify their programs before executing them on a real quantum computer (usually available via a cloud service), saving them the waiting time each time a change is needed.

Bibliography

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

[2] R. Rand, K. Hietala, and M. Hicks, "Formal verification vs. quantum uncertainty," in *SNAPL*, 2019.

[3] L. H. Ramshaw, "Formalizing the analysis of algorithms." STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, Tech. Rep., 1979.

[4] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[5] R. Chadha, P. Mateus, and A. Sernadas, "Reasoning about imperative quantum programs," *Electronic Notes in Theoretical Computer Science*, vol. 158, pp. 19–39, 2006.

[6] Y. Kakutani, "A logic for formal verification of quantum programs," in *Annual Asian Computing Science Conference*. Springer, 2009, pp. 79–93.

[7] M. Ying, "Floyd–hoare logic for quantum programs," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 33, no. 6, pp. 1–49, 2012.

[8] D. E. Deutsch, "Quantum computational networks," *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, vol. 425, no. 1868, pp. 73–90, 1989.

[9] M. Ross and M. Oskin, "Quantum computing," *Commun. ACM*, vol. 51, no. 7, p. 12–13, Jul. 2008. [Online]. Available: https://doi.org/10.1145/1364782.1364787

[10] N. Brunner, D. Cavalcanti, S. Pironio, V. Scarani, and S. Wehner, "Bell nonlocality," *Rev. Mod. Phys.*, vol. 86, pp. 419–478, Apr 2014. [Online]. Available: https://link.aps.org/doi/10.1103/RevModPhys.86.419

[11] A. K. Ekert, "Quantum cryptography based on Bell's theorem," *Physical review letters*, vol. 67, no. 6, pp. 661–663, Aug. 1991.

[12] A. Einstein, B. Podolsky, and N. Rosen, “Can quantum-mechanical description of physical reality be considered complete?” *Phys. Rev.*, vol. 47, pp. 777–780, May 1935. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRev.47.777

[13] J. S. Bell, “On the einstein podolsky rosen paradox,” *Physics Physique Fizika*, vol. 1, pp. 195–200, Nov 1964. [Online]. Available: https://link.aps.org/doi/10.1103/PhysicsPhysiqueFizika.1.195

[14] P. W. Shor, “Algorithms for quantum computation: discrete logarithms and factoring,” in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 124–134.

[15] F. Arute, K. Arya, R. Babbush, D. Bacon, J. Bardin, R. Barends, R. Biswas, S. Boixo, F. Brandao, D. Buell, B. Burkett, Y. Chen, Z. Chen, B. Chiaro, R. Collins, W. Courtney, A. Dunsworth, E. Farhi, B. Foxen, and J. Martinis, “Quantum supremacy using a programmable superconducting processor,” *Nature*, vol. 574, pp. 505–510, 10 2019.

[16] P. A. M. Dirac, “A new notation for quantum mechanics,” in *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 35, no. 3. Cambridge University Press, 1939, pp. 416–418.

[17] E. Knill, “Conventions for quantum pseudocode,” Los Alamos National Lab., NM (United States), Tech. Rep., 1996.

[18] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters, “Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels,” *Physical review letters*, vol. 70, no. 13, p. 1895, 1993.

[19] L. K. Grover, “A fast quantum mechanical algorithm for database search,” in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC ’96. New York, NY, USA: Association for Computing Machinery, 1996, p. 212–219. [Online]. Available: https://doi.org/10.1145/237814.237866

[20] D. Deutsch and R. Jozsa, “Rapid solution of problems by quantum computation,” *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, vol. 439, no. 1907, pp. 553–558, 1992.

[21] B. Bichsel, M. Baader, T. Gehr, and M. Vechev, “Silq: A high-level quantum language with safe uncomputation and intuitive semantics,” in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020, pp. 286–300.

[22] P. Selinger, “Towards a quantum programming language,” *Mathematical Structures in Computer Science*, vol. 14, no. 4, pp. 527–586, 2004.

[23] T. Kubota, Y. Kakutani, G. Kato, and Y. Kawano, “A formal approach to unconditional security proofs for quantum key distribution,” in *International Conference on Unconventional Computation*. Springer, 2011, pp. 125–137.

[24] E. D’hondt and P. Panangaden, “Quantum weakest preconditions,” *Mathematical Structures in Computer Science*, vol. 16, no. 3, pp. 429–451, 2006.

[25] C. Morgan, A. McIver, and K. Seidel, “Probabilistic predicate transformers,” *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 18, no. 3, pp. 325–353, 1996.

[26] T. C. development team, *The Coq proof assistant reference manual*, LogiCal Project, 2004, version 8.0. [Online]. Available: http://coq.inria.fr

[27] B. C. Pierce, A. A. de Amorim, C. Casinghino, M. Gaboardi, M. Greenberg, C. Hriţcu, V. Sjöberg, and B. Yorgey, *Logical Foundations*, ser. Software Foundations, B. C. Pierce, Ed. Electronic textbook, 2021, vol. 1, version 6.1, http://softwarefoundations.cis.upenn.edu.

[28] J. Paykin, R. Rand, and S. Zdancewic, “Qwire: a core language for quantum circuits,” *ACM SIGPLAN Notices*, vol. 52, no. 1, pp. 846–858, 2017.

[29] T. Liu, Y. Li, S. Wang, M. Ying, and N. Zhan, “A theorem prover for quantum hoare logic and its applications,” *arXiv preprint arXiv:1601.03835*, 2016.

A

Some Proofs and Calculations

A.1 Theorems

In this appendix, we list some theorems that we used with an informal proof, calculations and some examples.

Theorem 2. *For any quantum classical state δ and a quantum classical assertion θ, x is a classical variable and e is an expression*

$$exp[\delta[x/e] \models \theta] = exp[\delta \models \theta[x/e]]$$

Proof.

$$\begin{aligned} exp[\delta[x/e] \models (\theta, \psi)] &= \sum_{(\sigma,\rho)\in\delta\wedge\sigma\models\theta[x/e]} trace(\rho.(\psi \otimes I^{qVars(\delta)\setminus qVars(\theta))}) \\ &= \sum_{(\sigma,\rho)\in\delta\wedge\sigma[x/e]\models\theta} trace(\rho.(\psi \otimes I^{qVars(\delta)\setminus qVars(\theta))}) \\ &= exp[\delta \models \theta[x/e]] \end{aligned}$$

□

Theorem 3. *if A, B were two square matrices with dimensions $n \otimes n$, then:*

$$trace(A.B) = trace(B.A)$$

Proof.

$$\begin{aligned} trace(A.B) &= \sum_{k=1}^{n}\sum_{i=1}^{n} a_{ki}b_{ik} \\ &= \sum_{i=1}^{n}\sum_{k=1}^{n} b_{ki}a_{ik} \\ &= trace(B.A) \end{aligned}$$

□

In the following, we prove the theorem in 1

Theorem 4.

$$\forall\, n \in \mathbf{N}_{dim} :\ M_0(n).M_0(n)^\dagger + M_1(n).M_1(n)^\dagger = I$$

Proof. first we start by proving that $\forall\, n \leq dim : M_0(n).M_0(n)^\dagger = M_0(n)$:

$$\begin{aligned} M_0(n).M_0(n)^\dagger &= ((\otimes_{i=0}^{n-1} I) \otimes |0\rangle\langle 0| \otimes (\otimes_{i=n+1}^{dim} I))((\otimes_{i=0}^{n-1} I) \otimes |0\rangle\langle 0| \otimes (\otimes_{i=n+1}^{dim} I))^\dagger \\ &= ((\otimes_{i=0}^{n-1} I)(\otimes_{i=0}^{n-1} I)^\dagger) \otimes ((|0\rangle\langle 0|)(|0\rangle\langle 0|)^\dagger) \otimes ((\otimes_{i=0}^{n-1} I)(\otimes_{i=0}^{n-1} I)^\dagger) \\ &= (\otimes_{i=0}^{n-1} I) \otimes |0\rangle\langle 0| \otimes (\otimes_{i=n+1}^{dim} I) \\ &= M_0(n) \end{aligned}$$

and similarly we prove that $\forall\, n \leq dim : M_1(n).M_1(n)^\dagger = M_1(n)$.

$$\begin{aligned}
M_0(n).M_0(n)^\dagger + M_1(n).M_1(n)^\dagger &= M_0(n) + M_1(n) \\
&= ((\otimes_{i=0}^{n-1} I) \otimes |0\rangle \langle 0| \otimes (\otimes_{i=n+1}^{dim} I)) + ((\otimes_{i=0}^{n-1} I) \otimes |1\rangle \langle 1| \otimes (\otimes_{i=n+1}^{dim} I)) \\
&= (\otimes_{i=0}^{n-1} I) \otimes (|0\rangle \langle 0| + |1\rangle \langle 1|) \otimes (\otimes_{i=n+1}^{dim} I) \\
&= (\otimes_{i=0}^{n-1} I) \otimes I \otimes (\otimes_{i=n+1}^{dim} I) \\
&= \otimes_{i=0}^{dim} I
\end{aligned}$$

□

A.2 Calculations

In 4.7.1, we did some calculations to obtain $P[q_2\ measures\ to\ 1]$ and $P[q_2\ measures\ to\ 1]$ for each one of the possibilities in the final state of 5. Here we show the calculations with details:

$$\begin{aligned}
\rho_0 &= (|0\rangle \otimes |0\rangle \otimes (\alpha\,|0\rangle + \beta\,|1\rangle))((\overline{\alpha}\,\langle 0| + \overline{\beta}\,\langle 1|) \otimes \langle 0| \otimes \langle 0|) \\
&= \begin{pmatrix}
|\alpha|^2 & 0 & 0 & 0 & \alpha\overline{\beta} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\beta\overline{\alpha} & 0 & 0 & 0 & |\beta|^2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\end{aligned}$$

$$\begin{aligned}
\rho_1 &= (I \otimes H \otimes I)\rho_0(I \otimes H \otimes I)^\dagger \\
&= 0.5 \begin{pmatrix}
2|\alpha|^2 & 2\alpha\overline{\beta} & 0 & 0 & 2|\alpha|^2 & 2\alpha\overline{\beta} & 0 & 0 \\
2\beta\overline{\alpha} & 2|\beta|^2 & 0 & 0 & 2\beta\overline{\alpha} & 2|\beta|^2 & 0 & 0 \\
2|\alpha|^2 & 2\alpha\overline{\beta} & 0 & 0 & 2|\alpha|^2 & 2\alpha\overline{\beta} & 0 & 0 \\
2\beta\overline{\alpha} & 2|\beta|^2 & 0 & 0 & 2\beta\overline{\alpha} & 2|\beta|^2 & 0 & 0 \\
2|\alpha|^2 & 2\alpha\overline{\beta} & 0 & 0 & 2|\alpha|^2 & 2\alpha\overline{\beta} & 0 & 0 \\
2\beta\overline{\alpha} & 2|\beta|^2 & 0 & 0 & 2\beta\overline{\alpha} & 2|\beta|^2 & 0 & 0 \\
2|\alpha|^2 & 2\alpha\overline{\beta} & 0 & 0 & 2|\alpha|^2 & 2\alpha\overline{\beta} & 0 & 0 \\
2\beta\overline{\alpha} & 2|\beta|^2 & 0 & 0 & 2\beta\overline{\alpha} & 2|\beta|^2 & 0 & 0
\end{pmatrix}
\end{aligned}$$

$$\rho_2 = (I \otimes CNOT)\rho_1(I \otimes CNOT)^\dagger$$
$$= 0.5 \begin{pmatrix} 2|\alpha|^2 & 2\alpha\overline{\beta} & 0 & 0 & 2|\alpha|^2 & 2\alpha\overline{\beta} & 0 & 0 \\ 2\beta\overline{\alpha} & 2|\beta|^2 & 0 & 0 & 2\beta\overline{\alpha} & 2|\beta|^2 & 0 & 0 \\ 2\beta\overline{\alpha} & 2|\beta|^2 & 0 & 0 & 2\beta\overline{\alpha} & 2|\beta|^2 & 0 & 0 \\ 2|\alpha|^2 & 2\alpha\overline{\beta} & 0 & 0 & 2|\alpha|^2 & 2\alpha\overline{\beta} & 0 & 0 \\ 2|\alpha|^2 & 2\alpha\overline{\beta} & 0 & 0 & 2|\alpha|^2 & 2\alpha\overline{\beta} & 0 & 0 \\ 2\beta\overline{\alpha} & 2|\beta|^2 & 0 & 0 & 2\beta\overline{\alpha} & 2|\beta|^2 & 0 & 0 \\ 2\beta\overline{\alpha} & 2|\beta|^2 & 0 & 0 & 2\beta\overline{\alpha} & 2|\beta|^2 & 0 & 0 \\ 2|\alpha|^2 & 2\alpha\overline{\beta} & 0 & 0 & 2|\alpha|^2 & 2\alpha\overline{\beta} & 0 & 0 \end{pmatrix}$$

$$\rho_3 = (CNOT \otimes I)\rho_2(CNOT \otimes I)^\dagger$$
$$= 0.5 \begin{pmatrix} 2|\alpha|^2 & 2\alpha\overline{\beta} & 0 & 0 & 0 & 0 & 2|\alpha|^2 & 2\alpha\overline{\beta} \\ 2\beta\overline{\alpha} & 2|\beta|^2 & 0 & 0 & 0 & 0 & 2\beta\overline{\alpha} & 2|\beta|^2 \\ 2\beta\overline{\alpha} & 2|\beta|^2 & 0 & 0 & 0 & 0 & 2\beta\overline{\alpha} & 2|\beta|^2 \\ 2|\alpha|^2 & 2\alpha\overline{\beta} & 0 & 0 & 0 & 0 & 2|\alpha|^2 & 2\alpha\overline{\beta} \\ 2|\alpha|^2 & 2\alpha\overline{\beta} & 0 & 0 & 0 & 0 & 2|\alpha|^2 & 2\alpha\overline{\beta} \\ 2\beta\overline{\alpha} & 2|\beta|^2 & 0 & 0 & 0 & 0 & 2\beta\overline{\alpha} & 2|\beta|^2 \\ 2\beta\overline{\alpha} & 2|\beta|^2 & 0 & 0 & 0 & 0 & 2\beta\overline{\alpha} & 2|\beta|^2 \\ 2|\alpha|^2 & 2\alpha\overline{\beta} & 0 & 0 & 0 & 0 & 2|\alpha|^2 & 2\alpha\overline{\beta} \end{pmatrix}$$

$$\rho_4 = (H \otimes I \otimes I)\rho_3(H \otimes I \otimes I)^\dagger$$
$$= 0.25 \begin{pmatrix} 4|\alpha|^2 & 4\alpha\overline{\beta} & 4|\alpha|^2 & 4\alpha\overline{\beta} & 4|\alpha|^2 & 4\alpha\overline{\beta} & 4|\alpha|^2 & 4\alpha\overline{\beta} \\ 4\beta\overline{\alpha} & 4|\beta|^2 & 4\beta\overline{\alpha} & 4|\beta|^2 & 4\beta\overline{\alpha} & 4|\beta|^2 & 4\beta\overline{\alpha} & 4|\beta|^2 \\ 4\beta\overline{\alpha} & 4|\beta|^2 & 4\beta\overline{\alpha} & 4|\beta|^2 & 4\beta\overline{\alpha} & 4|\beta|^2 & 4\beta\overline{\alpha} & 4|\beta|^2 \\ 4|\alpha|^2 & 4\alpha\overline{\beta} & 4|\alpha|^2 & 4\alpha\overline{\beta} & 4|\alpha|^2 & 4\alpha\overline{\beta} & 4|\alpha|^2 & 4\alpha\overline{\beta} \\ 4|\alpha|^2 & 4\alpha\overline{\beta} & 4|\alpha|^2 & 4\alpha\overline{\beta} & 4|\alpha|^2 & 4\alpha\overline{\beta} & 4|\alpha|^2 & 4\alpha\overline{\beta} \\ 4\beta\overline{\alpha} & 4|\beta|^2 & 2\beta\overline{\alpha} & 4|\beta|^2 & 4\beta\overline{\alpha} & 4|\beta|^2 & 4\beta\overline{\alpha} & 4|\beta|^2 \\ 4\beta\overline{\alpha} & 4|\beta|^2 & 2\beta\overline{\alpha} & 4|\beta|^2 & 4\beta\overline{\alpha} & 4|\beta|^2 & 4\beta\overline{\alpha} & 4|\beta|^2 \\ 4|\alpha|^2 & 4\alpha\overline{\beta} & 2|\alpha|^2 & 4\alpha\overline{\beta} & 4|\alpha|^2 & 4\alpha\overline{\beta} & 4|\alpha|^2 & 4\alpha\overline{\beta} \end{pmatrix}$$

$$\rho_{4_0} = (I \otimes I \otimes |0\rangle\langle 0|)\rho_4(I \otimes I \otimes |0\rangle\langle 0|)^\dagger$$
$$= 0.25 \begin{pmatrix} 4|\alpha|^2 & 0 & 4|\alpha|^2 & 0 & 4|\alpha|^2 & 0 & 4|\alpha|^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4\beta\overline{\alpha} & 0 & 4\beta\overline{\alpha} & 0 & 4\beta\overline{\alpha} & 0 & 4\beta\overline{\alpha} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4|\alpha|^2 & 0 & 4|\alpha|^2 & 0 & 4|\alpha|^2 & 0 & 4|\alpha|^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4\beta\overline{\alpha} & 0 & 2\beta\overline{\alpha} & 0 & 4\beta\overline{\alpha} & 0 & 4\beta\overline{\alpha} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\rho_{4_1} = (I \otimes I \otimes |1\rangle\langle 1|)\rho_4(I \otimes I \otimes |1\rangle\langle 1|)^\dagger$$
$$= 0.25 \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4|\beta|^2 & 0 & 4|\beta|^2 & 0 & 4|\beta|^2 & 0 & 4|\beta|^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4\alpha\overline{\beta} & 0 & 4\alpha\overline{\beta} & 0 & 4\alpha\overline{\beta} & 0 & 4\alpha\overline{\beta} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4|\beta|^2 & 0 & 4|\beta|^2 & 0 & 4|\beta|^2 & 0 & 4|\beta|^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4\alpha\overline{\beta} & 0 & 4\alpha\overline{\beta} & 0 & 4\alpha\overline{\beta} & 0 & 4\alpha\overline{\beta} \end{pmatrix}$$

$$\rho_{4_{00}} = (I \otimes |0\rangle\langle 0| \otimes I)\rho_{4_0}(I \otimes |0\rangle\langle 0| \otimes I)^\dagger$$
$$= 0.25 \begin{pmatrix} 4|\alpha|^2 & 0 & 4|\alpha|^2 & 0 & 4|\alpha|^2 & 0 & 4|\alpha|^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4\beta\overline{\alpha} & 0 & 4\beta\overline{\alpha} & 0 & 4\beta\overline{\alpha} & 0 & 4\beta\overline{\alpha} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4|\alpha|^2 & 0 & 4|\alpha|^2 & 0 & 4|\alpha|^2 & 0 & 4|\alpha|^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4\beta\overline{\alpha} & 0 & 2\beta\overline{\alpha} & 0 & 4\beta\overline{\alpha} & 0 & 4\beta\overline{\alpha} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\rho_{4_{01}} = (I \otimes |0\rangle\langle 0| \otimes I)\rho_{4_1}(I \otimes |0\rangle\langle 0| \otimes I)^\dagger$$
$$= 0.25 \begin{pmatrix} 4|\alpha|^2 & 0 & 4|\alpha|^2 & 0 & 4|\alpha|^2 & 0 & 4|\alpha|^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4\beta\overline{\alpha} & 0 & 4\beta\overline{\alpha} & 0 & 4\beta\overline{\alpha} & 0 & 4\beta\overline{\alpha} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4|\alpha|^2 & 0 & 4|\alpha|^2 & 0 & 4|\alpha|^2 & 0 & 4|\alpha|^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4\beta\overline{\alpha} & 0 & 2\beta\overline{\alpha} & 0 & 4\beta\overline{\alpha} & 0 & 4\beta\overline{\alpha} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\rho_{4_{10}} = (I \otimes |1\rangle\langle 1| \otimes I)\rho_{4_0}(I \otimes |1\rangle\langle 1| \otimes I)^\dagger$$
$$= 0.25 \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4|\beta|^2 & 0 & 4|\beta|^2 & 0 & 4|\beta|^2 & 0 & 4|\beta|^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4\alpha\overline{\beta} & 0 & 4\alpha\overline{\beta} & 0 & 4\alpha\overline{\beta} & 0 & 4\alpha\overline{\beta} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4|\beta|^2 & 0 & 4|\beta|^2 & 0 & 4|\beta|^2 & 0 & 4|\beta|^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4\alpha\overline{\beta} & 0 & 4\alpha\overline{\beta} & 0 & 4\alpha\overline{\beta} & 0 & 4\alpha\overline{\beta} \end{pmatrix}$$

$$\rho_{4_{11}} = (I \otimes |1\rangle\langle 1| \otimes I)\rho_{4_1}(I \otimes |1\rangle\langle 1| \otimes I)^\dagger$$
$$= 0.25 \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4|\beta|^2 & 0 & 4|\beta|^2 & 0 & 4|\beta|^2 & 0 & 4|\beta|^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4\alpha\overline{\beta} & 0 & 4\alpha\overline{\beta} & 0 & 4\alpha\overline{\beta} & 0 & 4\alpha\overline{\beta} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4|\beta|^2 & 0 & 4|\beta|^2 & 0 & 4|\beta|^2 & 0 & 4|\beta|^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4\alpha\overline{\beta} & 0 & 4\alpha\overline{\beta} & 0 & 4\alpha\overline{\beta} & 0 & 4\alpha\overline{\beta} \end{pmatrix}$$

$$\rho_{00} = \rho_{4_{00}}$$
$$= 0.25 \begin{pmatrix} 4|\alpha|^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4|\alpha|^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4|\beta|^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -4|\beta|^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\rho_{01} = (X \otimes I \otimes I)\rho_{4_{01}}(X \otimes I \otimes I)^\dagger$$
$$= 0.25 \begin{pmatrix} 4|\alpha|^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4|\alpha|^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4|\beta|^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -4|\beta|^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\rho_{10} = (Z \otimes I \otimes I)\rho_{4_{10}}(Z \otimes I \otimes I)^\dagger$$
$$= 0.25 \begin{pmatrix} 4|\alpha|^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4|\alpha|^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4|\beta|^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -4|\beta|^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\rho_{11} = (X \otimes I \otimes I)(Z \otimes I \otimes I)\rho_{4_{11}}(Z \otimes I \otimes I)^\dagger(X \otimes I \otimes I)^\dagger$$
$$= 0.25 \begin{pmatrix} 4|\alpha|^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4|\alpha|^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4|\beta|^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -4|\beta|^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

the first state:

$$P[q_2 \; measures \; to \; 0] = \langle 000| \, \rho_{00} \, |000\rangle$$
$$= 0.25 \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 4|\alpha|^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4|\beta|^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4|\alpha|^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -4|\beta|^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$
$$= |\alpha|^2$$

$$P[q_2 \; measures \; to \; 1] = \langle 100| \, \rho_{00} \, |001\rangle$$
$$= 0.25 \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4|\beta|^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4|\alpha|^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4|\beta|^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4|\alpha|^2 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$
$$= |\beta|^2$$

the second state:

$$P[q_2 \; measures \; to \; 0] = \langle 010| \, \rho_{01} \, |010\rangle$$
$$= \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} 0.25 \begin{pmatrix} 4|\alpha|^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4|\alpha|^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4|\beta|^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -4|\beta|^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$
$$= |\alpha|^2$$

$$P[q_2\ measures\ to\ 1] = \langle 110|\ \rho_{01}\ |011\rangle$$

$$= \begin{pmatrix}0 & 0 & 0 & 1 & 0 & 0 & 0 & 0\end{pmatrix} 0.25 \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4|\alpha|^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4|\beta|^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4|\alpha|^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4|\beta|^2 \end{pmatrix} \begin{pmatrix}0\\0\\0\\1\\0\\0\\0\\0\end{pmatrix}$$

$$= |\beta|^2$$

the third state:

$$P[q_2\ measures\ to\ 0] = \langle 001|\ \rho_{10}\ |100\rangle$$

$$= \begin{pmatrix}0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\end{pmatrix} 0.25 \begin{pmatrix} 4|\alpha|^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4|\beta|^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4|\alpha|^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -4|\beta|^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix}0\\0\\0\\0\\1\\0\\0\\0\end{pmatrix}$$

$$= |\alpha|^2$$

$$P[q_2\ measures\ to\ 1] = \langle 101|\ \rho_{10}\ |101\rangle$$

$$= \begin{pmatrix}0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\end{pmatrix} 0.25 \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4|\alpha|^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4|\beta|^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4|\beta|^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4|\alpha|^2 \end{pmatrix} \begin{pmatrix}0\\0\\0\\0\\0\\1\\0\\0\end{pmatrix}$$

$$= |\beta|^2$$

the forth state:

$$P[q_2\ measures\ to\ 0] = \langle 011|\, \rho_{11}\, |110\rangle$$

$$= 0.25 \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 4|\beta|^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4|\alpha|^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4|\beta|^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 4|\alpha|^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$= |\alpha|^2$$

$$P[q_2\ measures\ to\ 1] = \langle 111|\, \rho_{11}\, |111\rangle$$

$$= 0.25 \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4|\alpha|^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4|\beta|^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4|\alpha|^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4|\beta|^2 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

$$= |\beta|^2$$

The following calculations refers to the expressions in 4.3:

$$\rho_{00} = (|0\rangle \langle 0| \otimes I)(X \otimes I)(I \otimes |0\rangle \langle 0|)(I \otimes H)\, |00\rangle \langle 00|\, (I \otimes H)^\dagger (I \otimes |0\rangle \langle 0|)^\dagger (X \otimes I)^\dagger (|0\rangle \langle 0| \otimes I)^\dagger$$

$$= 0.5 \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\rho_{01} = (|1\rangle \langle 1| \otimes I)(X \otimes I)(I \otimes |0\rangle \langle 0|)(I \otimes H)\, |00\rangle \langle 00|\, (I \otimes H)^\dagger (I \otimes |0\rangle \langle 0|)^\dagger (X \otimes I)^\dagger (|1\rangle \langle 1| \otimes I)^\dagger$$

$$= 0.5 \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\rho_{10} = (|0\rangle\langle 0| \otimes I)(I \otimes |1\rangle\langle 1|)(I \otimes H)\,|00\rangle\langle 00|\,(I \otimes H)^\dagger (I \otimes |1\rangle\langle 1|)^\dagger (|0\rangle\langle 0| \otimes I)^\dagger$$
$$= 0.5 \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\rho_{11} = (|1\rangle\langle 1| \otimes I)(I \otimes |1\rangle\langle 1|)(I \otimes H)\,|00\rangle\langle 00|\,(I \otimes H)^\dagger (I \otimes |1\rangle\langle 1|)^\dagger (|1\rangle\langle 1| \otimes I)^\dagger$$
$$= 0.5 \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$