

# Laws of Quantum Programming

MINGSHENG YING\*, Centre for Quantum Software and Information, University of Technology Sydney, Australia

LI ZHOU\*, Institute of Software, Chinese Academy of Sciences, China

GILLES BARTHE, Max Planck Institute for Security and Privacy, Germany and IMDEA Software Institute, Spain

In this paper, we investigate the fundamental laws of quantum programming. We extend a comprehensive set of Hoare et al.'s basic laws of classical programming to the quantum setting. These laws characterise the algebraic properties of quantum programs, such as the distributivity of sequential composition over (quantum) if-statements and the unfolding of nested (quantum) if-statements. At the same time, we clarify some subtle differences between certain laws of classical programming and their quantum counterparts. Additionally, we derive a fixpoint characterisation of quantum while-loops and a loop-based realisation of tail recursion in quantum programming. Furthermore, we establish two normal form theorems: one for quantum circuits and one for finite quantum programs. The theory in which these laws are established is formalised in the Coq proof assistant, and all of these laws are mechanically verified. As an application case of our laws, we present a formal derivation of the principle of deferred measurements in dynamic quantum circuits.

We expect that these laws can be utilised in correctness-preserving transformation, compilation, and automatic code optimisation in quantum programming. In particular, because these laws are formally verified in Coq, they can be confidently applied in quantum program development.

CCS Concepts: • **Software and its engineering** → **Functionality; Automated static analysis; Theory of computation** → **Algebraic semantics**.

Additional Key Words and Phrases: Quantum programming, algebraic laws, normal forms, formal verification in Coq.

## ACM Reference Format:

Mingsheng Ying, Li Zhou, and Gilles Barthe. 2026. Laws of Quantum Programming. *ACM Trans. Softw. Eng. Methodol.* 1, 1 (April 2026), 51 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

## 1 Introduction

It is widely accepted that scientists and mathematicians uncover the fundamental laws of nature and mathematical structures, respectively, while engineers leverage these discoveries to address practical problems. This fundamental concept spurred Hoare et al. [32, 33, 56] to pose a crucial question:

- *What are the laws governing computer programming that establish the formal foundation for the field of software engineering?*

Their argument, rooted in the perspective that computer programs are essentially mathematical expressions, led them to establish a series of algebraic laws for programming, ranging from sequential to concurrent processes. The foundational laws articulated in [32, 33, 56] serve as a cornerstone in the establishment of the science of computer programming. Simultaneously, they prove invaluable

---

\*The first two authors contribute equally.

---

Authors' Contact Information: [Mingsheng Ying](mailto:Mingsheng.Ying@uts.edu.au), Centre for Quantum Software and Information, University of Technology Sydney, Ultimo, Australia, [Mingsheng.Ying@uts.edu.au](mailto:Mingsheng.Ying@uts.edu.au); [Li Zhou](mailto:Li.Zhou@ios.ac.cn), Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China, [zhouli@ios.ac.cn](mailto:zhouli@ios.ac.cn), [zhou31416@gmail.com](mailto:zhou31416@gmail.com); [Gilles Barthe](mailto:Gilles.Barthe@mpi-sp.org), Max Planck Institute for Security and Privacy, Bochum, Germany and IMDEA Software Institute, Madrid, Spain, [gilles.barthe@mpi-sp.org](mailto:gilles.barthe@mpi-sp.org).

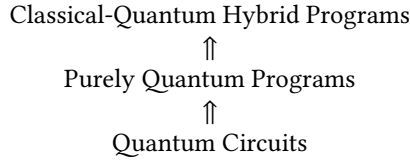


Fig. 1. Three-Layer Framework for Quantum Programming.

in the realm of software engineering, encompassing tasks such as compilation, optimisation, and program transformation. For instance, during program development, a programmer may create a program aligning with end-user requirements. Utilising these laws, an optimising compiler can then transform it into an efficient program that effectively exploits the underlying architecture of the target machine (refer to [33] for a detailed example). This strategic approach has the advantage of ensuring correctness by construction during compilation. Each transformation within the process is mathematically proven based on these laws, providing a robust framework for the development and optimisation of computer programs.

The past decade has witnessed remarkable advancements in quantum computer hardware, leading to a surge in research on quantum programming languages and platforms [1, 19, 26–29, 34, 45, 49, 61, 64, 65]. The research on quantum programming also spans from theoretical aspects, such as formal semantics, program logic, static analysis, and verification, to more practical considerations like compilation, optimisation, testing, and debugging. Despite this progress, the knowledge base of quantum programming as a scientific discipline remains fragmented and disjointed. Consequently, two fundamental questions naturally arise:

- *What are the foundational laws that underpin the methodologies and techniques of quantum programming?*
- *Are there discernible differences between these laws and those governing classical programming, as established in works like [32, 33, 56]?*

Addressing these questions is crucial for consolidating the understanding of quantum programming as a coherent and well-defined field, bridging the gap between theoretical insights and practical applications.

The objective of this paper is to delve into fundamental inquiries and identify key laws that can establish an indispensable scientific foundation for quantum programming. To accomplish this, we present a comprehensive three-layer framework for quantum programming in Figure 1, with each layer intricately embedded into the one above:

- *Quantum Circuits Description Layer:* At the bottom is a description language for quantum circuits. Assuming a set of basic gates, any quantum circuit can be defined using sequential composition and a quantum if-statement [2, 9, 73, 78, 83].
- *Purely Quantum Programs Layer:* The middle layer introduces a language for writing purely quantum programs (without classical variables). Building upon quantum circuits, an if-statement with the outcome of a quantum measurement as its condition is introduced. This layer also defines the loop and recursion of quantum programs (with classical control flow). Nondeterministic choice is added for the specification and refinement of quantum programs.
- *Classical-Quantum Hybrid Programming Layer:* At the top is the classical-quantum hybrid programming layer, where purely quantum programs are embedded into a classical programming language. This integration enables the programming of classical and quantum computation together, a necessity for most practical quantum algorithms.

While the components of this framework are not novel and have been extensively studied in previous literature, we organise them into a unified framework that clearly delineates the application of each law of quantum programming.

Within this three-layer framework, we establish a comprehensive set of basic laws of quantum programming. This includes two subsets of laws:

- The first subset is proven for quantum circuits at the bottom layer of Figure 1. Among them, several laws about the distributivity of quantum **if**-statements are particularly useful.
- The second subset is established for purely quantum programs at the middle layer of Figure 1. A congruence property is introduced, facilitating the lifting of laws for quantum circuits to quantum programs. Then a diverse range of laws governing **if**-statements, loops, recursion, and nondeterministic choice are derived.

These quantum programming laws can be further integrated with existing laws for classical (and probabilistic) programs [32, 33, 35, 56]. This integration enhances the framework's utility for reasoning about classical-quantum hybrid programs at the top layer of Figure 1.

Noteworthy technical contributions of this paper encompass:

- **Normal forms:** We prove that every quantum circuit is equivalent to a normal form featuring a sequential composition of flat (but large) quantum **if**-statements. A similar normal form (with an **if**-statement) is presented for finite quantum programs. The significance of the normal forms lies in their applications, including providing a compilation strategy of quantum programs as a generalisation of that given in [33] and offering a method for checking the equivalence of two quantum programs by reducing them to a normal form.
- **Tail Recursion:** It is well-known in classical programming that tail recursion can be realised using loops. We prove that this result holds true for recursive quantum programs with classical control flow. A practical implication of this finding is that quantum programs with tail recursion can be easily implemented and optimised during compilation.
- **Verification in Coq:** We formalise the theory in which our laws are established in the Coq proof assistant, by leveraging CoqQ [85], a general-purpose project for quantum program verification. The formalisation from the first principle ensures the soundness of the laws presented in this paper. We expect that they will be confidently and frequently used in quantum development. As observed in [5, 8], algebraic laws like these are also very helpful in the security verification of quantum cryptography.

## 1.1 An Illustrative Example

The laws proposed in this paper are anticipated to play a role in quantum programming, akin to that played by the laws presented in [32, 33, 56] in classical programming. To explore this prospect, this subsection demonstrates a simple example applying our proposed laws.

Quantum error correction (QEC) techniques are the key to large-scale and practical quantum computing. Usually, QEC codes can be conveniently written as quantum programs. Let us consider the simplest QEC code—the *three qubit bit-flip code* (see [48], Section 10.1.1). The following example shows how the correctness of this code can be derived directly using our laws.

**EXAMPLE 1.1 (QEC CODE).** *Suppose that we want to send a qubit  $q$  in state  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$  through a channel that flips from  $|0\rangle$  to  $|1\rangle$  (and vice versa) with a certain probability, called a bit flip channel. The basic idea of QEC is:*

- (1) (**Encoding**) *Encode the state  $|\psi\rangle$  in three qubits  $q, q_1$  and  $q_2$  as  $\alpha|000\rangle + \beta|111\rangle$ . This can be done by initialising  $q_1, q_2$  both in basis state  $|0\rangle$ , and then performing the controlled-NOT gate twice, both with  $q$  as the control qubit, but the first with  $q_1$  and the second with  $q_2$  as the target qubit.*

- (2) (**Sending via noisy channel**) Each of the three qubits is sent through an independent bit flip channel. Assume that a bit flip happens only on one or none of the qubits.
- (3) (**Decoding**) Perform the controlled-NOT gate twice, both with  $q$  as the control qubit, but the first with  $q_2$  and the second with  $q_1$  as the target qubit.
- (4) (**Error correction**) Measure qubit  $q_2$  in the computational basis: if the outcome is 0, then the procedure terminates. Otherwise, we measure qubit  $q_1$  also in the computational basis: if the outcome is 0, then terminate; otherwise, perform the NOT gate on qubit  $q$ . Additional NOT gates are applied to  $q_1$  and  $q_2$  conditioned on the measurement outcomes to recover them to their initial state.

As suggested in [24], the above entire process can be appropriately modelled as a nondeterministic quantum program:

$$\begin{array}{l}
 \text{QEC} \triangleq \begin{array}{l}
 1 \quad q_1 := |0\rangle; q_2 := |0\rangle; \text{CNOT}[q, q_1]; \text{CNOT}[q, q_2]; \\
 2 \quad (\text{skip} \sqcup X[q] \sqcup X[q_1] \sqcup X[q_2]); \\
 3 \quad \text{CNOT}[q, q_2]; \text{CNOT}[q, q_1]; q_2 \triangleright (q_1 \triangleright (X[q]; X[q_1]); X[q_2]).
 \end{array}
 \end{array}$$

Line 1 represents the encoding part where CNOT denotes the controlled-NOT gate. Line 2 models the noise where  $X$  denotes the NOT gate and  $\sqcup$  stands for nondeterministic choice; thus, a bit flip occurs in at most one of the three channels. Line 3 corresponds to the decoding and error correction part, where  $r \triangleright P$  denotes the conditional statement that if the outcome of the computational measurement on a qubit  $r$  is 1, then execute  $P$ .

It was verified in [24] using quantum Hoare logic that the program  $\text{QEC}$  is correct: if  $q$  in state  $|\psi\rangle$  is inputted to it, then  $q$  will be in the same state  $|\psi\rangle$  at the output of  $\text{QEC}$  (hence, error was corrected). Here, we prove the correctness of  $\text{QEC}$  by *program transformation* using only the laws established in this paper.

Let  $C_0 \leftarrow q \rightarrow C_1$  be a quantum if-statement (see Eq. (12) for its definition). First, we have:

$$\begin{aligned}
 & \text{CNOT}[q, q_2]; X[q]; \text{CNOT}[q, q_2] \\
 &= \text{skip} \leftarrow q \rightarrow X[q_2]; X[q]; \text{skip} \leftarrow q \rightarrow X[q_2] \\
 &\equiv X[q]; X[q_2] \leftarrow q \rightarrow \text{skip}; \text{skip} \leftarrow q \rightarrow X[q_2] \quad (\text{By Prop. 4.3(1)}) \\
 &\equiv X[q]; (X[q_2]; \text{skip}) \leftarrow q \rightarrow (\text{skip}; X[q_2]) \quad (\text{By Prop. 4.2(7)}) \\
 &\equiv X[q]; X[q_2] \leftarrow q \rightarrow X[q_2] \quad (\text{By Prop. 4.2(1)}) \\
 &\equiv X[q]; X[q_2]. \quad (\text{By Prop. 4.1(3)})
 \end{aligned} \tag{1}$$

Then we obtain:

$$\begin{aligned}
 & \text{CNOT}[q, q_1]; \text{CNOT}[q, q_2]; X[q]; \text{CNOT}[q, q_2]; \text{CNOT}[q, q_1] \\
 &\equiv \text{CNOT}[q, q_1]; X[q]; X[q_2]; \text{CNOT}[q, q_1] \\
 &\equiv \text{CNOT}[q, q_1]; X[q]; \text{CNOT}[q, q_1]; X[q_2] \quad (\text{By Prop. 4.2(5)}) \\
 &\equiv X[q]; X[q_1]; X[q_2]. \quad (\text{By an argument similar to (1)})
 \end{aligned} \tag{2}$$

Likewise, we can prove:

$$\begin{cases}
 \text{CNOT}[q, q_1]; \text{CNOT}[q, q_2]; \text{skip}; \text{CNOT}[q, q_2]; \text{CNOT}[q, q_1] \equiv \text{skip}, \\
 \text{CNOT}[q, q_1]; \text{CNOT}[q, q_2]; X[q_1]; \text{CNOT}[q, q_2]; \text{CNOT}[q, q_1] \equiv X[q_1], \\
 \text{CNOT}[q, q_1]; \text{CNOT}[q, q_2]; X[q_2]; \text{CNOT}[q, q_2]; \text{CNOT}[q, q_1] \equiv X[q_2].
 \end{cases} \tag{3}$$

On the other hand, we have:

$$\begin{aligned}
& q_1 := |0\rangle; q_2 := |0\rangle; X[q]; X[q_1]; X[q_2]; q_2 \triangleright (q_1 \triangleright (X[q]; X[q_1]); X[q_2]) \\
& \equiv X[q]; (q_1 := |0\rangle; X[q_1]); (q_2 := |0\rangle; X[q_2]); \\
& \quad q_2 \triangleright (q_1 \triangleright (X[q]; X[q_1]); X[q_2]) \quad (\text{By Prop. 5.4(2)}) \\
& \equiv X[q]; q_1 := |1\rangle; q_2 := |1\rangle; q_2 \triangleright (q_1 \triangleright (X[q]; X[q_1]); X[q_2]) \quad (\text{By Prop. 5.1(2)}) \\
& \equiv X[q]; q_1 := |1\rangle; q_2 := |1\rangle; (q_1 \triangleright (X[q]; X[q_1]); X[q_2]) \quad (\text{By Prop. 5.1(4)}) \quad (4) \\
& \equiv X[q]; q_2 := |1\rangle; q_1 := |1\rangle; q_1 \triangleright (X[q]; X[q_1]); X[q_2] \quad (\text{By Prop. 5.4(2)}) \\
& \equiv X[q]; q_2 := |1\rangle; q_1 := |1\rangle; (X[q]; X[q_1]); X[q_2] \quad (\text{By Prop. 5.1(4)}) \\
& \equiv (q_1 := |1\rangle; X[q_1]); (q_2 := |1\rangle; X[q_2]); (X[q]; X[q]) \quad (\text{By Prop. 5.4(2)}) \\
& \equiv q_1 := |0\rangle; q_2 := |0\rangle. \quad (\text{By Props. 5.4(2), 4.2(1-3)})
\end{aligned}$$

Similarly, we obtain:

$$\left\{ \begin{aligned}
& q_1 := |0\rangle; q_2 := |0\rangle; \mathbf{skip}; q_2 \triangleright (q_1 \triangleright (X[q]; X[q_1]); X[q_2]) \equiv q_1 := |0\rangle; q_2 := |0\rangle, \\
& q_1 := |0\rangle; q_2 := |0\rangle; X[q_1]; q_2 \triangleright (q_1 \triangleright (X[q]; X[q_1]); X[q_2]) \equiv q_1 := |0\rangle; q_2 := |0\rangle, \\
& q_1 := |0\rangle; q_2 := |0\rangle; X[q_2]; q_2 \triangleright (q_1 \triangleright (X[q]; X[q_1]); X[q_2]) \equiv q_1 := |0\rangle; q_2 := |0\rangle.
\end{aligned} \right. \quad (5)$$

Finally, by combining Eqs. (2) to (5) and using Proposition 7.1(5), we assert:

$$\begin{aligned}
QEC & \equiv q_1 := |0\rangle; q_2 := |0\rangle; (\mathbf{skip} \sqcup (X[q]; X[q_1]; X[q_2]) \sqcup X[q_1] \sqcup X[q_2]); \\
& \quad q_2 \triangleright (q_1 \triangleright (X[q]; X[q_1]); X[q_2]) \\
& \equiv (q_1 := |0\rangle; q_2 := |0\rangle) \sqcup (q_1 := |0\rangle; q_2 := |0\rangle) \quad (6) \\
& \quad \sqcup (q_1 := |0\rangle; q_2 := |0\rangle) \sqcup (q_1 := |0\rangle; q_2 := |0\rangle) \\
& \equiv q_1 := |0\rangle; q_2 := |0\rangle.
\end{aligned}$$

It is clear from the last line of Eq. (6) that  $QEC$  does not change the state of  $q$ , and thus  $q$  is still in state  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$  at the output of  $QEC$ .

## 1.2 Organisation of the Paper

The paper is organised as follows. For convenience of comparing the laws of quantum programming with their classical counterparts, we briefly review some basic laws of classical programming in Section 2. The three-layer framework of quantum programming is described in Section 3. Our laws of quantum programming are developed from Sections 4 through 7. In Section 4, various algebraic laws for quantum circuits at the bottom layer of our quantum programming framework are presented. The normal form of quantum circuits is also shown there. The laws for purely quantum programs at the second layer of our programming framework are introduced in Section 5. In particular, the normal form of finite quantum programs is proved there. In Section 6, several laws for quantum loops and recursive quantum programs are established, including the fixpoint characterisation of quantum loops and the loop-based realisation of tail recursion in quantum programming. Some laws for deterministic quantum programs given in Sections 5 are generalised to nondeterministic quantum programs in Section 7. Several new laws about nondeterministic quantum programs are also given there. In Section 8, we present a series of laws for refinement of quantum programs. To show the effect of the laws developed in this paper, a more sophisticated example is given in Section 9 where the principle of deferred measurements is formally derived using our laws. The formalisation of all these laws in the Coq proof assistant is presented in Section 10. The related work is discussed in Section 11. A brief conclusion is drawn in Section 12, where we also point out some topics for future research.

$$\begin{aligned}
& \text{(If-1)} \ P \triangleleft \text{true} \triangleright Q \equiv P & \text{(If-2)} \ P \triangleleft \text{false} \triangleright Q \equiv Q & \text{(If-3)} \ P \triangleleft b \triangleright P \equiv P \\
& \text{(If-4)} \ P \triangleleft b \triangleright (Q \triangleleft b \triangleright R) \equiv (P \triangleleft b \triangleright Q) \triangleleft b \triangleright R \\
& \text{(If-5)} \ P \triangleleft b \triangleright Q \equiv Q \triangleleft \neg b \triangleright P \\
& \text{(If-6)} \ P \triangleleft (c \triangleleft b \triangleright d) \triangleright Q \equiv (P \triangleleft c \triangleright Q) \triangleleft b \triangleright (P \triangleleft d \triangleright Q) \\
& \text{(If-7)} \ P \triangleleft b \triangleright (Q \triangleleft b \triangleright R) \equiv P \triangleleft b \triangleright R \\
& \text{(If-8)} \ (P \triangleleft b \triangleright Q) \triangleleft c \triangleright R \equiv (P \triangleleft c \triangleright R) \triangleleft b \triangleright (Q \triangleleft c \triangleright R) \\
& \text{(Sq-1)} \ \text{skip}; P \equiv P; \text{skip} \equiv P & \text{(Sq-2)} \ \text{abort}; P \equiv P; \text{abort} \equiv \text{abort} \\
& \text{(Sq-3)} \ P; (Q; R) \equiv (P; Q); R & \text{(Sq-4)} \ (P \triangleleft b \triangleright Q); R \equiv (P; R) \triangleleft b \triangleright (Q; R) \\
& \text{(Nd-1)} \ P \sqcup Q \equiv Q \sqcup P & \text{(Nd-2)} \ P \sqcup (Q \sqcup R) \equiv (P \sqcup Q) \sqcup R \\
& \text{(Nd-3)} \ (P \sqcup Q) \triangleleft b \triangleright R \equiv (P \triangleleft b \triangleright R) \sqcup (Q \triangleleft b \triangleright R) \\
& \text{(Nd-4)} \ R \triangleleft b \triangleright (P \sqcup Q) \equiv (R \triangleleft b \triangleright P) \sqcup (R \triangleleft b \triangleright Q) \\
& \text{(Nd-5)} \ (P \triangleleft b \triangleright Q) \sqcup R \equiv (P \sqcup R) \triangleleft b \triangleright (Q \sqcup R) \\
& \text{(Nd-6)} \ (P \sqcup Q); R \equiv (P; R) \sqcup (Q; R) & \text{(Nd-7)} \ R; (P \sqcup Q) \equiv (R; P) \sqcup (R; Q) \\
& \text{(Re-1)} \ b * P \equiv \mu X. ((P; X) \triangleleft b \triangleright \text{skip}) \\
& \text{(Re-2)} \ \mu X. F(X) \equiv F(\mu X. F(X)) & \text{(Re-3)} \ Y = F(Y) \Rightarrow \mu X. F(X) \subseteq Y
\end{aligned}$$

Fig. 2. Laws of Classical Programming. In law (If-6),  $c \triangleleft b \triangleright d$  is a boolean expression that assumes value  $c$  if  $b$  is true and value  $d$  if  $b$  is false. In law (Re-3), the inclusion relation  $\subseteq$  represents the refinement of programs.

## 2 Laws of Classical Programming

In this section, we briefly recall the laws of classical programming established in [32, 33, 56]. On the one hand, these laws motivate our laws of quantum programming. On the other hand, a careful comparison between these laws and our laws of quantum programming presented in the subsequent sections will help us to understand better some basic (but subtle) differences between classical and quantum program constructs.

The language employed in [32] to formulate the laws is Dijkstra's nondeterministic sequential programming language [22], which has the **skip**, **abort** commands, assignment  $x := E$ , sequential composition  $P; Q$ , conditional  $P \triangleleft b \triangleright Q$  (standing for **if**  $b$  **then**  $P$  **else**  $Q$ ), nondeterministic choice  $P \sqcup Q$ , iteration  $b * P$  (standing for **while**  $b$  **do**  $P$ ), and recursion  $\mu X. F(X)$ .

The main laws established in [32, 33, 56] are presented in Figure 2. Most of these laws will be generalised in this paper to both the layers of quantum circuits and purely quantum programs (see Figure 1). The connections between these laws and our laws of quantum programming will be discussed whenever the latter are presented. At the third layer in Figure 1, the classical laws in Figure 2 and our quantum laws can be combined together to reason about classical-quantum hybrid programs.

## 3 A Framework of Quantum Programming

In this section, we define a three-layer framework of quantum programming in which our laws can be properly accommodated. As pointed out in the Introduction, all ingredients of this framework are not new and have been widely used by the quantum programming community.

We assume a set  $QV$  of quantum variables and a set  $\mathcal{U}$  of *unitary matrix constants*. The unitary matrix constants in  $\mathcal{U}$  will be instantiated in applications. For convenience, it is always assumed that the  $2 \times 2$  *unit matrix* is in  $\mathcal{U}$ . We fix the following notations:

- Each quantum variable  $q$  stands for a quantum system with a Hilbert space denoted by  $\mathcal{H}_q$  as its state space.
- A sequence  $\bar{q} = q_1, \dots, q_n$  of distinct quantum variables is called a quantum register. It denotes a composite quantum system consisting of subsystems  $q_1, \dots, q_n$ . Its state space, or type, denoted by  $type(\bar{q})$ , is given by the tensor product  $\bigotimes_{i=1}^n \mathcal{H}_{q_i}$ . We write  $\bar{q}_1, \bar{q}_2$  for the concatenation of two distinct registers  $\bar{q}_1$  and  $\bar{q}_2$ . For simplicity of the presentation, we often identify  $\bar{q}$  with the set  $\{q_1, \dots, q_n\}$  of quantum variables occurring in  $\bar{q}$ .
- Each unitary matrix constant  $U \in \mathcal{U}$  assumes a type of the form  $type(U) = \mathcal{H}_1 \otimes \dots \otimes \mathcal{H}_n$ . This means that the unitary transformation denoted by  $U$  can be performed on a composite quantum system consisting of  $n$  subsystems with state spaces  $\mathcal{H}_1, \dots, \mathcal{H}_n$ , respectively. Thus, if  $\bar{q} = q_1, \dots, q_n$  is a quantum register with  $\mathcal{H}_{q_i} = \mathcal{H}_i$  for  $i = 1, \dots, n$  (that is, the types of  $U$  matches that of the register  $\bar{q}$ ), then  $U[\bar{q}]$  can be thought of as a basic quantum gate with quantum wires  $q_1, \dots, q_n$ .

### 3.1 Quantum circuits

The first layer of our framework (see Figure 1) is a description language of quantum circuits. Formally, we have:

DEFINITION 3.1. *Quantum circuits*  $C \in \mathbf{QC}$  are defined by the syntax:

$$C ::= \mathbf{skip} \mid U[\bar{q}] \mid C_1; C_2 \mid \mathbf{qif} [\bar{q}] \left( \square_{i=1}^d |\psi_i\rangle \rightarrow C_i \right) \mathbf{fiq} \quad (7)$$

More precisely, quantum circuits  $C$  are inductively defined by the following clauses, where  $qv(C)$  denotes the quantum variables in  $C$ :

- (1) **Skip**:  $qv(\mathbf{skip}) = \emptyset$ ;
- (2) **Basic gates**: If  $U \in \mathcal{U}$  is a unitary matrix constant and  $\bar{q}$  is a quantum register such that their types match, then quantum gate  $U[\bar{q}]$  is a circuit, and  $qv(U[\bar{q}]) = \bar{q}$ ;
- (3) **Sequential composition**: If  $C_1$  and  $C_2$  are circuits, then  $C \equiv C_1; C_2$  is a circuit too, and  $qv(C) = qv(C_1) \cup qv(C_2)$ ;
- (4) **Quantum if-statement**: If  $\bar{q}$  is a quantum register,  $\{|\psi_i\rangle\}_{i=1}^d$  is an orthonormal basis of the Hilbert space  $type(\bar{q})$ , and  $C_i$  ( $i = 1, \dots, d$ ) are circuits with

$$\bar{q} \cap \left( \bigcup_{i=1}^d qv(C_i) \right) = \emptyset, \quad (8)$$

then

$$C \equiv \mathbf{qif} [\bar{q}] \left( \square_{i=1}^d |\psi_i\rangle \rightarrow C_i \right) \mathbf{fiq} \quad (9)$$

is a circuit, and  $qv(C) = \bar{q} \cup \left( \bigcup_{i=1}^d qv(C_i) \right)$ .  $\bar{q}$  is called the *guard register*, and  $\{|\psi_i\rangle\}$  is called the *guard basis* of the quantum if-statement (9).

Intuitively, each  $C \in \mathbf{QC}$  represents a circuit with quantum wires  $qv(C)$ . In other words, the type of  $C$  is  $type(C) = \bigotimes_{q \in qv(C)} \mathcal{H}_q$ , and as will be seen in Definition 3.3, the semantics of  $C$  is a unitary operator on  $type(C)$ . The circuit constructs introduced in the above definition are explained as follows.

- (i) The circuit  $C_1; C_2$  in Clause (3) stands for the sequential composition of circuits  $C_1$  and  $C_2$ . Indeed, if  $C_1$  and  $C_2$  do not share quantum variables; that is,  $qv(C_1) \cap qv(C_2) = \emptyset$ , we can also define their parallel composition  $C_1 \otimes C_2$ . But  $C_1; C_2$  and  $C_1 \otimes C_2$  are semantically equivalent whenever  $qv(C_1) \cap qv(C_2) = \emptyset$ . So, the parallel composition is not included in the above definition.
- (ii) The quantum **if**-statement defined in Eq. (9) has been extensively discussed in the previous literature [2, 9, 57, 73, 78, 83]. The condition (8) means that  $\bar{q}$  is a system external to all  $C_i$  ( $i = 1, \dots, d$ ). Semantically, a quantum **if**-statement (9) is a quantum multiplexor (i.e., a multi-way generalisation of conditional) [60]. A multiplexer can be understood as a switch that passes one of its data inputs through to the output, as a function of a set of select inputs. Here,  $\bar{q}$  is the select register, and if  $\bar{q}$  is in state  $|\psi_i\rangle$ , then a quantum datum  $|\varphi\rangle$  is inputted to the corresponding circuit  $C_i$  and an output  $|\varphi_i\rangle$  is obtained at the end of  $C_i$ . A basic difference between classical and quantum multiplexors is that the quantum select register  $\bar{q}$  can be in a superposition of  $|\psi_i\rangle$  ( $i = 1, \dots, d$ ), say  $\sum_{i=1}^d \alpha_i |\psi_i\rangle$ . In this case, the output is then  $\sum_{i=1}^d \alpha_i |\varphi_i\rangle$ , a superposition of the outputs of different circuits  $C_i$  ( $i = 1, \dots, d$ ). Therefore, the control flow of a quantum **if**-statement is inherently quantum.

For simplicity of presentation, let us introduce a syntactic sugar for quantum **if**-statement when some of the branches are **skip**:

$$\mathbf{qif} [\bar{q}] (\square_{i \in I} |\psi_i\rangle \rightarrow U_i) \mathbf{fiq} \triangleq \mathbf{qif} [\bar{q}] ((\square_{i \in I} |\psi_i\rangle \rightarrow U_i) \square (\square_{i \notin I} |\psi_i\rangle \rightarrow \mathbf{skip})) \mathbf{fiq}$$

The following example illustrates how some frequently used (multi-qubit) quantum gates can be defined in terms of quantum **if**-statements.

EXAMPLE 3.1. Assume that the single-qubit NOT matrix  $X$ , and  $\bar{r}_x(\theta) = iR_x(2\theta)$  are unitary matrix constants in  $\mathcal{U}$ , where  $R_x(\theta)$  is the rotation about the  $X$  axis:

$$R_x(\theta) = \begin{pmatrix} \cos \frac{\theta}{2} & -i \sin \frac{\theta}{2} \\ -i \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{pmatrix}.$$

Then:

- (1) The CNOT (controlled-NOT) gate with  $q_1$  as its control qubit can be defined by

$$\text{CNOT}[q_1, q_2] \triangleq \mathbf{qif} [q_1] (|1\rangle \rightarrow X[q_2]) \mathbf{fiq}.$$

- (2) The Toffoli gate with  $q_1, q_2$  as its control qubits is defined by

$$\text{Toffoli}[q_1, q_2, q_3] \triangleq \mathbf{qif} [q_1, q_2] (|11\rangle \rightarrow X[q_3]) \mathbf{fiq}.$$

- (3) The Deutsch gate with  $q_1, q_2$  as its control qubits is defined by

$$\text{Deutsch}(\theta)[q_1, q_2, q_3] \triangleq \mathbf{qif} [q_1, q_2] (|11\rangle \rightarrow \bar{r}_x(\theta)[q_3]) \mathbf{fiq}.$$

Note that  $\text{Deutsch}(\frac{\pi}{2}) = \text{Toffoli}$ .

- (4) The Fredkin gate with  $q_1$  as its control qubit is defined by

$$\text{Fredkin}[q_1, q_2, q_3] \triangleq \mathbf{qif} [q_1] (|1\rangle \rightarrow \text{SWAP}[q_2, q_3]) \mathbf{fiq}$$

where the SWAP gate is defined by

$$\text{SWAP}[q_2, q_3] \triangleq \text{CNOT}[q_2, q_3]; \text{CNOT}[q_3, q_2]; \text{CNOT}[q_2, q_3].$$

### 3.2 Quantum while-programs

Upon the layer of quantum circuits described in the previous subsection, the second layer consists of purely quantum programs, namely, quantum programs without classical variables. They are written in the quantum **while**-language as follows.

DEFINITION 3.2. *Quantum programs*  $P \in \mathbf{QProg}$  are defined by the syntax<sup>1</sup>:

$$P ::= \mathbf{skip} \mid \mathbf{abort} \mid \bar{q} := |\psi\rangle \mid C \mid P_1; P_2 \\ \mid \mathbf{if} M[\bar{q}] (\square m \rightarrow P_m) \mathbf{fi} \mid \mathbf{while} M[\bar{q}] = 1 \mathbf{do} P \mathbf{od}. \quad (10)$$

Here,  $\bar{q} := |\psi\rangle$  means that the quantum register (i.e., a sequence of quantum variables)  $\bar{q}$  is initialised in state  $|\psi\rangle$ .  $C$  stands for an arbitrary quantum circuit defined in Eq. (7). **skip**, **abort** and sequential composition  $P_1; P_2$  are the same as in classical programming. Quantum measurement  $M$ , represented by a set of linear operators  $M = \{M_i\}_{i \in \mathcal{J}}$  satisfying the completeness equation  $\sum_{i \in \mathcal{J}} M_i^\dagger M_i = I$ , is employed as the guard of branch statements. The **if**-statement means that a measurement  $M$  is performed on quantum register  $\bar{q}$ , and if the outcome is  $m$ , then the corresponding subprogram  $P_m$  is executed next. It is fundamentally different from the quantum **if**-statement defined in the above subsection. The control flow of this **if**-statement is classical and determined by classical information; that is, the outcomes of measurement  $M$ . In sharp contrast, the control flow of a quantum **if**-statement is quantum, as discussed in the above subsection. We often abbreviate  $\mathbf{if} M[\bar{q}] (\square m \rightarrow P_m) \mathbf{fi}$  to  $\mathbf{if} [\bar{q}] (\square m \rightarrow P_m) \mathbf{fi}$  whenever  $M$  is the measurement in the computational basis of qubits. The **while**-loop performs a measurement (with only two possible outcomes 0 and 1) on  $\bar{q}$ ; if the outcome is 1 then the loop body  $P$  is executed, and if the outcome is 0 then the loop terminates. So, its control flow is also classical.

This simple quantum programming language will be expanded by adding recursion and non-deterministic choice in Sections 6 and 7, respectively.

### 3.3 Semantics

In this subsection, we briefly describe the denotational semantics of quantum circuits **QC** and purely quantum programs **QProg** (for more details, we refer to Chapter 3 of [77]). The correctness of our laws of quantum programming will be proved based on them. In Section 10, these semantics will be formalised in the Coq proof assistant so that the laws of quantum programming developed in this paper can be mechanically proved in Coq.

DEFINITION 3.3. *The denotational semantics*  $\llbracket C \rrbracket$  of a quantum circuit  $C \in \mathbf{QC}$  is a unitary operator on Hilbert space  $\mathcal{H}_{q_v(C)}$  inductively defined as follows:

- (1) If  $C = \mathbf{skip}$ , then  $\llbracket C \rrbracket = I$  (the identity operator);
- (2) If  $C = U[\bar{q}]$ , then  $\llbracket C \rrbracket = U_{\bar{q}}$ ;
- (3) If  $C = C_1; C_2$ , then  $\llbracket C \rrbracket = (\llbracket C_2 \rrbracket \otimes I_{q_v(C_1) \setminus q_v(C_2)}) (\llbracket C_1 \rrbracket \otimes I_{q_v(C_2) \setminus q_v(C_1)})$ ;
- (4) If  $C = \mathbf{qif} [\bar{q}] (\square_{i=1}^d |\psi_i\rangle_{\bar{q}} \rightarrow C_i) \mathbf{fiq}$ , then  $\llbracket C \rrbracket = \sum_{i=1}^d |\psi_i\rangle_{\bar{q}} \langle \psi_i| \otimes (\llbracket C_i \rrbracket \otimes I_{q_v(C) \setminus \bar{q} \setminus q_v(C_i)})$ .

Let  $\mathcal{D}(\mathcal{H}_{QV})$  denote the set of partial density operators (i.e., positive operators with traces  $\leq 1$ ) on  $\mathcal{H}_{QV}$ . Then the denotational semantics of a quantum program  $P \in \mathbf{QProg}$  was defined in [77] as a mapping  $\llbracket P \rrbracket : \mathcal{D}(\mathcal{H}_{QV}) \rightarrow \mathcal{D}(\mathcal{H}_{QV})$ . The Löwner order  $\sqsubseteq$  between operators on  $\mathcal{H}_{QV}$ , defined as  $A \sqsubseteq B$  if  $B - A$  is positive semi-definite, can be lifted pointwise to an order between quantum operations (i.e., super-operators), also denoted  $\sqsubseteq$ . We write  $\mathbf{QO}(\mathcal{H}_{QV})$  for the set of quantum operations on  $\mathcal{H}_{QV}$ . It can be proved that  $(\mathbf{QO}(\mathcal{H}_{QV}), \sqsubseteq)$  is a CPO (Complete Partial Order) [77]. Furthermore, the denotational semantics of quantum programs enjoys the following:

<sup>1</sup>Note that we use the same notations **skip** and  $;$  (sequential composition) at both the circuit and program layers. This does not cause ambiguity in semantics.

PROPOSITION 3.1 (STRUCTURAL REPRESENTATION). *For any input  $\rho$ , we have:*

- (1)  $\llbracket \text{skip} \rrbracket(\rho) = \rho; \llbracket \text{abort} \rrbracket(\rho) = 0.$
- (2)  $\llbracket q := |\psi\rangle \rrbracket(\rho) = \sum_n |\psi\rangle_q \langle n|\rho|n\rangle_q \langle \psi|$ , where  $\{|n\rangle\}$  is a given orthonormal basis of  $\mathcal{H}_q$ .
- (3)  $\llbracket C \rrbracket(\rho) = \llbracket C \rrbracket \rho \llbracket C \rrbracket^{\dagger 2}.$
- (4)  $\llbracket P_1; P_2 \rrbracket(\rho) = \llbracket P_2 \rrbracket(\llbracket P_1 \rrbracket(\rho)).$
- (5)  $\llbracket \text{if } M[\bar{q}] (\square m \rightarrow P_m) \text{ fi} \rrbracket(\rho) = \sum_m \llbracket P_m \rrbracket \left( (M_m)_{\bar{q}} \rho (M_m)_{\bar{q}}^{\dagger} \right).$
- (6) Let **while** stand for quantum loop **while**  $M[\bar{q}] = 1$  **do**  $P$  **od**. Then

$$\llbracket \text{while} \rrbracket(\rho) = \sum_{n=0}^{\infty} [\mathcal{E}_0 \circ (\llbracket P \rrbracket \circ \mathcal{E}_1)^n] = \bigsqcup_{k=0}^{\infty} \llbracket \text{while}^{(k)} \rrbracket(\rho), \quad (11)$$

where for every  $k \geq 0$ ,  $\text{while}^{(k)}$  is the first  $k$  iterations of the loop; that is,  $\llbracket \text{while}^{(k)} \rrbracket = \sum_{n=0}^{k-1} [\mathcal{E}_0 \circ (\llbracket P \rrbracket \circ \mathcal{E}_1)^n]$  with  $\mathcal{E}_i(\rho) = (M_i)_{\bar{q}} \rho (M_i)_{\bar{q}}^{\dagger}$  for  $i = 0, 1$ , and the symbol  $\bigsqcup$  stands for the supremum of quantum operations; i.e., the least upper bound in the CPO  $(\mathcal{QO}(\mathcal{H}_{QV}), \sqsubseteq)$ .

### 3.4 Further Expansion

As outlined in the Introduction, at the third layer of our framework, purely quantum programming language **QProg** defined in Subsection 3.2 is further embedded into a classical programming language (e.g., Dijkstra's nondeterministic sequential programming language [22] employed in Section 2 for formulating the laws of classical programming), and a program construct of the form  $x := M[\bar{q}]$  is used to link classical and quantum computing, where quantum measurement  $M$  is performed on quantum register  $\bar{q}$  and the outcome is stored in classical variable  $x$ . Thus, the laws of classical programming, like those presented in Figure 2, can be combined with our laws to be established in the sequent sections for reasoning about classical-quantum hybrid programs at this layer. Due to the limitation of space, we are not going to elaborate on such a combination in this paper.

## 4 Algebraic Laws for Quantum Circuits

Now we are ready to introduce our laws of quantum programming. As the first step, in this section, we prove a set of algebraic laws for quantum circuits in **QC**, constructed from basic gates given by  $\mathcal{U}$  using sequential composition and quantum **if**-statement. To describe these laws, we need the notion of equivalence between quantum circuits:

DEFINITION 4.1. *Let  $C_1, C_2 \in \mathcal{QC}$  be two quantum circuits. They are called equivalent, written  $C_1 \equiv C_2$ , if  $\llbracket C_1 \rrbracket \otimes I_{qV(C_2) \setminus qV(C_1)} = \llbracket C_2 \rrbracket \otimes I_{qV(C_1) \setminus qV(C_2)}$ .*

A general form of quantum **if**-statement defined by a quantum register is allowed in Definition 3.1. But for simplicity of presentation, we often only consider quantum **if**-statements defined by a single qubit. Following an idea in [32], we choose to use an infix notation for them. Let  $|\varphi_0\rangle, |\varphi_1\rangle$  be a basis of the Hilbert space of qubit  $q$ . Then

$$C_0 \xleftarrow{|\varphi_0\rangle} q \xrightarrow{|\varphi_1\rangle} C_1 \triangleq \mathbf{qif} [q] (|\varphi_0\rangle \rightarrow C_0 \square |\varphi_1\rangle \rightarrow C_1) \mathbf{fiq}. \quad (12)$$

In particular, whenever the basis  $\{|\varphi_0\rangle, |\varphi_1\rangle\}$  is fixed (e.g., the computational basis  $\{|0\rangle, |1\rangle\}$ ) or known from the context, then  $C_0 \xleftarrow{|\varphi_0\rangle} q \xrightarrow{|\varphi_1\rangle} C_1$  is abbreviated to  $C_0 \leftarrow q \rightarrow C_1$ . Furthermore,  $\mathbf{skip} \leftarrow q \rightarrow C_1$  is abbreviated to  $q \rightarrow C_1$ . We also write  $\bar{q} \xrightarrow{|\varphi\rangle} C$  for  $\mathbf{qif} [\bar{q}] (|\varphi\rangle \rightarrow C) \mathbf{fiq}$ , where

<sup>2</sup>We slightly abuse notation here:  $\llbracket C \rrbracket$  on the LHS denotes the program semantics, which is a quantum operation, whereas  $\llbracket C \rrbracket$  on the RHS denotes the circuit semantics, which is a unitary operator.

only the  $|\varphi\rangle$  branch is not **skip**. These notations will significantly simplify the presentation of our laws for quantum circuits.

#### 4.1 Basic Laws

Let us first present some basic laws for quantum circuits. For readability, the proofs of them are postponed to the Appendix. Our first proposition shows a group of laws for quantum **if**-statements.

PROPOSITION 4.1 (LAWS OF QUANTUM **if**-STATEMENT).

- (1) (*Changing Basis*) If unitary  $U$  transforms basis  $|\varphi_0\rangle, |\varphi_1\rangle$  to basis  $|\psi_0\rangle, |\psi_1\rangle$  up to a phase; that is,  $U|\varphi_i\rangle = \lambda_i|\psi_i\rangle$  ( $i = 0, 1$ ), then

$$C_0 \xleftarrow{|\psi_0\rangle} q \xrightarrow{|\psi_1\rangle} C_1 \equiv U^\dagger[q]; C_0 \xleftarrow{|\varphi_0\rangle} q \xrightarrow{|\varphi_1\rangle} C_1; U[q].$$

- (2) (*Symmetry*)  $C_0 \xleftarrow{|\varphi_0\rangle} q \xrightarrow{|\varphi_1\rangle} C_1 \equiv C_1 \xleftarrow{|\varphi_1\rangle} q \xrightarrow{|\varphi_0\rangle} C_0$ .

- (3) (*Idempotence*)  $C \leftarrow q \rightarrow C \equiv C$ .

- (4) (*Distributivity*) If  $q_2 \notin qv(C)$ , then

$$C \leftarrow q_1 \rightarrow (C_0 \leftarrow q_2 \rightarrow C_1) \equiv (C \leftarrow q_1 \rightarrow C_0) \leftarrow q_2 \rightarrow (C \leftarrow q_1 \rightarrow C_1).$$

- (5) (*Nested-**qif***)  $(C_{00} \leftarrow q_2 \rightarrow C_{01}) \leftarrow q_1 \rightarrow (C_{10} \leftarrow q_2 \rightarrow C_{11}) \equiv \mathbf{qif} [q_1, q_2] (|i, j\rangle \rightarrow C_{ij}) \mathbf{fiq}$ .

Clause (1) in the above proposition is quantum-specific and has no classical counterpart. Clauses (2) to (5) are generalisations of classical laws (If-3), (If-5), (If-6) and (If-8) in Figure 2.

The next proposition gives several laws for the sequential composition of quantum circuits.

PROPOSITION 4.2 (LAWS OF SEQUENTIAL COMPOSITION).

- (1) (*Unit*) **skip**;  $C \equiv C$ ; **skip**  $\equiv C$ .

- (2) (*Identity*)  $I[\bar{q}] \equiv \mathbf{skip}$ , where  $I$  is the identity operator.

- (3) (*Sequential Composition*)  $U_1[\bar{q}]; U_2[\bar{q}] \equiv (U_2U_1)[\bar{q}]$  if  $U_2U_1 \in \mathcal{U}$ .

- (4) (*Parallel Composition*)  $U_1[\bar{q}]; U_2[\bar{r}] \equiv U_2[\bar{r}]; U_1[\bar{q}] \equiv (U_1 \otimes U_2)[\bar{q}, \bar{r}]$  if  $\bar{q} \cap \bar{r} = \emptyset$  and  $U_1 \otimes U_2 \in \mathcal{U}$ .

- (5) (*Commutativity*) If  $qv(C_1) \cap qv(C_2) = \emptyset$ , then  $C_1; C_2 \equiv C_2; C_1$ .

- (6) (*Associativity*)  $(C_1; C_2); C_3 \equiv C_1; (C_2; C_3)$ .

- (7) (*Sequentiality*)  $(C_0 \leftarrow q \rightarrow C_1); (D_0 \leftarrow q \rightarrow D_1) \equiv (C_0; D_0) \leftarrow q \rightarrow (C_1; D_1)$ .

- (8) (*Distributivity over **qif***) If  $q \notin qv(C)$ , then

$$C; (C_0 \leftarrow q \rightarrow C_1) = (C; C_0) \leftarrow q \rightarrow (C; C_1), \quad (C_0 \leftarrow q \rightarrow C_1); C = (C_0; C) \leftarrow q \rightarrow (C_1; C).$$

It is obvious that Clauses (1), (6) and (8) in the above proposition are generalisations of classical laws (Sq-1), (Sq-3) and (Sq-4) in Figure 2, respectively. As a corollary of the law (Sequentiality), we have:

$$(\text{Merging}) \quad \bar{q} \xrightarrow{|\varphi\rangle} (C_1; C_2) \equiv (\bar{q} \xrightarrow{|\varphi\rangle} C_1); (\bar{q} \xrightarrow{|\varphi\rangle} C_2) \quad (13)$$

$$(\text{Splitting}) \quad \mathbf{qif} [\bar{q}] \left( \square_{i=1}^d |\varphi_i\rangle \rightarrow C_i \right) \mathbf{fiq} \equiv (\bar{q} \xrightarrow{|\varphi_1\rangle} C_1); \dots; (\bar{q} \xrightarrow{|\varphi_d\rangle} C_d). \quad (14)$$

A particularly useful syntactic sugar is quantum choice defined in [78] as follows: for any quantum circuit  $C \in \mathbf{QC}$  with  $qv(C) = \{q\}$ ,

- (1) (Pre-choice)  $C_0 \leftarrow C[q] \rightarrow C_1 \triangleq C; (C_0 \leftarrow q \rightarrow C_1)$ .

- (2) (Post-choice)  $C_0 \leftarrow [q]C \rightarrow C_1 \triangleq (C_0 \leftarrow q \rightarrow C_1); C$ .

We often call  $C$  the coin tossing circuit of the choice. Intuitively, in the pre-choice,  $C$  tosses quantum choice  $q$  so that  $q$  is in a superposition, which then determines the control flow of quantum if-statement  $C_0 \leftarrow q \rightarrow C_1$ . In the post-choice, the coin tossing operation  $C$  mixes the execution paths represented by the quantum if-statement. Quantum choice can be seen as a quantum counterpart of probabilistic choice in the probabilistic programming language pGCL [35]. The following example of a quantum random walk shows an application of quantum choice. Indeed, the notion of quantum choice introduced in [78] was inspired by the construction of quantum random walks.

**EXAMPLE 4.1 (QUANTUM RANDOM WALK).** *The one-dimensional random walk can be thought of as a particle moving on a discrete line whose nodes are denoted by integers  $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$ . At each step, the particle moves one position left or right, depending on the flip of a coin. A quantum variant of it is the Hadamard walk, with a quantum coin operator given by the Hadamard gate  $H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ . Formally, the Hilbert space of the Hadamard walk is  $\mathcal{H}_d \otimes \mathcal{H}_p$ , where  $\mathcal{H}_d = \text{span}\{|L\rangle, |R\rangle\}$  is a 2-dimensional Hilbert space, called the direction space,  $|L\rangle, |R\rangle$  are used to indicate the direction Left and Right, respectively,  $\mathcal{H}_p = \overline{\text{span}\{|n\rangle : n \in \mathbb{Z}\}}$  is an infinite-dimensional Hilbert space,  $|n\rangle$  indicates the position marked by integer  $n$ , and  $\overline{\text{span } X}$  is the Hilbert space spanned by a nonempty set  $X$  of vectors. One step of the Hadamard walk is modelled by the unitary operator  $W = T(H \otimes I_p)$ , where the translation  $T$  is a unitary operator on  $\mathcal{H}_d \otimes \mathcal{H}_p$  defined by  $T|L, n\rangle = |L, n-1\rangle$  and  $T|R, n\rangle = |R, n+1\rangle$  for every  $n \in \mathbb{Z}$ ,  $H$  is the Hadamard gate on  $\mathcal{H}_d$ , and  $I_p$  is the identity operator on  $\mathcal{H}_p$ . The Hadamard walk is then described by repeated applications of the operator  $W$ .*

Now we define the left and right translation operators  $T_L$  and  $T_R$  on the position space  $\mathcal{H}_p$  by  $T_L|n\rangle = |n-1\rangle$  and  $T_R|n\rangle = |n+1\rangle$  for each  $n \in \mathbb{Z}$ . The translation can be written as quantum if-statement  $T = T_L \leftarrow d \rightarrow T_R$ , the walk operator is a quantum choice  $W = T_L[p] \leftarrow H[d] \rightarrow T_R[p]$ , and an  $n$ -step quantum walk is the sequential composition of  $n$  copies of  $W$ .

Several laws for quantum choices are presented in the following:

**PROPOSITION 4.3 (LAWS OF QUANTUM CHOICE).**

(1) (Symmetry) Let  $X$  be the NOT gate. Then

$$C_0 \leftarrow X[q] \rightarrow C_1 \equiv C_1 \leftarrow [q]X \rightarrow C_0.$$

More generally, if  $U|\varphi_i\rangle = \lambda_i|\psi_i\rangle$  for  $(i = 0, 1)$ , then

$$C_0 \xleftarrow{|\psi_0\rangle} U[q] \xrightarrow{|\psi_1\rangle} C_1 \equiv C_0 \xleftarrow{|\varphi_0\rangle} [q]U \xrightarrow{|\varphi_1\rangle} C_1.$$

(2) (Sequentiality)  $(C_0 \leftarrow B[q] \rightarrow C_1); (D_0 \leftarrow q \rightarrow D_1) \equiv (C_0; D_0) \leftarrow B[q] \rightarrow (C_1; D_1)$ ,

$$(C_0 \leftarrow q \rightarrow C_1); (D_0 \leftarrow [q]B \rightarrow D_1) \equiv (C_0; D_0) \leftarrow [q]B \rightarrow (C_1; D_1).$$

(3) (Distributivity) If  $q \notin \text{qv}(C)$ , then

$$C; (C_0 \leftarrow B[q] \rightarrow C_1) \equiv (C; C_0) \leftarrow B[q] \rightarrow (C; C_1),$$

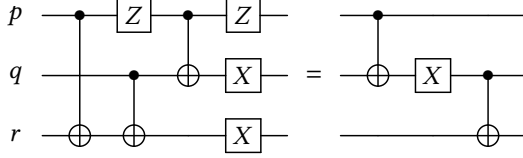
$$(C_0 \leftarrow B[q] \rightarrow C_1); C \equiv (C_0; C) \leftarrow B[q] \rightarrow (C_1; C).$$

The same holds for post-choice.

It is easy to see the correspondence between the laws (Symmetry), (Sequentiality) and (Distributivity) in the above proposition and the ones in Propositions 4.1 and 4.2. On the other hand, the laws in the above proposition can be regarded as quantum counterparts of the corresponding laws of probabilistic programming given in [35].

To show that the algebraic laws proposed above are actually useful for circuit optimisation, let us see the following:

EXAMPLE 4.2 (CIRCUIT OPTIMISATION). We take an example from an earlier speech by Ross Duncan<sup>3</sup>:



It is used to motivate using ZX-calculus for circuit optimisation in [63]. The equivalence is not obvious at first glance. Let us write the two circuits in our language as:

$$p \rightarrow X[r]; Z[p]; q \rightarrow X[r]; p \rightarrow X[q]; Z[p]; X[q]; X[r] \equiv p \rightarrow X[q]; X[q]; q \rightarrow X[r].$$

We first check that:

$$\begin{aligned} & p \rightarrow X[r]; Z[p]; q \rightarrow X[r]; p \rightarrow X[q]; Z[p]; X[q]; X[r] \\ \equiv & p \rightarrow X[r]; q \rightarrow X[r]; (Z^\dagger[p]; p \rightarrow X[q]; Z[p]); X[q]; X[r] && \text{(By Prop. 4.2(5, 6); } Z^\dagger = Z) \\ \equiv & p \rightarrow X[r]; q \rightarrow X[r]; p \rightarrow X[q]; X[q]; X[r] && \text{(By Prop. 4.1(1))} \\ \equiv & (q \rightarrow X[r]; X[q]; X[r]) \leftarrow p \rightarrow (X[r]; q \rightarrow X[r]; X[q]; X[r]). && \text{(By Props. 4.1(3), 4.2(7))} \end{aligned}$$

Next, we simplify the sub-circuits as follows:

$$\begin{aligned} & q \rightarrow X[r]; X[q]; X[r] \\ \equiv & X[q]; X[r] \leftarrow q; X[r] && \text{(By Prop. 4.3(1))} \\ \equiv & X[q]; (X[r]; X[r]) \leftarrow q \rightarrow X[r] && \text{(By Props. 4.1(3), 4.2(7))} \\ \equiv & X[q]; q \rightarrow X[r], && \text{(By Prop. 4.2(3, 2); } XX = I) \end{aligned}$$

and

$$\begin{aligned} & X[r]; q \rightarrow X[r]; X[q]; X[q]; X[r] \\ \equiv & X[r]; q \rightarrow X[r]; X[r] && \text{(By Prop. 4.2(3, 2); } XX = I) \\ \equiv & (X[r]; X[r]) \leftarrow q \rightarrow (X[r]; X[r]; X[r]) && \text{(By Props. 4.1(3), 4.2(7))} \\ \equiv & q \rightarrow X[r] && \text{(By Prop. 4.2(3, 2); } XX = I) \\ \equiv & X[q]; X[q]; q \rightarrow X[r]. && \text{(By Prop. 4.2(3, 2); } XX = I) \end{aligned}$$

Plug them into the previous formula:

$$\begin{aligned} & p \rightarrow X[r]; Z[p]; q \rightarrow X[r]; p \rightarrow X[q]; Z[p]; X[q]; X[r] \\ \equiv & (X[q]; q \rightarrow X[r]) \leftarrow p \rightarrow (X[q]; X[q]; q \rightarrow X[r]) \\ \equiv & p \rightarrow X[q]; (X[q]; q \rightarrow X[r]) \leftarrow p \rightarrow (X[q]; q \rightarrow X[r]) && \text{(By Prop. 4.2(1,7))} \\ \equiv & p \rightarrow X[q]; X[q]; q \rightarrow X[r] && \text{(By Prop. 4.1(3))} \end{aligned}$$

which completes the proof.

## 4.2 Normal Forms

In this subsection, we establish a normal form for a large class of quantum circuits using the laws presented in the previous subsection. For each quantum variable  $q \in QV$ , we choose a fixed orthonormal basis of  $\mathcal{H}_q$  as the standard basis of  $q$ . The standard basis of a quantum register  $\bar{q}$  is then defined as the set of tensor products of the standard basis states of variables in  $\bar{q}$ ; for example, the computational basis  $\{|i\rangle : i \in \{0, 1\}^n\}$  of  $n$  qubits.

<sup>3</sup><https://www.youtube.com/watch?v=QbtSSqeYuFM#t=01h35m30s>

- DEFINITION 4.2.** (1) A flat quantum if-statement is a quantum circuit without nested quantum if-statements, i.e., of the form  $\mathbf{qif} [\bar{q}] (\square_i |i\rangle \rightarrow G_i) \mathbf{fiq}$ , where  $\{|i\rangle\}$  is the standard basis of  $\bar{q}$ , and each  $G_i$  is a sequential composition of basic quantum gates or **skip**.
- (2) A normal form of the quantum circuit is a sequential composition of the form  $C_1; \dots; C_n$ , where  $C_i$  is a flat quantum if-statement for each  $1 \leq i \leq n$ .

**DEFINITION 4.3.** A quantum circuit  $C \in \mathbf{QC}$  is called regular if for every guard basis  $\{|\psi_i\rangle\}$  of a quantum if-statement in  $C$ , the unitary transformation from  $\{|\psi_i\rangle\}$  to the standard basis of the guard register  $\bar{q}$  and its inverse, i.e., unitary transformations  $\sum_i |\psi_i\rangle\langle i|$  and  $\sum_i |i\rangle\langle \psi_i|$ , can be implemented by a sequence of basic gates or **skip**.

The following normal form theorem eliminates nested quantum if-statements in a quantum circuit and reduces them to flat (but large) quantum if-statements. We use the term “normal form” in the sense of [32, 33], without requiring uniqueness—similar to conjunctive or disjunctive normal forms in propositional logic. This differs from the notion of normal forms in rewriting theory.

**THEOREM 4.1 (NORMAL FORM).** Each regular quantum circuit  $C \in \mathbf{QC}$  is equivalent to a normal form.

**PROOF.** We proceed by induction on the structure of  $C$ .

**Case 1.**  $C = U[\bar{q}]$  is a basic gate. Then we introduce an external quantum variable  $r \notin \bar{q}$ . It follows from a simple generalisation of the idempotence law given in Proposition 4.1(3) that  $C \equiv \mathbf{qif} [r] (\square_i |i\rangle \rightarrow G_i) \mathbf{fiq}$  where  $\{|i\rangle\}$  is the standard basis of  $r$ , and  $G_i = U[\bar{q}]$  for all  $i$ .

**Case 2.**  $C = C_1; C_2$ , where both  $C_1 \equiv C_{11}; \dots; C_{1m}$  and  $C_2 \equiv C_{21}; \dots; C_{2n}$  are in a normal form; that is, all  $C_{1i}$  ( $1 \leq i \leq m$ ) and  $C_{2j}$  ( $1 \leq j \leq n$ ) are flat quantum if-statements. Then it is obvious that  $C \equiv C_{11}; \dots; C_{1m}; C_{21}; \dots; C_{2n}$  is a normal form.

**Case 3.**  $C = \mathbf{qif} [\bar{q}] (\square_{i=1}^d |\psi_i\rangle \rightarrow C_i) \mathbf{fiq}$ , where each  $C_i$  is in a normal form, i.e.,  $C_i \equiv C_{i1}; \dots; C_{im_i}$  with each  $C_{ij}$  being a flat quantum if-statement. First, let  $\{|i\rangle\}$  be the standard basis of  $\bar{q}$ , and let  $U$  be the unitary transformation such that  $|\psi_i\rangle = U|i\rangle$  for all  $i$ . Since  $C$  is regular, there exist two sequential compositions  $G_1, G_2$  of basic gates such that  $\llbracket G_1 \rrbracket = U$  and  $\llbracket G_2 \rrbracket = U^\dagger$ . Then we have:

$$C \equiv G_2; \mathbf{qif} [\bar{q}] (\square_{i=1}^d |i\rangle \rightarrow C_i) \mathbf{fiq}; G_1. \quad (15)$$

The equivalence (15) is indeed a generalisation of the law of changing basis given in Proposition 4.1(1). To prove (15), we note that  $qv(G_1) \subseteq \bar{q}$ ,  $qv(G_2) \subseteq \bar{q}$  and  $\bar{q} \cap (\cup_i qv(C_i)) = \emptyset$ . Thus, it holds that

$$\begin{aligned} \llbracket C \rrbracket &= \sum_i |\psi_i\rangle\langle \psi_i| \otimes \llbracket C_i \rrbracket = \sum_i (U|i\rangle\langle i|U^\dagger \otimes \llbracket C_i \rrbracket) \\ &= U \left( \sum_i |i\rangle\langle i| \otimes \llbracket C_i \rrbracket \right) U^\dagger = \llbracket G_2 \rrbracket \llbracket \mathbf{qif} [\bar{q}] (\square_{i=1}^d |i\rangle \rightarrow C_i) \mathbf{fiq} \rrbracket \llbracket G_1 \rrbracket = \llbracket RHS \rrbracket. \end{aligned}$$

Next, by applying Proposition 4.1(14) and 4.2(7), we rewrite  $C$  as

$$C \equiv G_2; ((\bar{q} \xrightarrow{|1\rangle} C_{11}); \dots; (\bar{q} \xrightarrow{|1\rangle} C_{1m_1})); \dots; ((\bar{q} \xrightarrow{|d\rangle} C_{d1}); \dots; (\bar{q} \xrightarrow{|d\rangle} C_{dm_d})); G_1.$$

Now, by the conclusions of Case 2, it is sufficient to show that each  $\bar{q} \xrightarrow{|i\rangle} C_{ij}$  is equivalent to a normal form. Since  $C_{ij}$  is a flat quantum if-statement, we can assume that  $C_{ij} \equiv \mathbf{qif} [\bar{r}] (\square_{k=1}^{d_r} |k\rangle \rightarrow D_k) \mathbf{fiq}$ , where  $D_k$  is a sequence of basic gates or **skip**. Then we claim

$$\bar{q} \xrightarrow{|i\rangle} C_{ij} \equiv \mathbf{qif} [\bar{q}, \bar{r}] (\square_{k=1}^{d_r} |i, k\rangle \rightarrow D_k) \mathbf{fiq},$$

where the RHS is a normal form. To show this, note that  $\bar{q} \cap \bar{r} = \emptyset$  and  $(\bar{q}, \bar{r}) \cap qv(D_k) = \emptyset$ . This can be rigorously proved by showing that the normal form we constructed for each sub-program shares the same variable sets with the original sub-program, except the external variable  $r_0$  we used in Case 1, which can be chosen disjoint from the original program. We omit the tedious details here, but the reader can see them from the Coq implementation discussed in Section 10. Thus, we have:

$$\begin{aligned}
\llbracket \bar{q} \xrightarrow{|i\rangle} C_{ij} \rrbracket &= |i\rangle_{\bar{q}} \langle i| \otimes \llbracket C_{ij} \rrbracket + \sum_{i' \neq i} |i'\rangle_{\bar{q}} \langle i'| \otimes I \\
&= |i\rangle_{\bar{q}} \langle i| \otimes \sum_k |k\rangle_{\bar{r}} \langle k| \otimes \llbracket D_k \rrbracket + \sum_{i' \neq i} |i'\rangle_{\bar{q}} \langle i'| \otimes I \\
&= \sum_k |i, k\rangle_{\bar{q}, \bar{r}} \langle i, k| \otimes \llbracket D_k \rrbracket + \sum_{i' \neq i} \sum_k |i', k\rangle_{\bar{q}} \langle i', k| \otimes I \\
&= \llbracket \mathbf{qif} [\bar{q}, \bar{r}] \left( \square_{k=1}^{d_r} |i, k\rangle \rightarrow D_k \right) \mathbf{fiq} \rrbracket.
\end{aligned}$$

□

An application of the above normal form is equivalence checking of two quantum circuits  $C_1$  and  $C_2$ , which is a key issue in the design and verification of quantum circuits and has been widely studied in the literature. We can transform them to two normal forms  $C'_1$  and  $C'_2$ , respectively. Obviously, it is much easier to check the equivalence of  $C'_1$  and  $C'_2$  than that of  $C_1$  and  $C_2$ . This approach is different from the existing ones in the literature (see for example [14, 72]). A general exposition of this application is out of the scope of the present paper. Here, we only give a very simple example:

**EXAMPLE 4.3 (EQUIVALENT CHECKING VIA NORMAL FORM).** *Consider the following two quantum circuits  $C_1$  and  $C_2$ :*

$$\begin{aligned}
C_1 &\triangleq H[y] \xleftarrow{|-\rangle} x \xrightarrow{|+\rangle} (\mathbf{skip} \xleftarrow{|-\rangle} y \xrightarrow{|+\rangle} X[z]; H[y]), \\
C_2 &\triangleq H[x]; H[y]; y \rightarrow (x \rightarrow X[z]); H[x].
\end{aligned}$$

*Since  $C_1$  and  $C_2$  contain nested quantum if-statements and are not in the standard basis, the equivalence is not obvious. We translate them into normal forms. Note that the proof of Theorem 4.1 is not the unique way to rewrite the circuits; here we proceed in a slightly different and simpler manner:*

$$\begin{aligned}
C_1 &\equiv H[y] \xleftarrow{|-\rangle} x \xrightarrow{|+\rangle} (H[y]; y \rightarrow X[z]; H[y]; H[y]) && \text{(By Prop. 4.1(1); } H^\dagger = H) \\
&\equiv H[x]; H[y] \leftarrow x \rightarrow (H[y]; y \rightarrow X[z]; H[y]; H[y]); H[x] && \text{(By Prop. 4.1(1); } H^\dagger = H) \\
&\equiv H[x]; H[y] \leftarrow x \rightarrow (H[y]; y \rightarrow X[z]); H[x] && \text{(By Prop. 4.2(3, 4); } HH = I) \\
&\equiv H[x]; H[y]; x \rightarrow (y \rightarrow X[z]); H[x] && \text{(By Prop. 4.2(7, 8))} \\
&\equiv H[x]; H[y]; y \rightarrow (x \rightarrow X[z]); H[x] && \text{(By Props. 4.1(4, 3))} \\
&\equiv H[x]; H[y]; \mathbf{qif} [y, x] (|11\rangle \rightarrow X[z]) \mathbf{fiq}; H[x], && \text{(By Prop. 4.1(5))} \\
C_2 &\equiv H[x]; H[y]; \mathbf{qif} [y, x] (|11\rangle \rightarrow X[z]) \mathbf{fiq}; H[x]. && \text{(By Prop. 4.1(5))}
\end{aligned}$$

*Now their equivalence is trivial, since they share the same normal form.*

An approach to classical compiler design based on normal forms of programs was proposed in [33] and extended to quantum programs in [86]. Thus, another possible application of Theorem 4.1 is to combine it with the normal form of quantum programs (Theorem 5.1) for quantum compiler design.

## 5 Algebraic Laws for Quantum Programs

In this section, we turn to investigate laws for purely quantum programs in **QProg**. To present them, we need the notion of equivalence of two quantum programs:

**DEFINITION 5.1.** *Two quantum programs  $P$  and  $Q$  are called equivalent, written  $P \equiv Q$ , if they have the same denotational semantics; that is,  $\llbracket P \rrbracket(\rho) = \llbracket Q \rrbracket(\rho)$  for all inputs  $\rho \in \mathcal{D}(\mathcal{H}_{QV})$ .*

First of all, by induction on the structure of  $P$ , it is routine to prove the following lifting lemma, which gives a link from the equivalence of quantum programs to the equivalence of quantum circuits.

**LEMMA 5.1 (CONGRUENCE - LIFTING LAW).** *Let quantum program  $P = P(C_1, \dots, C_n)$  contain quantum circuits  $C_1, \dots, C_n$ , and let  $Q = P(D_1, \dots, D_n)$  be the quantum program obtained through substituting  $C_1, \dots, C_n$  in  $P$  by quantum circuits  $D_1, \dots, D_n$ , respectively. If  $C_i \equiv D_i$  for  $i = 1, \dots, n$ , then  $P \equiv Q$ .*

This lemma enables us to employ the laws of quantum circuits established in the last section in reasoning about the equivalence of quantum programs.

The **if**-statement in Eq. (10) is defined for a general quantum measurement. For simplicity of presentation, we mainly consider a special class of measurements in our laws for quantum programs. By the term *binary measurement*, we mean a measurement  $M$  with only two possible outcomes, say 0 and 1, which can then be written as  $M = \{M_0, M_1\}$ ; for instance, the measurement on a qubit in the computational basis. Following [32] again, we introduce an infix notation for an **if**-statement defined by a binary measurement  $M = \{M_0, M_1\}$ :

$$P_0 \triangleleft M[q] \triangleright P_1 \triangleq \text{if } M[q] \text{ (} 0 \rightarrow P_0 \square 1 \rightarrow P_1 \text{) fi.}$$

We often drop the measured quantum variable  $q$  and simply write  $P_0 \triangleleft M \triangleright P_1$  for  $P_0 \triangleleft M[q] \triangleright P_1$  when the exact name  $q$  is not essential. In particular,

- **skip**  $\triangleleft M \triangleright P$  is often abbreviated to  $M \triangleright P$ ;
- $M \triangleright$  **skip** is further abbreviated to  $[M]$ ;
- **abort**  $\triangleleft M \triangleright$  **skip** is abbreviated to  $(M)$ ;
- **skip**  $\triangleleft M \triangleright$  **abort** is abbreviated to  $[M]$ .

Intuitively,  $[M]$  can be understood as a test or assertion using a measurement  $M$  where we care about both outcomes 0 and 1. But in the test  $(M)$  (respectively,  $[M]$ ), we only care the case of measurement outcome 1 (respectively, outcome 0). The three tests  $[M]$ ,  $(M)$ , and  $[M]$  are useful in testing, debugging, and runtime assertion of quantum programs.

### 5.1 Laws of Initialisation

Let us now present several laws of initialisation. A special kind of quantum measurement will be used in one of these laws:

**DEFINITION 5.2.** *We write  $M = [|\psi\rangle]$  for a binary measurement  $M = \{M_0, M_1\}$  if it is defined by a pure quantum state  $|\psi\rangle$  (normalized state vector), i.e., if  $M_0 = I - |\psi\rangle\langle\psi|$  and  $M_1 = |\psi\rangle\langle\psi|$ .*

Obviously, the measurement  $[|\psi\rangle]$  tests whether a quantum system is in state  $|\psi\rangle$  or not. The laws given in the following proposition describe how initialisation interacts with other program constructs.

**PROPOSITION 5.1 (LAWS OF INITIALIZATION).**

(1) (Cancellation)  $\bar{q} := |\varphi\rangle; \bar{q} := |\psi\rangle \equiv \bar{q} := |\psi\rangle$ . If  $\bar{q}_1 \cap \bar{q}_2 = \emptyset$ , then

$$\bar{q}_1 := |\phi\rangle; \bar{q}_2 := |\psi\rangle \equiv \bar{q}_1, \bar{q}_2 := |\phi\rangle|\psi\rangle.$$

In particular, if  $qv(\bar{q}) = qv(\bar{q}_1) \cup qv(\bar{q}_2)$ , then  $\bar{q}_1 := |0\rangle; \bar{q}_2 := |0\rangle \equiv \bar{q} := |0\rangle$ .

- (2) (*Unitary-Elimination*) If  $\llbracket C \rrbracket = U_{\bar{q}}$  for some unitary operator  $U$ , then  $\bar{q} := |\psi\rangle; C \equiv \bar{q} := U|\psi\rangle$ .  
In particular,  $\bar{q} := |\psi\rangle; U[\bar{q}] \equiv \bar{q} := U|\psi\rangle$ .
- (3) (*qif-Elimination*) We write  $|\alpha\rangle \perp |\beta\rangle$  if two states  $|\alpha\rangle, |\beta\rangle$  are orthogonal, i.e.,  $\langle \alpha | \beta \rangle = 0$ .

$$\frac{|\psi\rangle \perp |\varphi_0\rangle}{q := |\psi\rangle; C_0 \xleftarrow{|\varphi_0\rangle} q \xrightarrow{|\varphi_1\rangle} C_1 \equiv q := |\varphi_1\rangle; C_1 \equiv q := |\psi\rangle; C_1}$$

More generally, if for each  $i$ , there exists  $U_i$  such that  $\llbracket C_i \rrbracket = (U_i)_{\bar{r}}$ , then

$$\bar{q}, \bar{r} := \sum_i \lambda_i |\phi_i\rangle |\psi_i\rangle; \mathbf{qif} [\bar{q}] \left( \prod_{i=1}^d |\phi_i\rangle \rightarrow C_i \right) \mathbf{fiq} \equiv \bar{q}, \bar{r} := \sum_i \lambda_i |\phi_i\rangle (U_i |\psi_i\rangle).$$

- (4) (*if-Elimination*) The left-elimination

$$\llbracket M[q] \rrbracket; q := |\phi\rangle \equiv q := |\phi\rangle,$$

and the right-eliminations

$$\frac{M = [|\psi\rangle]}{q := |\psi\rangle; P_0 \triangleleft M[q] \triangleright P_1 \equiv q := |\psi\rangle; P_1}, \quad \frac{M = [|\psi\rangle] \quad |\varphi\rangle \perp |\psi\rangle}{q := |\varphi\rangle; P_0 \triangleleft M[q] \triangleright P_1 \equiv q := |\varphi\rangle; P_0}.$$

- (5) (*if-Expansion*) If  $\bar{q} \subseteq \bar{r}$  and for all  $m$ ,  $\bar{r} \cap qv(P_m) = \emptyset$ , then

$$\mathbf{if} [\bar{q}] \left( \square m \rightarrow P_m \right) \mathbf{fi}; \bar{r} := |0\rangle \equiv \mathbf{if} [\bar{r}] \left( \square k \rightarrow P_{k \downarrow \bar{q}} \right) \mathbf{fi}; \bar{r} := |0\rangle,$$

where  $m$  and  $k$  are bit strings, and  $k \downarrow \bar{q}$  denotes the restriction of  $k$  on  $\bar{q}$ . Intuitively, a register can be additionally measured before its initialisation.

## 5.2 Laws of if-statement

In this subsection, we give some laws of if-statements. To this end, we need the following:

**DEFINITION 5.3.** Given two linear operators  $A$  and  $B$  and non-negative real number  $k \in \mathbb{R}$ , we define:

- (1) ( *$k$ -proportion*)  $A \bowtie_k B$  if there exists  $c \in \mathbb{C}$  such that  $|c| = k$  and  $A = cB$ . Whenever  $k = 1$ , we simply write  $A \bowtie B$ .
- (2) ( *$k$ -left-absorption*)  $A \bowtie_k B \triangleq AB \bowtie_k B$ , i.e., if there exists  $c \in \mathbb{C}$  such that  $|c| = k$  and  $AB = cB$ . Whenever  $k = 1$ , we simply write  $A \bowtie B$ .
- (3) ( *$k$ -right-absorption*)  $A \bowtie_k B \triangleq AB \bowtie_k A$ , i.e., if there exists  $c \in \mathbb{C}$  such that  $|c| = k$  and  $AB = cA$ . Whenever  $k = 1$ , we simply write  $A \bowtie B$ .
- (4) (*Orthogonal*)  $A \perp B$  if  $AB = 0$ . It is obvious that  $A \perp B \leftrightarrow A \bowtie_0 B \leftrightarrow A \bowtie_0 B$ .

**DEFINITION 5.4.** Let  $M = \{M_0, M_1\}$  and  $N = \{N_0, N_1\}$  be two binary measurements. Then:

- (1)  $N$  is called the complement of  $M$ , written  $N = M^\perp$ , if  $N_0 = M_1$  and  $N_1 = M_0$ .
- (2) We say that  $M$  entails  $N$ , written  $M \blacktriangleright N$ , if  $N_0 \perp M_1$ .
- (3) We say  $M$  is logically weaker than  $N$ , written  $M \propto N$ , if  $M_1 \bowtie N_1$ . Whenever  $M \propto N^\perp$ , we simply write  $M \gg N$ , and say  $M$  logically contradicts to  $N$ .
- (4) A binary measurement  $K = \{K_0, K_1\}$  is called a pseudo-meet of  $M$  and  $N$ , written  $K \cong M \& N$ , if  $K_1 \bowtie N_1 M_1$  (or, equivalently,  $N_1 M_1 \bowtie K_1$ ).

The following example can help us understand the notions introduced in the above definition.

**EXAMPLE 5.1.** (1) *The intuition behind entailment, logical weakness and contradiction becomes clear when we consider projective measurements. Let two projective measurements  $M = \{M_0, M_1\}$  and  $N = \{N_0, N_1\}$  with  $M_0 = I - P_X$ ,  $M_1 = P_X$ ,  $N_0 = I - P_Y$  and  $N_1 = P_Y$ , where  $P_X, P_Y$  are the projections onto closed subspaces  $X$  and  $Y$ , respectively, then:*

- $M \blacktriangleright N$  if and only if  $X \subseteq Y$ ;
- $M \propto N$  if and only if  $Y \subseteq X$ ;
- $M \gg N$  if and only if  $X^\perp \subseteq Y$ .

Thus, the following properties hold for projective measurements  $M$  and  $N$ :

$$M \blacktriangleright N \Leftrightarrow N^\perp \blacktriangleright M^\perp \Leftrightarrow N \propto M \Leftrightarrow M^\perp \propto N^\perp \Leftrightarrow M^\perp \gg N \Leftrightarrow N \gg M^\perp.$$

- (2) In the definition of pseudo-meet  $K \cong M \& N$ , many different choices of  $K_0$  are possible; one of them is  $K_0 = \sqrt{M_0^\dagger M_0 + M_1^\dagger N_0^\dagger N_0 M_1}$ .
- (3)  $M \propto N$  ( $M$  is logically weaker than  $N$ ) if and only if  $N \cong N \& M$ . Also, we have:  $M \propto N$  implies  $N \blacktriangleright M$ .

Now we are able to present a set of simple laws for **if**-statements:

PROPOSITION 5.2 (LAWS OF **if**-STATEMENT).

- (1) (Truth-Falsity) For the trivial measurements  $M_T = \{M_{T0}, M_{T1}\}$  and  $M_F = \{M_{F0}, M_{F1}\}$  with  $M_{T0} = M_{F1} = 0$  and  $M_{T1} = M_{F0} = I$ , we have:

$$P \triangleleft M_T \triangleright Q \equiv Q, \quad P \triangleleft M_F \triangleright Q \equiv P.$$

- (2) (Idempotence)  $P \triangleleft M \triangleright P \equiv [M]; P$ , where  $[M]$  stands for test  $M \triangleright$  **skip**.
- (3) (Complementation)  $P \triangleleft M^\perp \triangleright Q \equiv Q \triangleleft M \triangleright P$ .
- (4) (Associativity) If  $M$  is a projective measurement, then

$$P \triangleleft M \triangleright (Q \triangleleft M \triangleright R) \equiv (P \triangleleft M \triangleright Q) \triangleleft M \triangleright R \equiv P \triangleleft M \triangleright R.$$

- (5) (if-Elimination)

$$\frac{M \blacktriangleright N \quad K \cong M \& N}{[M]; (P \triangleleft N \triangleright Q) \equiv [K]; Q}, \quad \frac{M \propto N}{P \triangleleft N \triangleright (Q \triangleleft M \triangleright R) \equiv P \triangleleft N \triangleright R}.$$

The laws given in the above proposition deserve some explanation. In particular, it is interesting to compare them with the corresponding laws of classical programming given in Figure 2. Clause (1) mimics the truth-falsity laws (If-1) and (If-2) in Figure 2. There is a basic difference between the idempotence law (If-3) in classical programming and that in quantum programming. In the classical case, we have  $P \triangleleft b \triangleright P \equiv P$ . In contrast, a testing  $[M]$  is added in the right-hand side of Clause (2). This difference comes from the fact that checking condition  $b$  does not change the state of classical program variables, but performing measurement  $M$  changes the state of quantum variable  $q$ . Clause (3) is a quantum generalisation of law (If-5). The associativity law (4) is a quantum generalisation of laws (If-4) and (If-7). Note that it only holds for a projective measurement  $M$ , and we can easily find a counter-example showing that it is not true for a general measurement  $M$ . Clause (5) is a complement of Proposition 5.1(4).

To present quantum generalisations of laws (If-6) and (If-8) in Figure 2 for nested **if**-statements, we need the following:

DEFINITION 5.5. Let  $M = \{M_0, M_1\}$  and  $N = \{N_0, N_1\}$  be two binary quantum measurements. Then we say that  $M$  and  $N$  left-commute (respectively, right-commute), written  $M \diamond_L N$  (respectively,  $M \diamond_R N$ ), if  $M_0 N_1 = N_1 M_0$  and  $M_1 N_1 = N_1 M_1$  (respectively,  $M_0 N_0 = N_0 M_0$  and  $M_1 N_0 = N_0 M_1$ ).

It is obvious that  $M \diamond_L N$  if and only if  $M^\perp \diamond_R N$ . The following example can help us to understand the commutativity introduced in the above definition.

EXAMPLE 5.2. If both  $M = \{M_0, M_1\}$  and  $N = \{N_0, N_1\}$  are projective measurements, say,  $M_0 = I - P_X$ ,  $M_1 = P_X$ ,  $N_0 = I - P_Y$  and  $N_1 = P_Y$ , where  $P_X$  and  $P_Y$  are the projection onto closed subspaces  $X, Y$ , respectively, then  $M \diamond_L N$  and  $M \diamond_R N$  if and only if  $P_X P_Y$  is also a projection. In this case,  $P_X P_Y = P_{X \cap Y}$ . Furthermore, if  $M \blacktriangleright N$  or  $M \propto N$ , then  $M \diamond_L N$ ,  $M \diamond_R N$ ,  $M \diamond_L N^\perp$  and  $M \diamond_R N^\perp$ .

Now our laws for nested **if**-statements in quantum programming can be given in the following:

PROPOSITION 5.3 (LAWS OF NESTED **if**-STATEMENT).

(1) (*Reduction*)

$$\frac{[N] \equiv [K] \triangleleft M \triangleright [L] \quad [N] \equiv [K] \triangleleft M \triangleright [L]}{(P \triangleleft K \triangleright Q) \triangleleft M \triangleright (P \triangleleft L \triangleright Q) \equiv P \triangleleft N \triangleright Q}.$$

For measurement in the computational basis, we further have: if  $\bar{q} \cap \bar{r} = \emptyset$ , then

$$\text{if } [\bar{q}] (\square m \rightarrow \text{if } [\bar{r}] (\square n \rightarrow P_{mn}) \mathbf{fi}) \mathbf{fi} \equiv \text{if } [\bar{q}, \bar{r}] (\square(m, n) \rightarrow P_{mn}) \mathbf{fi}.$$

(2) (*Left-Distributivity*)

$$\frac{M \diamond_L N \quad [N] \equiv [M]; [N]}{R \triangleleft N \triangleright (P \triangleleft M \triangleright Q) \equiv (R \triangleleft N \triangleright P) \triangleleft M \triangleright (R \triangleleft N \triangleright Q)}.$$

(3) (*Right-Distributivity*)

$$\frac{M \diamond_R N \quad [N] \equiv [M]; [N]}{(P \triangleleft M \triangleright Q) \triangleleft N \triangleright R \equiv (P \triangleleft N \triangleright R) \triangleleft M \triangleright (Q \triangleleft N \triangleright R)}.$$

(4) (*Left-Distributivity for Projection*) If  $M, N$  are projective measurements,

$$\frac{M \blacktriangleright N \quad \text{or} \quad M^\perp \blacktriangleright N}{R \triangleleft N \triangleright (P \triangleleft M \triangleright Q) \equiv (R \triangleleft N \triangleright P) \triangleleft M \triangleright (R \triangleleft N \triangleright Q)}.$$

(5) (*Right-Distributivity for Projection*) If  $M, N$  are projective measurements,

$$\frac{N \blacktriangleright M \quad \text{or} \quad N \blacktriangleright M^\perp}{(P \triangleleft M \triangleright Q) \triangleleft N \triangleright R \equiv (P \triangleleft N \triangleright R) \triangleleft M \triangleright (Q \triangleleft N \triangleright R)}.$$

Obviously, Clause (1) in the above proposition is a quantum generalisation of law (If-6) in Figure 2, and Clauses (2) and (3) are quantum generalisations of law (If-8) and its dual. But as one can expect, certain conditions about the involved measurements are needed in the quantum case. These laws will be very useful for reducing the number of **if**-statements in quantum program optimisation. For example, by applying equations in Clause (1) from left to right, and equations in Clauses (2) - (5) from right to left, the number of **if**-statements can be reduced.

### 5.3 Laws of sequential composition

Several basic laws for the sequential composition of quantum programs are collected in the following:

PROPOSITION 5.4 (LAWS OF SEQUENTIAL COMPOSITION).

(1) (*Unit and Zero*) **skip**;  $P \equiv P$ ; **skip**;  $P \equiv P$  and **abort**;  $P \equiv P$ ; **abort**;  $P \equiv \mathbf{abort}$ .

(2) (*Commutativity*) If  $qv(P_1) \cap qv(P_2) = \emptyset$ , then  $P_1; P_2 \equiv P_2; P_1$ .

(3) (*Associativity*)  $(P; Q); R \equiv P; (Q; R)$ .

(4) (*Sequentiality*) If  $M$  is a projective measurement, and  $\bar{q} \cap qv(P_0) = \bar{q} \cap qv(P_1) = \emptyset$ , then

$$(P_0 \triangleleft M[\bar{q}] \triangleright P_1); (Q_0 \triangleleft M[\bar{q}] \triangleright Q_1) \equiv (P_0; Q_0) \triangleleft M[\bar{q}] \triangleright (P_1; Q_1).$$

(5) (*Right-Distributivity*)  $(P_0 \triangleleft M \triangleright P_1); P \equiv (P_0; P) \triangleleft M \triangleright (P_1; P)$ .

(6) (*Left-Distributivity-I*) If  $\bar{q} \cap qv(P) = \emptyset$ , then

$$P; (P_0 \triangleleft M[\bar{q}] \triangleright P_1) \equiv (P; P_0) \triangleleft M[\bar{q}] \triangleright (P; P_1).$$

(7) (*Left-Distributivity-II*) Generally, we have:

$$\frac{P; [M] \equiv [N]; P' \quad P; [M] \equiv [N]; P'}{P; (P_0 \triangleleft M \triangleright P_1) \equiv (P'; P_0) \triangleleft N \triangleright (P'; P_1)}$$

In particular, if  $U$  is a unitary operator and  $U(M) = \{M'_0, M'_1\}$  is the binary measurement with  $M'_i = M_i U$  for  $i = 0, 1$ , then  $U[q]; (P_0 \triangleleft M[q] \triangleright P_1) \equiv P_0 \triangleleft U(M)[q] \triangleright P_1$ .

Clauses (1), (3) and (5) in the above proposition are quantum generalisations of classical laws (Sq-1), (Sq-2), (Sq-3) and (Sq-4) in Figure 2, respectively.

#### 5.4 Interplay between if-Statements and Quantum if-Statements

Up to now, in this section, all laws are concerned only with the second layer of purely quantum programs in our quantum programming framework. In this subsection, we present the following law that shows how a quantum if-statement is absorbed into an if-statement, and thus connects the second layer with the first layer of quantum circuits.

**PROPOSITION 5.5.** *Let  $\mathcal{B} = \{|\psi_i\rangle\}_{i=1}^d$  be an orthonormal basis and  $M = \{M_i\}$  be the measurement in basis  $\mathcal{B}$ ; that is,  $M_i = |\psi_i\rangle\langle\psi_i|$  for every  $i$ . Then*

$$\mathbf{qif} [\bar{q}] \left( \square_{i=1}^d |\psi_i\rangle \rightarrow C_i \right) \mathbf{fiq}; \mathbf{if} M[\bar{q}] (\square_i \rightarrow P_i) \mathbf{fi} \equiv \mathbf{if} M[\bar{q}] (\square_i \rightarrow C_i; P_i) \mathbf{fi}.$$

This law will play a crucial role in the application given in Section 9.

**EXAMPLE 5.3 (CORRECTNESS OF QUANTUM TELEPORTATION).** *Quantum teleportation is a simple yet important protocol that reveals how entanglement helps transmit quantum states without a quantum channel. It can be written in our language as follows:*

$$\begin{aligned} \mathbf{Tel} &\triangleq q := |0\rangle; r := |0\rangle; \mathbf{H}[q]; q \rightarrow \mathbf{X}[r]; \\ &p := a|0\rangle + b|1\rangle; p \rightarrow \mathbf{X}[q]; \mathbf{H}[p]; \\ &q \triangleright \mathbf{X}[r]; p \triangleright \mathbf{Z}[r]; p := |0\rangle; q := |0\rangle. \end{aligned}$$

The first line prepares the initial Bell state  $|\Phi^+\rangle \triangleq \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$  between qubits  $q, r$ . The second line prepares the state  $a|0\rangle + b|1\rangle$  in qubit  $p$  and performs the local computation. The third line sends the classical information and performs the recovery, and finally resets  $p, q$  to the default state  $|0\rangle$ . The correctness of the teleportation protocol can be specified as the following equivalence:

$$\mathbf{Tel} \equiv p := |0\rangle; q := |0\rangle; r := a|0\rangle + b|1\rangle,$$

which asserts that the final state of qubit  $r$  is the original state we want to transmit.

To prove it, we first show the functionality of the first line:

$$\begin{aligned} &q := |0\rangle; r := |0\rangle; \mathbf{H}[q]; q \rightarrow \mathbf{X}[r] \\ &\equiv q := \mathbf{H}|0\rangle; r := |0\rangle; q \rightarrow \mathbf{X}[r] && \text{(By Prop. 5.4(2, 3), 5.1(2))} \\ &\equiv q, r := \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle); q \rightarrow \mathbf{X}[r] && \text{(By Prop. 5.1(1))} \\ &\equiv q, r := \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle). && \text{(By Prop. 5.1(3, 2))} \end{aligned}$$

Next, we observe that the third line is equivalent to:

$$\begin{aligned} &q \triangleright \mathbf{X}[r]; p \triangleright \mathbf{Z}[r]; p := |0\rangle; q := |0\rangle \\ &\equiv q \rightarrow \mathbf{X}[r]; [M[q]]; p \rightarrow \mathbf{Z}[r]; [M[p]]; p := |0\rangle; q := |0\rangle && \text{(By Prop. 5.5)} \\ &\equiv q \rightarrow \mathbf{X}[r]; p \rightarrow \mathbf{Z}[r]; ([M[p]]; p := |0\rangle); ([M[q]]; q := |0\rangle) && \text{(By Prop. 5.4(2, 3))} \\ &\equiv q \rightarrow \mathbf{X}[r]; p \rightarrow \mathbf{Z}[r]; p := |0\rangle; q := |0\rangle. && \text{(By Prop. 5.1(4))} \end{aligned}$$

Finally, we combine these results to complete the proof:

$$\begin{aligned}
& q, r := |\Phi^+\rangle; p := a|0\rangle + b|1\rangle; p \rightarrow X[q]; H[p]; q \rightarrow X[r]; p \rightarrow Z[r]; p := |0\rangle; q := |0\rangle \\
& \equiv q, r := |\Phi^+\rangle; p := a|0\rangle + b|1\rangle; p \rightarrow X[q]; q \rightarrow X[r]; H[p]; p \rightarrow Z[r]; p := |0\rangle; q := |0\rangle \\
& \equiv q := |+\rangle; p, r := a|00\rangle + b|11\rangle; H[p]; p \rightarrow Z[r]; p := |0\rangle; q := |0\rangle \\
& \equiv q := |+\rangle; p, r := a|+0\rangle + b|-1\rangle; p \rightarrow Z[r]; p := |0\rangle; q := |0\rangle && \text{(By Prop. 5.1(2))} \\
& \equiv q := |+\rangle; p, r := a|+0\rangle + b|+1\rangle; p := |0\rangle; q := |0\rangle && \text{(By Prop. 5.1(3, 2))} \\
& \equiv q := |+\rangle; p := |+\rangle; r := a|0\rangle + b|1\rangle; p := |0\rangle; q := |0\rangle && \text{(By Prop. 5.1(1))} \\
& \equiv p := |+\rangle; p := |0\rangle; q := |+\rangle; q := |0\rangle; r := a|0\rangle + b|1\rangle && \text{(By Prop. 5.4(2, 3))} \\
& \equiv p := |0\rangle; q := |0\rangle; r := a|0\rangle + b|1\rangle, && \text{(By Prop. 5.1(1))}
\end{aligned}$$

where the first equality is due to Proposition 5.4(2, 3) and the second equality can be derived by

$$\begin{aligned}
& q, r := |\Phi^+\rangle; p := a|0\rangle + b|1\rangle; p \rightarrow X[q]; q \rightarrow X[r] \\
& \equiv p, q, r := \frac{a}{\sqrt{2}}|0\rangle(|00\rangle + |11\rangle) + \frac{b}{\sqrt{2}}|1\rangle(|00\rangle + |11\rangle); p \rightarrow X[q]; q \rightarrow X[r] && \text{(By Prop. 5.1(1))} \\
& \equiv p, q, r := \frac{a}{\sqrt{2}}(|000\rangle + |011\rangle) + \frac{b}{\sqrt{2}}(|110\rangle + |101\rangle); q \rightarrow X[r] && \text{(By Prop. 5.1(3, 2))} \\
& \equiv p, q, r := \frac{a}{\sqrt{2}}(|000\rangle + |010\rangle) + \frac{b}{\sqrt{2}}(|111\rangle + |101\rangle) && \text{(By Prop. 5.1(3, 2))} \\
& \equiv q := |+\rangle; p, r := a|00\rangle + b|11\rangle. && \text{(By Prop. 5.1(1)).}
\end{aligned}$$

## 5.5 Normal Forms of Finite, Purely Quantum Programs

In this subsection, we establish a normal form of purely quantum programs in  $\mathbf{QProg}$  using the laws introduced in the previous subsections.

**DEFINITION 5.6.** A normal form of quantum programs is an if-statement

$$\mathbf{if} M[\bar{q}] (\square m \rightarrow C_m) \mathbf{fi} \quad (16)$$

where each  $C_m$  is either **abort** or a quantum circuit.

**THEOREM 5.1 (NORMAL FORMS).** Each finite, purely quantum program (i.e., a program that does not contain iteration or recursion)  $P \in \mathbf{Qprog}$  is equivalent to a normal form.

**PROOF.** We proceed by induction on the structure of  $P$ .

**Case 1.**  $P = \mathbf{skip}$  or **abort**. Then we can choose a trivial measurement  $M = \{M_0, M_1\}$  with  $M_0 = I$  and  $M_1 = 0$ , where  $I, 0$  are the identity and zero operators, respectively, on the 2-dimensional Hilbert space. Let  $q$  be a qubit. Then we have  $\mathbf{skip} \equiv \mathbf{skip} \triangleleft M[q] \triangleright \mathbf{abort}$  and  $\mathbf{abort} \equiv \mathbf{abort} \triangleleft M[q] \triangleright \mathbf{skip}$ . Both of them are in a normal form.

**Case 2.**  $P = P_1; P_2$ , and  $P_1, P_2$  have the following normal forms:

$$P_1 \equiv \mathbf{if} M[\bar{q}] (\square m \rightarrow C_m) \mathbf{fi}, \quad P_2 \equiv \mathbf{if} N[\bar{r}] (\square n \rightarrow D_n) \mathbf{fi}.$$

For each  $m$ , let the semantics of quantum circuit  $C_m$  be a unitary operator  $U_m$ . Then for any density operator  $\rho$ , by the linearity and Proposition 3.1(4) and (5) we obtain:

$$\begin{aligned}
\llbracket P \rrbracket(\rho) &= \llbracket P_2 \rrbracket(\llbracket P_1 \rrbracket(\rho)) = \llbracket P_2 \rrbracket \left( \sum_m \llbracket C_m \rrbracket (M_m \rho M_m^\dagger) \right) = \sum_m \llbracket P_2 \rrbracket \left( \llbracket C_m \rrbracket (M_m \rho M_m^\dagger) \right) \\
&= \sum_m \left( \sum_n \llbracket D_n \rrbracket (N_n \llbracket C_m \rrbracket (M_m \rho M_m^\dagger) N_n^\dagger) \right) = \sum_{m,n} \llbracket D_n \rrbracket \left( (N_n U_m M_m) \rho (N_n U_m M_m)^\dagger \right).
\end{aligned}$$

For any  $m, n$ , we define operator  $K_{m,n} = N_n U_m M_m$ . Then by the unitarity of  $U_m$ , it is easy to check that  $\sum_{m,n} K_{m,n}^\dagger K_{m,n} = I$  and  $K = \{K_{m,n}\}$  is a quantum measurement. Thus, we have:

$$P \equiv \text{if } K[\bar{q} \cup \bar{r} \cup (\bigcup_m qv(C_m))] (\square(m, n) \rightarrow D_{m,n}) \text{ fi,}$$

which is in a normal form, where  $D_{m,n} = D_n$  for every  $m$ .

**Case 3.**  $P = \text{if } M[\bar{q}] (\square m \rightarrow P_m) \text{ fi}$ , and each  $P_m$  has normal form:

$$P_m \equiv \text{if } (\square n \in \Delta_m \cdot N_m[\bar{r}_m] = n \rightarrow C_{m,n}) \text{ fi.}$$

Then for any density operator  $\rho$ , with Proposition 3.1(6) we have:

$$\begin{aligned} \llbracket P \rrbracket(\rho) &= \sum_m \llbracket P_m \rrbracket (M_m \rho M_m^\dagger) = \sum_m \left( \sum_{n \in \Delta_m} \llbracket C_{m,n} \rrbracket (N_{m,n} M_m \rho M_m^\dagger N_{m,n}^\dagger) \right) \\ &= \sum_{(m,n) \in \Delta} \llbracket C_{m,n} \rrbracket \left( (N_{m,n} M_m) \rho (N_{m,n} M_m)^\dagger \right), \end{aligned}$$

where  $\Delta = \bigcup_m (\{m\} \times \Delta_m)$ . For each  $(m, n) \in \Delta$ , we define operator  $K_{m,n} = N_{m,n} M_m$ . Then it is easy to verify that  $\sum_{(m,n) \in \Delta} K_{m,n}^\dagger K_{m,n} = I$  and  $K = \{K_{m,n}\}_{(m,n) \in \Delta}$  is a quantum measurement. Consequently, it holds that

$$P \equiv \text{if } K[\bar{q} \cup (\bigcup_m \bar{r}_m)] (\square(m, n) \rightarrow C_{m,n}) \text{ fi,}$$

which is in a normal form. □

## 6 Laws for Recursion

In this section, we prove some laws for recursive quantum programs. The syntax and semantics of general recursive quantum programs (with classical control flows) were defined in [76, 77]. For simplicity of presentation, we consider a simple variant of them here. Let  $X$  be a program identifier, and let  $F(X)$  be a quantum program defined by adding  $X$  into the syntax (10). Then we write  $\mu X.F(X)$  for the recursive quantum program declared by the recursive equation  $X = F(X)$ . Intuitively, each occurrence of  $X$  within  $F(X)$  is a call on the recursive program.

For any program  $P$ , we write  $F(P) \triangleq F(X)[P/X]$  for the program obtained by replacing  $X$  in  $F(X)$  with  $P$ . Furthermore, we define  $F^{(n)}$  as the  $n$ th syntactic approximation of recursive quantum program  $\mu X.F(X)$ , i.e., the  $n$ -fold iteration of  $F$ :

$$\begin{cases} F^{(0)} = \text{abort,} \\ F^{(n+1)} = F(F^{(n)}) \text{ for } n \geq 0. \end{cases}$$

From its definition, we see that  $F^{(n)}$  behaves like  $\mu X.F(X)$  up to the depth  $n$  of recursive calls. Then the following proposition, proved in [77], describes the denotational semantics of recursive quantum programs.

**PROPOSITION 6.1.**  $\llbracket \mu X.F(X) \rrbracket = \sqcup_{n=0}^\infty \llbracket F^{(n)} \rrbracket$ , where  $\sqcup$  stands for the least upper bound in the CPO  $(\mathcal{QO}(\mathcal{H}_{QV}), \sqsubseteq)$  of quantum operations on  $\mathcal{H}_{QV}$ , as in Proposition 3.1(6).

A fixed point characterisation of recursive quantum programs follows immediately:

**PROPOSITION 6.2 (FIXED POINT).**  $\mu X.F(X)$  is the least solution of equation  $X = F(X)$ ; that is,

- (1)  $\mu X.F(X) \equiv F(\mu X.F(X))$ , and
- (2) For any quantum program  $P$ ,  $P \equiv F(P) \Rightarrow \mu X.F(X) \sqsubseteq P$ , where  $Q \sqsubseteq P$  means that  $\llbracket Q \rrbracket(\rho) \sqsubseteq \llbracket P \rrbracket(\rho)$  for any density operator  $\rho$ .

Intuitively, the above proposition indicates that a recursive call is equivalent to the whole program. The two laws in this proposition for recursive quantum programs look the same as laws (Re-2) and (Re-3) in Figure 2 for classical recursive programs. Sharing a similar spirit of using the Kleene fixed-point theorem in both the classical and quantum cases, the proofs of these laws are nevertheless quite different, particularly in the Coq proof assistant, as will be described in Section 10, because we need to handle limits of quantum operations (i.e., super-operators) on  $\mathcal{H}_{QV}$  in the quantum case.

## 6.1 Laws of Loops

As in classical programming, the quantum **while**-loop introduced in Definition 3.2 is a special case of the notion of recursion defined above. Following [32], we introduce a convenient notation for quantum **while**-loop:

$$M[\bar{q}] * P \triangleq \text{while } M[\bar{q}] = 1 \text{ do } P \text{ od.}$$

If the name of measured quantum variables  $\bar{q}$  is not essential or can be known from the context, we often write  $M * P$  for  $M[\bar{q}] * P$ . Then it is easy to see that

$$M * P \equiv \mu X. (M \triangleright P; X), \quad (17)$$

if  $P$  does not contain any further recursive call. Obviously, Eq. (17) is a quantum generalisation of law (Re-1) in Figure 2.

As shown in Example 4.1, quantum random walks (without absorbing boundaries) can be modelled as a fixed number of iterations of a unitary operator  $W$ . Their variants with absorbing boundaries can be properly described as a quantum loop:

EXAMPLE 6.1 (QUANTUM RANDOM WALK WITH ABSORBING BOUNDARY).

$$M[p] * (T_L[p] \leftarrow H[d] \rightarrow T_R[p])$$

where we adopt the notations from Example 4.1, and  $M$  represents a measurement that detects whether the particle has reached the left or right boundary. More precisely, if the left and right boundaries are set at position  $n_l$  and  $n_r$ , respectively, then the measurement is modelled by  $M = \{M_0, M_1\}$ , where  $M_0 = |n_l\rangle\langle n_l| + |n_r\rangle\langle n_r|$  and  $M_1 = I_p - M_0$ , and  $I_p$  is the identity operator on the position space  $\mathcal{H}_p$ .

The following proposition presents several useful laws for loops. It generalises the laws established in [33] for classical loops to quantum loops.

PROPOSITION 6.3 (LAWS FOR LOOP).

(1) (Loop as a Fixpoint)  $M * P \equiv M \triangleright (P; (M * P))$ .

(2) (Loop Unfolding)

$$\frac{M \propto N}{N \triangleright (M * P) \equiv N \triangleright (P; (M * P))}.$$

(3) (Loop Elimination)

$$\frac{M^\perp \propto N}{N \triangleright (M * P) \equiv [N]}.$$

(4) (Postcondition)

$$\frac{N \gg M}{M * P \equiv (M * P); [N]}.$$

Proposition 6.3(2) indicates that if a measurement  $N$  that is logically stronger than  $M$  is performed initially, then quantum loop  $M * P$  executes the loop body  $P$ , and then proceeds like  $M * P$  itself, and Proposition 6.3(3) says that if a measurement  $N$  that is logically stronger than  $M^\perp$  is performed initially, then loop  $M * P$  terminates immediately. Proposition 6.3(4) shows that at its termination,

loop  $M * P$  establishes a postcondition  $N$  which logically contradicts its guard that the outcome of measurement  $M$  is 1.

Recall the notions of logical weakness and contradiction defined in Definition 5.4. Proposition 6.3 shows that they are also useful for simplifying loops.

## 6.2 Tail Recursion

Tail recursion is a scheme of recursion in which the recursive call is the last statement to be executed by the program. It is particularly useful in practice because it is often easy to implement and optimise in compilation. It is well-known in classical programming that tail recursion can always be realised as a loop. This result extends to quantum programming as shown in the following:

**THEOREM 6.1 (TAIL RECURSION).** *For any quantum programs  $P, Q$  (without recursive calls), it holds that  $\mu X.(Q \triangleleft M \triangleright (P; X)) \equiv (M * P); Q$ .*

**PROOF.** We write  $F(X) \triangleq Q \triangleleft M \triangleright (P; X)$ . Let measurement  $M = \{M_0, M_1\}$ . Define quantum operations  $\mathcal{E}_i$  by  $\mathcal{E}_i(\rho) = M_i \rho M_i^\dagger$  for all density operators  $\rho$  ( $i = 0, 1$ ). Then we assert

$$\llbracket F^{(n)} \rrbracket(\rho) = \llbracket Q \rrbracket \left( \sum_{k=0}^{n-1} (\mathcal{E}_0 \circ (\llbracket P \rrbracket \circ \mathcal{E}_1)^k)(\rho) \right), \quad (18)$$

for  $n \geq 0$ . Indeed, (18) can be proved by induction on  $n$ . The case of  $n = 0$  is trivial. Now assume (18) is true for  $n$ . Then, for the case of  $n + 1$ , it holds that

$$\begin{aligned} \llbracket F^{(n+1)} \rrbracket(\rho) &= \llbracket Q \rrbracket(\mathcal{E}_0(\rho)) + \llbracket P; F^{(n)} \rrbracket(\mathcal{E}_1(\rho)) \\ &= \llbracket Q \rrbracket(\mathcal{E}_0(\rho)) + \llbracket F^{(n)} \rrbracket((\llbracket P \rrbracket \circ \mathcal{E}_1)(\rho)) \\ &= \llbracket Q \rrbracket(\mathcal{E}_0(\rho)) + \llbracket Q \rrbracket \left( \sum_{k=0}^{n-1} (\mathcal{E}_0 \circ (\llbracket P \rrbracket \circ \mathcal{E}_1)^k)((\llbracket P \rrbracket \circ \mathcal{E}_1)(\rho)) \right) \\ &= \llbracket Q \rrbracket \left( \mathcal{E}_0(\rho) + \sum_{k=0}^{n-1} (\mathcal{E}_0 \circ (\llbracket P \rrbracket \circ \mathcal{E}_1)^{k+1})(\rho) \right) \\ &= \llbracket Q \rrbracket \left( \sum_{k=0}^n (\mathcal{E}_0 \circ (\llbracket P \rrbracket \circ \mathcal{E}_1)^k)(\rho) \right). \end{aligned}$$

Here, the third equality comes from the induction hypothesis on  $n$ . Thus, Eq. (18) holds for  $n + 1$ , and we complete the proof of Eq. (18). Consequently, using Propositions 3.1(6) and 6.1 we obtain

$$\begin{aligned} \llbracket \mu X.(Q \triangleleft M \triangleright (P; X)) \rrbracket(\rho) &= \bigsqcup_{n=0}^{\infty} \llbracket Q \rrbracket \left( \sum_{k=0}^n (\mathcal{E}_0 \circ (\llbracket P \rrbracket \circ \mathcal{E}_1)^k)(\rho) \right) \\ &= \llbracket Q \rrbracket \left( \bigsqcup_{n=0}^{\infty} \sum_{k=0}^n (\mathcal{E}_0 \circ (\llbracket P \rrbracket \circ \mathcal{E}_1)^k)(\rho) \right) \\ &= \llbracket Q \rrbracket \left( \sum_{k=0}^{\infty} (\mathcal{E}_0 \circ (\llbracket P \rrbracket \circ \mathcal{E}_1)^k)(\rho) \right) \\ &= \llbracket (M * P); Q \rrbracket(\rho). \end{aligned}$$

Here, the first and fourth equalities come from Propositions 6.1 and 3.1, respectively, and the second from the linearity and continuity of  $\llbracket Q \rrbracket$ , which were proved in [77].  $\square$

## 7 Laws for Nondeterminism

All quantum programs considered in the previous sections are deterministic. The program construct of nondeterministic choice was introduced by Dijkstra [21, 22] to support the notion of refinement in program development methodology, in which an implemented program is required to be more deterministic than its specification. Intuitively, nondeterministic choice  $P \sqcup Q$  is executed by executing either  $P$  or  $Q$ , with the choice between them being arbitrary. To facilitate refinement technique in quantum programming, nondeterministic choice  $P \sqcup Q$  was generalised in [87] and more recently in [24] to quantum programs  $P, Q$ . In this section, we derive some algebraic laws for nondeterministic quantum programs.

The syntax of nondeterministic quantum programs is given by adding nondeterministic choice  $P \sqcup Q$  into Eq. (10). Recall from the previous sections that the denotational semantics of a deterministic quantum program  $P$  (without nondeterministic choice) is a quantum operation (i.e., trace-nonincreasing super-operators)  $\llbracket P \rrbracket : \mathcal{D}(\mathcal{H}_{QV}) \rightarrow \mathcal{D}(\mathcal{H}_{QV})$ . To accommodate nondeterminism, however, the denotational semantics of a quantum program  $P$  that may contain nondeterministic choice should be defined as a subset  $\llbracket P \rrbracket \subseteq \mathcal{D}(\mathcal{H}_{QV}) \rightarrow \mathcal{D}(\mathcal{H}_{QV})$  of quantum operations. Intuitively, for each input state  $\rho$ ,  $\llbracket P \rrbracket(\rho) = \{\mathcal{E}(\rho) \mid \mathcal{E} \in \llbracket P \rrbracket\}$  is the set of possible output states of  $P$ . As shown in [24], the structural representation of the denotational semantics of deterministic quantum programs (see Proposition 3.1) can be easily lifted to this case:

- (1)  $\llbracket \text{skip} \rrbracket = \{I\}$  and  $\llbracket \text{abort} \rrbracket = \{0\}$ , where  $I$  and  $0$  are the identity and zero operators, respectively, on  $\mathcal{H}_{QV}$ .
- (2)  $\llbracket q := |\psi\rangle \rrbracket = \{\mathcal{E}_{|\psi}\}$ , where quantum operation  $\mathcal{E}_{|\psi}$  is defined by  $\mathcal{E}_{|\psi}(\rho) = \sum_n |\psi\rangle\langle n|\rho|n\rangle\langle\psi|$  for any  $\rho$ , and  $\{|n\rangle\}$  is a given orthonormal basis of  $\mathcal{H}_{QV}$ .
- (3)  $\llbracket U[\bar{q}] \rrbracket = \{\mathcal{E}_U\}$ , where quantum operation  $\mathcal{E}_U$  is defined by  $\mathcal{E}_U(\rho) = U\rho U^\dagger$  for any  $\rho$ .
- (4)  $\llbracket P; Q \rrbracket = \{\mathcal{F} \circ \mathcal{E} \mid \mathcal{E} \in \llbracket P \rrbracket \text{ and } \mathcal{F} \in \llbracket Q \rrbracket\}$ .
- (5)  $\llbracket P \triangleleft M \triangleright Q \rrbracket = \{\mathcal{E} \circ \mathcal{E}_0 + \mathcal{F} \circ \mathcal{E}_1 \mid \mathcal{E} \in \llbracket P \rrbracket \text{ and } \mathcal{F} \in \llbracket Q \rrbracket\}$ , where  $M = \{M_0, M_1\}$ , and quantum operations  $\mathcal{E}_0, \mathcal{E}_1$  are defined by  $\mathcal{E}_i(\rho) = M_i\rho M_i^\dagger$  for  $i = 0, 1$ .
- (6)  $\llbracket M * P \rrbracket = \{\sum_{i=0}^{\infty} \mathcal{E}_0 \circ (\mathcal{F}_i \circ \mathcal{E}_1) \circ \dots \circ (\mathcal{F}_i \circ \mathcal{E}_1) \mid \mathcal{F}_1, \dots, \mathcal{F}_i, \dots \in \llbracket P \rrbracket\}$ , where  $\mathcal{E}_0, \mathcal{E}_1$  are the same as in (5).

The denotational semantics of nondeterministic choice  $P \sqcup Q$  of quantum programs is then defined in [24] as

$$\llbracket P \sqcup Q \rrbracket = \llbracket P \rrbracket \cup \llbracket Q \rrbracket. \quad (19)$$

Now we are ready to present our laws for nondeterministic quantum programs. Some of them were already applied in the example of quantum error correction code transformation in Subsection 1.1. We first point out that all laws presented in the previous sections, except those for recursion, can be easily generalised to nondeterministic quantum programs; but some of the laws need certain further assumptions on nondeterminism. More precisely, we need to assume that the following programs do not contain non-deterministic choice  $\sqcup$ :  $P$  in Proposition 5.2(2),  $P, Q$  in Proposition 5.3(1),  $R$  in Proposition 5.3(2–5),  $P$  in Proposition 5.4(5–6) and  $C$  in Proposition 5.4(7). The next proposition introduces some new laws about non-deterministic choice:

**PROPOSITION 7.1 (LAWS FOR NONDETERMINISM).**

- (1) (*Commutativity*)  $P \sqcup Q \equiv Q \sqcup P$ .
- (2) (*Associativity*)  $P \sqcup (Q \sqcup R) \equiv (P \sqcup Q) \sqcup R$ .
- (3) (*Idempotence*)  $P \sqcup P \equiv P$ .

(4) (*Distributivity I*) *If-conditional distributes over nondeterministic choice:*

$$(P \sqcup Q) \triangleleft M \triangleright R = (P \triangleleft M \triangleright R) \sqcup (Q \triangleleft M \triangleright R),$$

$$R \triangleleft M \triangleright (P \sqcup Q) = (R \triangleleft M \triangleright P) \sqcup (R \triangleleft M \triangleright Q).$$

(5) (*Distributivity II*) *Sequential composition distributes over nondeterministic choice:*

$$P; (Q \sqcup R) \equiv (P; Q) \sqcup (P; R), \quad (Q \sqcup R); P \equiv (Q; P) \sqcup (R; P).$$

These laws in the above proposition were already used in Subsection 1.1 for verification of the correctness of a simple quantum error correction code. Clearly, Clauses (1), (2), (4) and (5) in this proposition are quantum generalisations of classical laws (Nd-1), (Nd-2), (Nd-3,4) and (Nd-6,7) in Figure 2, respectively. However, it is worth pointing out that the quantum counterpart of law (Nd-5) for classical nondeterministic choice does not hold. For example, if  $P \equiv Q \equiv \mathbf{abort}$ ,  $M = \{M_0, M_1\}$  and  $R$  is a deterministic quantum program (without choice  $\sqcup$ , then  $\llbracket (P \triangleleft M \triangleright Q) \sqcup R \rrbracket = \{0, \llbracket R \rrbracket\} \neq \{0, \llbracket R \rrbracket \circ \mathcal{E}_0, \llbracket R \rrbracket \circ \mathcal{E}_1, \llbracket R \rrbracket \circ (\mathcal{E}_0 + \mathcal{E}_1)\} = \llbracket (P \sqcup R) \triangleleft M \triangleright (Q \sqcup R) \rrbracket$ , where for  $i = 0, 1$ , quantum operations  $\mathcal{E}_i$  is defined by  $\mathcal{E}_i(\rho) = M_i \rho M_i^\dagger$  for any  $\rho$ .

As in classical programming, we are able to prove a series of laws for the refinement of quantum programs.

## 8 Laws for Refinement

Refinement is one of the central ideas in program development methodology [46]. As mentioned in Section 7, it has recently been introduced into quantum programming [25, 51]. Formally, let  $P$  and  $Q$  be two (nondeterministic) quantum programs. Then the refinement relation between them is defined as follows:

$$P \sqsubseteq Q \text{ if and only if } \llbracket P \rrbracket \subseteq \overline{\text{Con}(\llbracket Q \rrbracket)},$$

where  $\text{Con}(\cdot)$  stands for the convex hull, and  $\overline{\phantom{x}}$  denotes topological closure. Intuitively,  $P \sqsubseteq Q$  means that  $P$  is a more deterministic program than  $Q$ . The statement that  $P$  is a refinement of  $Q$ , written  $P \sqsubseteq Q$ , should be understood in the following way [32]: whenever  $Q$  *reliably* serves some useful purpose, then  $Q$  can be replaced by  $P$ , in the certainty that it will serve the same purpose, but not vice versa; that is, there may be some purposes for which  $P$  is adequate, but  $Q$  cannot be relied on. Therefore,  $P \sqsubseteq Q$  conveys the idea that program  $P$  is better than, or at least as good as, program  $Q$ .

Obviously,  $\sqsubseteq$  is a partial order,  $P \sqsubseteq P \sqcup Q$  and  $Q \sqsubseteq P \sqcup Q$ . Furthermore, we can prove that the refinement relation is preserved by all quantum program constructs considered in this paper except loops and recursion:

PROPOSITION 8.1. *If  $P \sqsubseteq Q$ , then:*

- (1)  $P; R \sqsubseteq Q; R$  and  $R; P \sqsubseteq R; Q$ .
- (2)  $P \triangleleft M \triangleright R \sqsubseteq Q \triangleleft M \triangleright R$  and  $R \triangleleft M \triangleright P \sqsubseteq R \triangleleft M \triangleright Q$ .
- (3)  $P \sqcup R \sqsubseteq Q \sqcup R$  and  $R \sqcup P \sqsubseteq R \sqcup Q$ .

A discussion of preservation of refinement by loop is given in Appendix C.

Now, let us further consider two different refinements of nondeterministic choice. According to different strategies in the resolution of nondeterminism, we can define:

(1) *Probabilistic implementation:*

$$\llbracket P \sqcup_p Q \rrbracket = \{p\mathcal{E} + (1-p)\mathcal{F} \mid \mathcal{E} \in \llbracket P \rrbracket, \mathcal{F} \in \llbracket Q \rrbracket \text{ and } p \in [0, 1]\}.$$

(2) *Measurement-based implementation:* Suppose we have an extra qubit  $q$  in a pure state  $|\phi\rangle$ . It is possible to refine  $P \sqcup Q$  by: (i) measuring  $q$  according to an arbitrary measurement  $M$ ; (ii) for each branch, recovering the  $q$  by initialising  $q := |\phi\rangle$  and then executing  $P$  or  $Q$ , respectively.

In  $P \sqcup_p Q$ ,  $P$  and  $Q$  are selected to execute according to a probability distribution  $p$ ,  $1 - p$ . In contrast, in the measurement-based implementation, the computational outcome can be seen as obtained by a post-selection according to a quantum measurement  $M = \{M_0, M_1\}$ . The following proposition clarifies the refinement relation between nondeterministic choice and its probabilistic and measurement-based resolutions:

- PROPOSITION 8.2. (1)  $P \sqcup_p Q \sqsubseteq P \sqcup Q$ .  
 (2) If  $P \sqsubseteq Q$  then  $P \sqcup_p R \sqsubseteq Q \sqcup_p R$  and  $R \sqcup_p P \sqsubseteq R \sqcup_p Q$ .  
 (3)  $[M]; P \sqsubseteq P \triangleleft M \triangleright P$ .  
 (4)  $q := |\phi\rangle; ((q := |\phi\rangle); P) \triangleleft M[q] \triangleright (q := |\phi\rangle); Q) \sqsubseteq q := |\phi\rangle; (P \sqcup Q)$ .

## 9 An Application: Formal Derivation of the Principle of Deferred Measurements

A basic principle of great utility in quantum computing is:

- **Principle of deferred measurements** [48]: *Measurements can always be moved from an intermediate stage of a quantum circuit to the end of the circuit; if the measurement results are used at any stage of the circuit, then the classically controlled operations can be replaced by conditional quantum operations.*

Quantum circuits with measurements in the middle, whose results are used to control later computations, are essentially finite programs, as we called them in Section 5.5 of this paper. They are also often called dynamic quantum circuits [18].

The above principle can be formally formulated in a programming language with both **if**-statements and quantum **if**-statements:

THEOREM 9.1. *If plenty of fresh auxiliary variables are assumed, then for any finite quantum program (i.e., a program that does not contain iteration or recursion)  $P \in \mathbf{QProg}$ , there exists a quantum circuit  $C \in \mathbf{QC}$  such that:*

$$\bar{q}_a := |0\rangle; P \equiv \bar{q}_a := |0\rangle; C; \mathbf{if} [\bar{q}_a] (\square m \rightarrow P_m) \mathbf{fi}; \bar{q}_a := |0\rangle. \quad (20)$$

where  $P_m \in \{\mathbf{skip}, \mathbf{abort}\}$ ,  $\bar{q}_a$  is the auxiliary register that initialised and reset to  $|0\rangle$  through the program. In particular, quantum circuit  $C$  can be recursively generated using quantum **if**-statements according to the structure of  $P$ .

Informally, the above theorem suggests that  $P \equiv C; \mathbf{if} [\bar{q}_a] (\square m \rightarrow P_m) \mathbf{fi}$  if we ignore auxiliary variables, that is, a finite quantum program  $P$  can be equivalently modelled as a circuit followed by a computational measurement. Here, since classical variables are not included in our syntax, we implicitly discard any intermediate measurement outcome; thus, the irreversibility of  $\bar{q}_a := |0\rangle$  in the RHS of (20) might be interpreted as the irreversibility of intermediate measurement in  $P$ . On the other hand, if we are allowed to gather measurement outcomes, the measurement outcome of  $\bar{q}_a$  is exactly the collection of all measurement outcomes in  $P$ .

As an application of the laws we developed in the previous sections, we show how Theorem 9.1 can be formally derived using them.

LEMMA 9.1 (IMPLEMENTATION OF **if**-STATEMENT WITH AUXILIARY VARIABLE, C.F. [48]). *For any if statement  $\mathbf{if} M[\bar{q}] (\square m \rightarrow P_m) \mathbf{fi}$  where  $M = \{M_i\}_{i \in I}$ , let  $q_a$  be an arbitrary fresh auxiliary variable with Hilbert space of dimension  $|I|$  and initialised in basis state  $|0\rangle$ . Then there exists a unitary operator  $U_M$ , which is regarded as necessary information for implementing the measurement, such that:*

$$q_a := |0\rangle; \mathbf{if} M[\bar{q}] (\square m \rightarrow P_m) \mathbf{fi} \equiv q_a := |0\rangle; U_M[\bar{q}, q_a]; \mathbf{if} [q_a] (\square m \rightarrow P_m) \mathbf{fi}; q_a := |0\rangle.$$

PROOF OF THEOREM 9.1. We proceed by induction on the structure of  $P$ . Here, we consider only the case  $P = \mathbf{if} M[\bar{q}] (\square i \rightarrow P_i) \mathbf{fi}$ ; the proofs for the other cases are similar and postponed to

Appendix B. By the induction hypothesis, we assume that each  $P_i$  has the following normal form such that  $qv(C_i) \subseteq qv(P_i) \cup \bar{r}_i$  and

$$\bar{r}_i := |0\rangle; P_i \equiv \bar{r}_i := |0\rangle; C_i; \text{if } [\bar{r}_i] (\Box j \rightarrow P_{ij}) \text{ fi}; \bar{r}_i := |0\rangle. \quad (21)$$

Let  $a$  be a fresh variable that is not in all  $\bar{r}_i$ , and let  $\bar{q}_a \triangleq a \cup \bigcup_i \bar{r}_i$ . Then we claim:

$$\begin{aligned} \bar{q}_a := |0\rangle; \text{if } M[\bar{q}] (\Box i \rightarrow P_i) \text{ fi} &\equiv \bar{q}_a := |0\rangle; (U_M[\bar{q}, a]; \text{qif } [a] (\Box i) \rightarrow C_i \text{ fiq}); \\ &\text{if } [\bar{q}_a] (\Box k \rightarrow P_{k\downarrow_a k\downarrow_{\bar{r}_i}}) \text{ fi}; \bar{q}_a := |0\rangle, \end{aligned}$$

for some unitary  $U_M$ . For simplicity, we set  $\bar{r} \triangleq \bigcup_i \bar{r}_i$  and  $\bar{r}'_i \triangleq \bar{r} \setminus \bar{r}_i$ . Then

$$\begin{aligned} &\bar{q}_a := |0\rangle; \text{if } M[\bar{q}] (\Box i \rightarrow P_i) \text{ fi} \\ &\equiv \bar{q}_a := |0\rangle; \bar{r} := |0\rangle; a := |0\rangle; \text{if } M[\bar{q}] (\Box i \rightarrow P_i) \text{ fi} && \text{(By Prop. 5.1(1))} \\ &\equiv \bar{q}_a := |0\rangle; \bar{r} := |0\rangle; a := |0\rangle; U_M[\bar{q}, a]; \text{if } [a] (\Box i \rightarrow P_i) \text{ fi}; a := |0\rangle && \text{(By Lemma 9.1)} \\ &\equiv \bar{q}_a := |0\rangle; U_M[\bar{q}, a]; \text{if } [a] (\Box i \rightarrow \bar{r} := |0\rangle; P_i) \text{ fi}; a := |0\rangle && \text{(By Props. 5.1(1), 5.4(2, 6))} \\ &\equiv \bar{q}_a := |0\rangle; U_M[\bar{q}, a]; \text{if } [a] (\Box i \rightarrow \bar{r} := |0\rangle; \bar{r}_i := |0\rangle; \bar{r}'_i := |0\rangle; P_i) \text{ fi}; a := |0\rangle && \text{(By Prop. 5.1(1))} \\ &\equiv \bar{q}_a := |0\rangle; U_M[\bar{q}, a]; \text{if } [a] (\Box i \rightarrow \bar{r} := |0\rangle; \bar{r}_i := |0\rangle; P_i; \bar{r}'_i := |0\rangle) \text{ fi}; a := |0\rangle && \text{(By Prop. 5.4(2))} \\ &\equiv \bar{q}_a := |0\rangle; U_M[\bar{q}, a]; \text{if } [a] (\Box i \rightarrow \bar{r} := |0\rangle; \bar{r}_i := |0\rangle); \\ &\quad C_i; \text{if } [\bar{r}_i] (\Box j \rightarrow P_{ij}) \text{ fi}; \bar{r}_i := |0\rangle; \bar{r}'_i := |0\rangle) \text{ fi}; a := |0\rangle && \text{(By Ind. hyp. Eq. (21))} \\ &\equiv \bar{q}_a := |0\rangle; U_M[\bar{q}, a]; \text{if } [a] (\Box i \rightarrow \bar{r} := |0\rangle; C_i; \\ &\quad \text{if } [\bar{r}_i] (\Box j \rightarrow P_{ij}) \text{ fi}; \bar{r} := |0\rangle) \text{ fi}; a := |0\rangle && \text{(By Prop. 5.1(1))} \\ &\equiv \bar{q}_a := |0\rangle; U_M[\bar{q}, a]; \bar{r} := |0\rangle; \text{if } [a] (\Box i \rightarrow \\ &\quad C_i; \text{if } [\bar{r}] (\Box j \rightarrow P_{ij\downarrow_{\bar{r}_i}}) \text{ fi}; \bar{r} := |0\rangle) \text{ fi}; a := |0\rangle && \text{(By Props. 5.1(5), 5.4(2, 6))} \\ &\equiv \bar{q}_a := |0\rangle; U_M[\bar{q}, a]; \text{if } [a] (\Box i \rightarrow C_i; \text{skip}) \text{ fi}; \\ &\quad \text{if } [a] (\Box i \rightarrow \text{if } [\bar{r}] (\Box j \rightarrow P_{ij\downarrow_{\bar{r}_i}}) \text{ fi}; \bar{r} := |0\rangle) \text{ fi}; a := |0\rangle && \text{(By Props. 5.1(1), 5.4(1, 2, 4))} \\ &\equiv \bar{q}_a := |0\rangle; U_M[\bar{q}, a]; \text{qif } [a] (\Box i) \rightarrow C_i \text{ fiq}; \text{if } [a] (\Box i \rightarrow \text{skip}) \text{ fi}; \\ &\quad \text{if } [a] (\Box i \rightarrow \text{if } [\bar{r}] (\Box j \rightarrow P_{ij\downarrow_{\bar{r}_i}}) \text{ fi}) \text{ fi}; \bar{r} := |0\rangle; a := |0\rangle && \text{(By Props. 5.4(5), 5.5)} \\ &\equiv \bar{q}_a := |0\rangle; U_M[\bar{q}, a]; \text{qif } [a] (\Box i) \rightarrow C_i \text{ fiq}; \\ &\quad \text{if } [a] (\Box i \rightarrow \text{if } [\bar{r}] (\Box j \rightarrow P_{ij\downarrow_{\bar{r}_i}}) \text{ fi}) \text{ fi}; \bar{q}_a := |0\rangle && \text{(By Props. 5.1(1), 5.4(1, 4))} \\ &\equiv \bar{q}_a := |0\rangle; (U_M[\bar{q}, a]; \text{qif } [a] (\Box i) \rightarrow C_i \text{ fiq}); \\ &\quad \text{if } [\bar{q}_a] (\Box k \rightarrow P_{k\downarrow_a k\downarrow_{\bar{r}_i}}) \text{ fi}; \bar{q}_a := |0\rangle. && \text{(By Prop. 5.3(1))} \end{aligned}$$

It is particularly interesting to note that the law in Proposition 5.5 is used in the 10th equivalence of the above equation.  $\square$

## 10 Formalisation in the Coq Proof Assistant

We formalised the theory in which our laws are established and verified all of the laws presented in previous sections in the Coq proof assistant [67]. We achieved this by employing CoqQ [85], a general-purpose framework for quantum program verification built upon the state-of-the-art

mathematical libraries MathComp [44] and MathComp-Analysis [68]. The development can be found at <https://github.com/coq-quantum/CoqQ/tree/main/src/example/qlaws>.

There are several mechanised approaches to quantum circuits and quantum programs; see [16, 42] for a comprehensive review. Among them, targeting the verification of high-level quantum programs, CoqQ offers an extensive library on abstract linear algebra and analysis, e.g., existing theorems on convergence and fixed points are crucial for formalising our semantics of while and recursive programs. Recent development [25] enhances CoqQ by a formalised mechanism for quantum registers with automatic type checking, greatly facilitating the mature use of quantum variables. Other tools, such as QWIRE [50, 55] and SQIR [30, 31], have their respective focuses, as we will discuss in Section 11.

### 10.1 Formalizing the theory

The existing qwhile language in CoqQ is not sufficient to describe and prove the laws proposed in this paper. The further developments of formalising the theory in this work include:

- We implement the two layers of quantum circuits and purely quantum programs in our three-layer quantum programming framework (see Figure 1) from the bottom up. In particular, the purely quantum program layer is divided into quantum **while**-programs, quantum programs with recursion, and quantum programs with nondeterminism, so that we can prove the laws in different scenarios with slightly adjusted premises. As far as we know, this is the first time formalising the semantics of recursive quantum programs, in particular *with mutual recursive calls*.
- We provide automatic checks on the sets of variables, such as the well-formedness and disjointness of the programs, to facilitate the use of the laws. The disjointness condition asserts that a quantum register should consist of a set of distinct variables, which is required due to the no-cloning theorem. Disjointness is further employed in (1) the well-formedness of a quantum **if**-statement and (2) side-conditions of multiple laws. To this end, we extend the register mechanism in [25], making Coq automatically infer the disjointness conditions.
- *Convex hull* is employed as the basis of developing the refinement laws of nondeterministic quantum programs. As a foundation for proving these laws, we implemented the theory of convex hulls with related basic properties of addition + (i.e.,  $A+B = \{a+b \mid a \in A \ \& \ b \in B\}$ ) and composition  $\circ$  (i.e.,  $A \circ B = \{a \circ b \mid a \in A \ \& \ b \in B\}$ ), such as  $Conv(A+B) = Conv(A)+Conv(B)$  and  $Conv(A \circ B) = Conv(Conv(A) \circ Conv(B))$ .

### 10.2 Verification of the laws

Based on the developments described in the above subsection, we proved all of the laws of quantum programming introduced in this paper in Coq. More precisely:

- The basic laws for quantum circuits in Section 4 can be easily proved. More interestingly, we prove the existence of normal forms for quantum circuits in Coq. In detail, we first give a specific construction of normal form, which is defined inductively on the program syntax, and then prove that the construction is semantically equivalent to the original program.
- The laws for purely quantum programs are proved sound with respect to the semantics formalised in Subsection 10.1. Some laws are proved in a more general setting; for example, the rules for recursive programs hold even if more than one recursive functions are involved.

### 10.3 Applications

Furthermore, the formalisation and proof of our laws of quantum programming, described in the previous two subsections, support formal verification of more complicated applications in Coq. As

a simple example showing this prospect, we provide the proof of Example 1.1 in Coq, which can be found in the development.

## 11 Related Work

### 11.1 Quantum Programming Framework

We described a three-layer framework of quantum programming in Section 3 so that the laws of quantum programming developed in this paper can be grouped properly. As pointed out in the Introduction, all ingredients of this framework are not new. At the first layer is a quantum circuit description language. A basic difference between it and other quantum circuit languages like QWIRE [50] is that the construct of quantum **if**-statement is included in the former (see Clause (4) in Definition 3.1) but not in the latter. But quantum **if**-statement has been extensively studied in quantum programming literature [2, 9, 73, 78, 82, 83]. At the second layer of purely quantum programs, a quantum extension of classical **while**-language is employed, gradually expanded by adding recursion and nondeterministic choice. The quantum **while**-language is standard, as exposed in the book [77]. The recursion considered in this paper is defined with classical control flow rather than quantum control flow. This kind of recursion was carefully studied in Section 3.4 of [77] and [76]. Nondeterministic quantum programs were first introduced in [87], and program logic for them was recently developed in [24]. The third layer of our framework is quantum programs embedded in a classical programming language, which is a common practice in quantum programming. Indeed, all of these program constructs have already been widely adopted in mainstream quantum programming languages and platforms [1, 19, 26–29, 34, 45, 49, 61, 64, 65]. Therefore, one can expect that the laws established in this paper can be applied to the compilation, transformation, and optimisation of quantum programs in these languages. However, determining how to practically apply these laws in each case remains an important direction for future research.

### 11.2 Algebraic reasoning about quantum programs

Our laws of quantum programming, established within the framework discussed above, are essentially quantum generalisations of Hoare et al.’s algebraic laws of classical programming [32, 33, 56].

Several other approaches to algebraic reasoning about quantum programs have been proposed in the literature. The ZX-calculus is a graphical language for reasoning about linear maps between qubits [17, 23, 36, 37]. A rich theory of equational reasoning and algebraic laws has been developed around it. It has found successful applications across a wide range of areas in quantum computation and quantum information, particularly in quantum circuit verification, compilation, synthesis, and optimisation. The relationship between the ZX-calculus and the laws presented in this paper remains to be fully understood; for example, it is an open question how our laws—especially those concerning loops and recursion—can be represented within the ZX-calculus framework.

An interesting approach to quantum programming was proposed in [26, 27], where linear algebraic tools are used to provide basic insights into the idealised process of unitary evolution. Clearly, exploring how the laws developed in this paper can be combined with the ideas of [26, 27] is an interesting and worthwhile topic for future research.

An algebraic theory of unitary gates and quantum measurements was proposed in [62], from which an equational theory for quantum programs was extracted. A key difference between [62] and this paper is that the former emphasises high-level theories, whereas the latter aims to derive concrete laws that can be directly applied to quantum program development.

The successful applications of KAT (Kleene Algebras with Tests) [39] in reasoning about classical programs were extended to quantum programs and networks [13, 53]. However, most of the laws of quantum programming derived in this paper were not obtained in [53, 62]. A normal form of

quantum **while**-programs consisting of only a single loop was proved in [53, 81]. However, it is different from our normal form (Theorem 5.1) because the latter is about finite quantum programs without loops. Indeed, they can be combined to transform an arbitrary quantum **while**-program  $P$  to a stronger normal form  $P \equiv \mathbf{while} M[\bar{q}] = 1 \mathbf{do} Q \mathbf{od}$  with a single loop and a flat **if**-statement  $Q$  in the form of (16).

### 11.3 Rewrite Rules used in Compilation and Optimisation of Quantum Programs

In the last few years, several quantum-circuit optimisers have been implemented using rewrite rules, including VOQC [31], Quartz [75], and QUESO [74]. Some rewrite rules were also introduced in Giallar—a verification tool for the Qiskit compiler [66]. The rewrite rules employed in these works are essentially algebraic laws for quantum circuits, but they are quite different from our laws for quantum circuits given in Section 4 because the majority of our laws concern the quantum **if**-statement, which is not defined in the syntax of quantum circuits considered in [31, 66, 74, 75]. Indeed, the laws given in [31, 66, 74, 75] and our laws presented in Section 4 can be used complementarily for the optimisation of quantum circuits. They can be further combined with our laws presented in Section 5 for optimisation of quantum programs.

The normal form approach to compiler design in [33] was already generalised in [86] to the compilation of quantum programming language qGCL (quantum Guarded-Command Language) [58], which extends pGCL (probabilistic GCL) [35] with constructs of initialisation, unitary transformation, and measurement. The compiler described in [86] mainly deals with transformations at the classical-quantum hybrid layer, with the layers of quantum circuits and purely quantum programs untouched. The normal form employed in [86] is inherited directly from that in [33, 35] for GCL via pGCL, and thus is fundamentally different from the normal forms presented in this paper (Theorems 4.1 and 5.1). Our normal forms can help to further compile the compiled programs in [86] into a lower level.

### 11.4 Mechanised Approach for Quantum Programming

There is an active line of work that develops formal methods and, more specifically, machine-checkable verification of quantum programs (see recent reviews [16, 42]). Here, we only briefly discuss those that are mostly related to our work:

**Coq-based formalisation.** QWIRE [50, 55] formalises the circuit-like programming language and provides a denotational semantics of quantum circuits in terms of density matrices. Its extension ReQWIRE [54] is a verified compiler from classical functions to quantum circuits. SQIR [31] is a low-level language for intermediate representation. Its semantics is based on a density matrix representation of quantum states, and has been successfully applied to developing a verified optimiser of quantum circuits [31], as well as semantics-based verification of quantum programs [30, 52]. VyZX [40, 41, 59] is a certified (in Coq) formalisation of the ZX-calculus [17, 36], which is proposed for reasoning about quantum circuits in a flexible graphical structure.

**Isabelle/HOL-based formalisation.** QHLPover [43] provides the first formalisation of quantum Hoare logic [78] and is used to verify examples such as Grover’s algorithm. Isabelle Marries Dirac [11, 12] is an ongoing effort to formalise quantum information theory and to provide verified quantum algorithms; the formalisation is based on a matrix representation for quantum circuits. qrhl-tool [69, 70] is a verification tool for (post-)quantum cryptography based on quantum relational Hoare logic.

**Other tools.** Building upon Why3 [10], QBRICKS [15] provides a highly automated verification framework for circuit-building quantum programs based on path-sum representations of quantum states [3]. EasyPQC [6] is an extension of EasyCrypt [7] that aims to verify the security of post-quantum cryptography based on post-quantum relational Hoare logic. The proof assistant

Quantomatic [38] has been successfully used for diagrammatic reasoning in quantum information. A particularly interesting problem for future research is how the laws developed in this paper can be implemented in Quantomatic so that they can be used more conveniently for reasoning about quantum programs (with **if**-statements and **while**-loops).

## 12 Conclusion

A series of basic laws for quantum programming has been developed in this paper, generalising Hoare et al.'s fundamental laws for classical programming. These laws are formally verified in the Coq proof assistant, ensuring that they can be confidently applied in the compilation, transformation, optimisation, analysis, and verification of quantum programs.

For further development and application of this line of research, we propose the following topics:

- **Application in Quantum Compilers:** As briefly discussed earlier, the laws presented in this paper—particularly the two normal forms and the realisation of tail recursion by loops—can be incorporated into optimising compilers for quantum programs. However, it remains unclear whether these laws will be practically useful for quantum circuit and program optimisation. We plan to collaborate with the isQ team [29] to implement these laws in the isQ compiler and evaluate their effectiveness using benchmarks of quantum circuits and programs.
- **Laws for Recursion with Quantum Control Flows:** Quantum if-statement is the basic program construct representing quantum control flows. Recently, some recursion schemes with quantum control flows have been introduced into quantum programming; for example, recursion in linear-algebraic (quantum) lambda-calculus [4, 20, 71], quanta-morphism (a structural form of quantum recursion implementing cycles and folds on lists with quantum control flow) [47], quantum recursion in Fock spaces [77], and recursively defined quantum circuits [79, 80, 84]. A set of algebraic laws for quantum if-statements was established at the quantum circuit layer in Section 4. On the other hand, we proved several laws for loop and recursive quantum programs with classical control flows; in particular, Theorem 6.1 for tail recursion. A natural problem is how to extend these laws to quantum recursive programs with quantum control flows. It seems that this problem cannot be solved simply by adopting the approach used in Section 6.
- **Automated Verified Verifier/Optimiser:** Although we have formalised the proposed laws in Coq, the development currently only supports interactive proofs, which significantly limits its usefulness. Automation is particularly important, especially for verifying or optimising large-scale circuits and programs. Building rewriting systems is one of the common approaches for equivalence checking. Designing heuristics or AI-assisted algorithms to optimise the application of different laws is also a potential direction.

## Acknowledgement

This research was partly supported by the National Key R&D Program of China under Grant No. 2023YFA1009403.

## References

- [1] Ali J Abhari, Arvin Faruque, Mohammad J Dousti, Lukas Svec, Oana Catu, Amlan Chakrabati, Chen-Fu Chiang, Seth Vanderwilt, John Black, and Fred Chong. 2012. *Scaffold: Quantum programming language*. Technical Report. Princeton University.
- [2] T. Altenkirch and J. Grattage. 2005. A functional quantum programming language. In *20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05)*. 249–258. <https://doi.org/10.1109/LICS.2005.1>
- [3] Matthew Amy. 2018. Towards Large-scale Functional Verification of Universal Quantum Circuits. In *Proceedings 15th International Conference on Quantum Physics and Logic, QPL 2018, Halifax, Canada, 3-7th June 2018 (EPTCS, Vol. 287)*, Peter Selinger and Giulio Chiribella (Eds.). 1–21. <https://doi.org/10.4204/EPTCS.287.1>

- [4] Pablo Arrighi, Alejandro Díaz-Caro, and Benoît Valiron. 2017. The vectorial  $\lambda$ -calculus. *Information and Computation* 254 (2017), 105–139. <https://doi.org/10.1016/j.ic.2017.04.001>
- [5] Manuel Barbosa, Gilles Barthe, Xiong Fan, Benjamin Grégoire, Shih-Han Hung, Jonathan Katz, Pierre-Yves Strub, Xiaodi Wu, and Li Zhou. 2021. EasyPQC: Verifying Post-Quantum Cryptography. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (Virtual Event, Republic of Korea) (CCS '21)*. Association for Computing Machinery, New York, NY, USA, 2564–2586. <https://doi.org/10.1145/3460120.3484567>
- [6] Manuel Barbosa, Gilles Barthe, Xiong Fan, Benjamin Grégoire, Shih-Han Hung, Jonathan Katz, Pierre-Yves Strub, Xiaodi Wu, and Li Zhou. 2021. EasyPQC: Verifying Post-Quantum Cryptography. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (Virtual Event, Republic of Korea) (CCS '21)*. Association for Computing Machinery, New York, NY, USA, 2564–2586. <https://doi.org/10.1145/3460120.3484567>
- [7] Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. 2014. *EasyCrypt: A Tutorial*. Springer International Publishing, Cham, 146–166. [https://doi.org/10.1007/978-3-319-10082-1\\_6](https://doi.org/10.1007/978-3-319-10082-1_6)
- [8] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. 2010. Programming Language Techniques for Cryptographic Proofs. In *Interactive Theorem Proving*, Matt Kaufmann and Lawrence C. Paulson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 115–130.
- [9] Benjamin Bichsel, Maximilian Baader, Timon Gehr, and Martin Vechev. 2020. Silq: a high-level quantum language with safe uncomputation and intuitive semantics. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation (London, UK) (PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 286–300. <https://doi.org/10.1145/3385412.3386007>
- [10] François Bobot, Jean-Christophe Filliâtre, Claude Marché, and Andrei Paskevich. 2011. Why3: Shepherd Your Herd of Provers. In *Boogie 2011: First International Workshop on Intermediate Verification Languages*. Wroclaw, Poland, 53–64. <https://inria.hal.science/hal-00790310>
- [11] Anthony Bordg, Hanna Lachnitt, and Yijun He. 2020. Isabelle marries dirac: A library for quantum computation and quantum information. *Archive of Formal Proofs* (2020).
- [12] Anthony Bordg, Hanna Lachnitt, and Yijun He. 2021. Certified Quantum Computation in Isabelle/HOL. *Journal of Automated Reasoning* 65, 5 (01 June 2021), 691–709. <https://doi.org/10.1007/s10817-020-09584-7>
- [13] Anita Buckley, Pavel Chuprikov, Rodrigo Otoni, Robert Soulé, Robert Rand, and Patrick Eugster. 2024. An Algebraic Language for Specifying Quantum Networks. *Proc. ACM Program. Lang.* 8, PLDI, Article 200 (jun 2024), 23 pages. <https://doi.org/10.1145/3656430>
- [14] Lukas Burgholzer and Robert Wille. 2021. Advanced Equivalence Checking for Quantum Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 9 (2021), 1810–1824. <https://doi.org/10.1109/TCAD.2020.3032630>
- [15] Christophe Chareton, Sébastien Bardin, François Bobot, Valentin Perrelle, and Benoît Valiron. 2021. An Automated Deductive Verification Framework for Circuit-building Quantum Programs. In *Programming Languages and Systems*, Nobuko Yoshida (Ed.). Springer International Publishing, Cham, 148–177. [https://doi.org/10.1007/978-3-030-72019-3\\_6](https://doi.org/10.1007/978-3-030-72019-3_6)
- [16] Christophe Chareton, Sébastien Bardin, Dong Ho Lee, Benoît Valiron, Renaud Vilmart, and Zhaowei Xu. 2023. Formal Methods for Quantum Algorithms. In *Handbook of Formal Analysis and Verification in Cryptography*. CRC Press, 319–422. <https://cea.hal.science/cea-04479879>
- [17] Bob Coecke and Ross Duncan. 2011. Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics* 13, 4 (apr 2011), 043016. <https://doi.org/10.1088/1367-2630/13/4/043016>
- [18] A. D. Córcoles, Maika Takita, Ken Inoue, Scott Lekuch, Zlatko K. Mineev, Jerry M. Chow, and Jay M. Gambetta. 2021. Exploiting Dynamic Quantum Circuits in a Quantum Algorithm with Superconducting Qubits. *Phys. Rev. Lett.* 127 (Aug 2021), 100501. Issue 10. <https://doi.org/10.1103/PhysRevLett.127.100501>
- [19] Cirq Developers. 2025. *Cirq*. <https://doi.org/10.5281/zenodo.15191735>
- [20] Alejandro Díaz-Caro, Mauricio Guillermo, Alexandre Miquel, and Benoît Valiron. 2019. Realizability in the Unitary Sphere. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24–27, 2019*. 1–13. <https://doi.org/10.1109/LICS.2019.8785834>
- [21] Edsger W. Dijkstra. 1975. Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM* 18, 8 (aug 1975), 453–457. <https://doi.org/10.1145/360933.360975>
- [22] Edsger Wybe Dijkstra, Edsger Wybe Dijkstra, Edsger Wybe Dijkstra, États-Unis Informaticien, and Edsger Wybe Dijkstra. 1976. *A discipline of programming*. Vol. 613924118. prentice-hall Englewood Cliffs.
- [23] Ross Duncan, Aleks Kissinger, Simon Perdrix, and John van de Wetering. 2020. Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus. *Quantum* 4 (June 2020), 279. <https://doi.org/10.22331/q-2020-06-04-279>
- [24] Yuan Feng and Yingte Xu. 2023. Verification of Nondeterministic Quantum Programs. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3 (ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 789–805. <https://doi.org/10.1145/3582016.3582039>

- [25] Yuan Feng, Li Zhou, and Yingte Xu. 2023. Refinement calculus of quantum programs with projective assertions. arXiv:2311.14215 [cs.LO]
- [26] P. Gawron, J. Klamka, J. Miszczak, and R. Winiarczyk. 2010. Extending scientific computing system with structural quantum programming capabilities. online. *Bulletin of the Polish Academy of Sciences Technical Sciences* 58, No 1 (2010), 77–88. <https://doi.org/10.2478/v10175-010-0008-4>
- [27] Vladimir P. Gerdt and Alexander N. Prokopenya. 2013. Simulation of Quantum Error Correction with Mathematica. In *Computer Algebra in Scientific Computing*, Vladimir P. Gerdt, Wolfram Koepf, Ernst W. Mayr, and Evgenii V. Vorozhtsov (Eds.). Springer International Publishing, Cham, 116–129.
- [28] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoit Valiron. 2013. Quipper: a scalable quantum programming language. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Seattle, Washington, USA) (PLDI '13). Association for Computing Machinery, New York, NY, USA, 333–342. <https://doi.org/10.1145/2491956.2462177>
- [29] Jingzhe Guo, Huazhe Lou, Jintao Yu, Riling Li, Wang Fang, Junyi Liu, Peixun Long, Shenggang Ying, and Mingsheng Ying. 2023. isQ: An Integrated Software Stack for Quantum Programming. *IEEE Transactions on Quantum Engineering* 4 (2023), 1–16. <https://doi.org/10.1109/TQE.2023.3275868>
- [30] Kesha Hietala, Robert Rand, Shih-Han Hung, Liyi Li, and Michael Hicks. 2021. Proving Quantum Programs Correct. In *12th International Conference on Interactive Theorem Proving (ITP 2021) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 193)*, Liron Cohen and Cezary Kaliszyk (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 21:1–21:19. <https://doi.org/10.4230/LIPIcs.ITP.2021.21>
- [31] Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. 2021. A verified optimizer for Quantum circuits. 5, POPL, Article 37 (jan 2021), 29 pages. <https://doi.org/10.1145/3434318>
- [32] C. A. R. Hoare, I. J. Hayes, He Jifeng, C. C. Morgan, A. W. Roscoe, J. W. Sanders, I. H. Sorensen, J. M. Spivey, and B. A. Sufirin. 1987. Laws of programming. *Commun. ACM* 30, 8 (aug 1987), 672–686. <https://doi.org/10.1145/27651.27653>
- [33] C. A. R. Hoare, He Jifeng, and A. Sampaio. 1993. Normal form approach to compiler design. *Acta Informatica* 30, 8 (01 Aug 1993), 701–739. <https://doi.org/10.1007/BF01191809>
- [34] Ali Javadi-Abhari, Matthew Treinish, Kevin Krsulich, Christopher J. Wood, Jake Lishman, Julien Gacon, Simon Martiel, Paul D. Nation, Lev S. Bishop, Andrew W. Cross, Blake R. Johnson, and Jay M. Gambetta. 2024. Quantum computing with Qiskit. arXiv:2405.08810 [quant-ph] <https://arxiv.org/abs/2405.08810>
- [35] He Jifeng, K. Seidel, and A. McIver. 1997. Probabilistic models for the guarded command language. *Science of Computer Programming* 28, 2 (1997), 171–192. [https://doi.org/10.1016/S0167-6423\(96\)00019-6](https://doi.org/10.1016/S0167-6423(96)00019-6) Formal Specifications: Foundations, Methods, Tools and Applications.
- [36] Aleks Kissinger and John van de Wetering. 2019. PyZX: Large Scale Automated Diagrammatic Reasoning. In *Proceedings 16th International Conference on Quantum Physics and Logic, QPL 2019, Chapman University, Orange, CA, USA, June 10-14, 2019 (EPTCS, Vol. 318)*, Bob Coecke and Matthew Leifer (Eds.). 229–241. <https://doi.org/10.4204/EPTCS.318.14>
- [37] Aleks Kissinger and John van de Wetering. 2024. *Picturing Quantum Software: An Introduction to the ZX-Calculus and Quantum Compilation*. Preprint.
- [38] Aleks Kissinger and Vladimir Zamdzhiev. 2015. Quantomatic: A Proof Assistant for Diagrammatic Reasoning. In *Automated Deduction - CADE-25*, Amy P. Felty and Aart Middeldorp (Eds.). Springer International Publishing, Cham, 326–336.
- [39] Dexter Kozen. 1997. Kleene algebra with tests. *ACM Trans. Program. Lang. Syst.* 19, 3 (May 1997), 427–443. <https://doi.org/10.1145/256167.256195>
- [40] Adrian Lehmann, Ben Caldwell, and Robert Rand. 2022. VyZX : A Vision for Verifying the ZX Calculus. arXiv:2205.05781 [quant-ph] <https://arxiv.org/abs/2205.05781>
- [41] Adrian Lehmann, Ben Caldwell, Bhakti Shah, and Robert Rand. 2023. VyZX: Formal Verification of a Graphical Quantum Language. arXiv:2311.11571 [cs.PL] <https://arxiv.org/abs/2311.11571>
- [42] Marco Lewis, Sadegh Soudjani, and Paolo Zuliani. 2023. Formal Verification of Quantum Programs: Theory, Tools, and Challenges. 5, 1, Article 1 (dec 2023), 35 pages. <https://doi.org/10.1145/3624483>
- [43] Junyi Liu, Bohua Zhan, Shuling Wang, Shenggang Ying, Tao Liu, Yangjia Li, Mingsheng Ying, and Naijun Zhan. 2019. Formal Verification of Quantum Algorithms Using Quantum Hoare Logic. In *Computer Aided Verification*, Isil Dillig and Serdar Tasiran (Eds.). Springer International Publishing, Cham, 187–207.
- [44] Assia Mahboubi and Enrico Tassi. 2022. *Mathematical Components*. Zenodo. <https://doi.org/10.5281/zenodo.7118596>
- [45] Hynek Mlnarik. 2007. Quantum programming language LanQ. *Masaryk University-Faculty of Informatics, Czech Republic* (2007).
- [46] Carroll Morgan. 1990. *Programming from specifications*. Prentice-Hall, Inc., USA.
- [47] Ana Neri, Rui Soares Barbosa, and José N. Oliveira. 2022. Compiling Quantamorphisms for the IBM Q Experience. *IEEE Transactions on Software Engineering* 48, 11 (2022), 4339–4356. <https://doi.org/10.1109/TSE.2021.3117515>

- [48] Michael A Nielsen and Isaac L Chuang. 2010. *Quantum computation and quantum information*. Cambridge university press.
- [49] Bernhard Ömer. 2005. Classical Concepts in Quantum Programming. *International Journal of Theoretical Physics* 44, 7 (01 Jul 2005), 943–955. <https://doi.org/10.1007/s10773-005-7071-x>
- [50] Jennifer Paykin, Robert Rand, and Steve Zdancewic. 2017. QWIRE: a core language for quantum circuits. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, Giuseppe Castagna and Andrew D. Gordon (Eds.). ACM, 846–858. <http://dl.acm.org/citation.cfm?id=3009894>
- [51] Anurudh Peduri, Ina Schaefer, and Michael Walter. 2025. QbC: Quantum Correctness by Construction. *Proc. ACM Program. Lang.* 9, OOPSLA1, Article 99 (April 2025), 29 pages. <https://doi.org/10.1145/3720433>
- [52] Yuxiang Peng, Kesha Hietala, Runzhou Tao, Liyi Li, Robert Rand, Michael Hicks, and Xiaodi Wu. 2023. A formally certified end-to-end implementation of Shor’s factorization algorithm. *Proceedings of the National Academy of Sciences* 120, 21 (2023), e2218775120. <https://doi.org/10.1073/pnas.2218775120> arXiv:<https://www.pnas.org/doi/pdf/10.1073/pnas.2218775120>
- [53] Yuxiang Peng, Mingsheng Ying, and Xiaodi Wu. 2022. Algebraic reasoning of Quantum programs via non-idempotent Kleene algebra (*PLDI 2022*). Association for Computing Machinery, New York, NY, USA, 657–670. <https://doi.org/10.1145/3519939.3523713>
- [54] Robert Rand, Jennifer Paykin, Dong-Ho Lee, and Steve Zdancewic. 2018. ReQWIRE: Reasoning about Reversible Quantum Circuits. In *Proceedings 15th International Conference on Quantum Physics and Logic, QPL 2018, Halifax, Canada, 3-7th June 2018 (EPTCS, Vol. 287)*, Peter Selinger and Giulio Chiribella (Eds.). 299–312. <https://doi.org/10.4204/EPTCS.287.17>
- [55] Robert Rand, Jennifer Paykin, and Steve Zdancewic. 2017. QWIRE Practice: Formal Verification of Quantum Circuits in Coq. In *Proceedings 14th International Conference on Quantum Physics and Logic, QPL 2017, Nijmegen, The Netherlands, 3-7 July 2017. (EPTCS, Vol. 266)*, Bob Coecke and Aleks Kissinger (Eds.). 119–132. <https://doi.org/10.4204/EPTCS.266.8>
- [56] A.W. Roscoe and C.A.R. Hoare. 1988. The laws of OCCAM programming. *Theoretical Computer Science* 60, 2 (1988), 177–229. [https://doi.org/10.1016/0304-3975\(88\)90049-7](https://doi.org/10.1016/0304-3975(88)90049-7)
- [57] Amr Sabry, Benoît Valiron, and Juliana Kaizer Vizzotto. 2018. From Symmetric Pattern-Matching to Quantum Control. In *Foundations of Software Science and Computation Structures*, Christel Baier and Ugo Dal Lago (Eds.). Springer International Publishing, Cham, 348–364.
- [58] J. W. Sanders and P. Zuliani. 2000. Quantum Programming. In *Mathematics of Program Construction*, Roland Backhouse and José Nuno Oliveira (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 80–99.
- [59] Bhakti Shah, William Spencer, Laura Zielinski, Ben Caldwell, Adrian Lehmann, and Robert Rand. 2024. ViCAR: Visualizing Categories with Automated Rewriting in Coq. arXiv:[2404.08163](https://arxiv.org/abs/2404.08163) [cs.PL] <https://arxiv.org/abs/2404.08163>
- [60] Vivek V. Shende, Stephen S. Bullock, and Igor L. Markov. 2005. Synthesis of quantum logic circuits. In *Proceedings of the 2005 Asia and South Pacific Design Automation Conference (Shanghai, China) (ASP-DAC '05)*. Association for Computing Machinery, New York, NY, USA, 272–275. <https://doi.org/10.1145/1120725.1120847>
- [61] Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. 2020. t|ket>: a retargetable compiler for NISQ devices. *Quantum Science and Technology* 6, 1 (nov 2020), 014003. <https://doi.org/10.1088/2058-9565/ab8e92>
- [62] Sam Staton. 2015. Algebraic Effects, Linearity, and Quantum Programming Languages. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (Mumbai, India) (POPL '15)*. Association for Computing Machinery, New York, NY, USA, 395–406. <https://doi.org/10.1145/2676726.2676999>
- [63] Korbinian Staudacher. 2021. *Optimization Approaches for Quantum Circuits using ZX-calculus*. Ph. D. Dissertation. Master’s thesis, Ludwig-Maximilians-Universität, München. Available at <https://www.mn.uw.edu/~stauda/>
- [64] Damian S. Steiger, Thomas Häner, and Matthias Troyer. 2018. ProjectQ: an open source software framework for quantum computing. *Quantum* 2 (Jan. 2018), 49. <https://doi.org/10.22331/q-2018-01-31-49>
- [65] Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. 2018. Q#: Enabling Scalable Quantum Computing and Development with a High-level DSL. In *Proceedings of the Real World Domain Specific Languages Workshop 2018 (RWDSL2018)*. ACM. <https://doi.org/10.1145/3183895.3183901>
- [66] Runzhou Tao, Yunong Shi, Jianan Yao, Xupeng Li, Ali Javadi-Abhari, Andrew W. Cross, Frederic T. Chong, and Ronghui Gu. 2022. Giallar: push-button verification for the qiskit Quantum compiler. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation (San Diego, CA, USA) (PLDI 2022)*. Association for Computing Machinery, New York, NY, USA, 641–656. <https://doi.org/10.1145/3519939.3523431>
- [67] The Coq Development Team. 2023. *The Coq Proof Assistant*. <https://doi.org/10.5281/zenodo.8161141>
- [68] The MathComp Analysis Development Team. 2024. MathComp-Analysis: Mathematical Components compliant Analysis Library. <https://github.com/math-comp/analysis>. Since 2017. Version 1.0.0.

- [69] Dominique Unruh. 2019. Quantum Relational Hoare Logic. *Proc. ACM Program. Lang.* 3, POPL, Article 33 (jan 2019), 31 pages. <https://doi.org/10.1145/3290346>
- [70] Dominique Unruh. 2020. Post-Quantum Verification of Fujisaki-Okamoto. In *Advances in Cryptology – ASIACRYPT 2020*, Shihō Moriai and Huaxiong Wang (Eds.). Springer International Publishing, Cham, 321–352.
- [71] Benoît Valiron. 2022. Semantics of quantum programming languages: Classical control, quantum control. *Journal of Logical and Algebraic Methods in Programming* 128 (2022), 100790. <https://doi.org/10.1016/j.jlamp.2022.100790>
- [72] George F. Viamontes, Igor L. Markov, and John P. Hayes. 2007. Checking equivalence of quantum circuits and states. In *2007 IEEE/ACM International Conference on Computer-Aided Design*. 69–74. <https://doi.org/10.1109/ICCAD.2007.4397246>
- [73] Finn Voichick, Liyi Li, Robert Rand, and Michael Hicks. 2023. Qunity: A Unified Language for Quantum and Classical Computing. *Proc. ACM Program. Lang.* 7, POPL, Article 32 (jan 2023), 31 pages. <https://doi.org/10.1145/3571225>
- [74] Amanda Xu, Abtin Molavi, Lauren Pick, Swamit Tannu, and Aws Albarghouthi. 2023. Synthesizing Quantum-Circuit Optimizers. *Proc. ACM Program. Lang.* 7, PLDI, Article 140 (jun 2023), 25 pages. <https://doi.org/10.1145/3591254>
- [75] Mingkuan Xu, Zikun Li, Oded Padon, Sina Lin, Jessica Pointing, Auguste Hirth, Henry Ma, Jens Palsberg, Alex Aiken, Umut A. Acar, and Zhihao Jia. 2022. Quartz: superoptimization of Quantum circuits (*PLDI 2022*). Association for Computing Machinery, New York, NY, USA, 625–640. <https://doi.org/10.1145/3519939.3523433>
- [76] Zhaowei Xu, Mingsheng Ying, and Benoît Valiron. 2021. Reasoning about recursive quantum programs. arXiv:2107.11679 [cs.PL]
- [77] Mingsheng Ying. 2016. *Foundations of quantum programming*. Morgan Kaufmann.
- [78] Mingsheng Ying, Nengkun Yu, and Yuan Feng. 2012. Defining Quantum Control Flow. arXiv:1209.4379 [quant-ph]
- [79] Mingsheng Ying and Zhicheng Zhang. 2023. Quantum Recursive Programming with Quantum Case Statements. arXiv:2311.01725 [cs.PL] <https://arxiv.org/abs/2311.01725>
- [80] Mingsheng Ying and Zhicheng Zhang. 2024. Verification of Recursively Defined Quantum Circuits. arXiv:2404.05934 [quant-ph] <https://arxiv.org/abs/2404.05934>
- [81] Nengkun Yu. 2023. Structured theorem for quantum programs and its applications. *ACM Transactions on Software Engineering and Methodology* 32, 4, Article 103 (2023), 35 pages.
- [82] Charles Yuan and Michael Carbin. 2024. The T-Complexity Costs of Error Correction for Control Flow in Quantum Computation. *Proc. ACM Program. Lang.* 8, PLDI, Article 167 (June 2024), 26 pages. <https://doi.org/10.1145/3656397>
- [83] Charles Yuan, Agnes Villanyi, and Michael Carbin. 2024. Quantum Control Machine: The Limits of Control Flow in Quantum Programming. *Proc. ACM Program. Lang.* 8, OOPSLA1, Article 94 (apr 2024), 28 pages. <https://doi.org/10.1145/3649811>
- [84] Zhicheng Zhang and Mingsheng Ying. 2025. Quantum Register Machine: Efficient Implementation of Quantum Recursive Programs. *Proc. ACM Program. Lang.* 8, PLDI (June 2025). <https://doi.org/10.1145/3729283>
- [85] Li Zhou, Gilles Barthe, Pierre-Yves Strub, Junyi Liu, and Mingsheng Ying. 2023. CoqQ: Foundational Verification of Quantum Programs. *Proc. ACM Program. Lang.* 7, POPL, Article 29 (jan 2023), 33 pages. <https://doi.org/10.1145/3571222>
- [86] Paolo Zuliani. 2005. Compiling quantum programs. *Acta Informatica* 41, 7 (01 Jun 2005), 435–474. <https://doi.org/10.1007/s00236-005-0165-3>
- [87] Paolo Zuliani et al. 2004. Non-deterministic quantum programming. In *Proceedings of the 2nd International Workshop on Quantum Programming Languages (QPL)*, Vol. 33. TUCS, 179–195.

## Appendices

### A Proofs of the Laws

For readability, we omitted the detailed proofs of the laws developed in this paper from the main text. For the reader's convenience, we present them here.

#### A.1 Proofs of the Laws in Section 4

PROOF OF PROPOSITION 4.1. (1) By assumption, we obtain:

$$\begin{aligned}
 \llbracket C_0 \xleftarrow{|\psi_0\rangle} q \xrightarrow{|\psi_1\rangle} C_1 \rrbracket &= |\psi_0\rangle_q \langle \psi_0| \otimes \llbracket C_0 \rrbracket + |\psi_1\rangle_q \langle \psi_1| \otimes \llbracket C_1 \rrbracket \\
 &= U |\varphi_0\rangle_q \langle \varphi_0| U^\dagger \otimes \llbracket C_0 \rrbracket + U |\varphi_1\rangle_q \langle \varphi_1| U^\dagger \otimes \llbracket C_1 \rrbracket \\
 &= U (|\varphi_0\rangle_q \langle \varphi_0| \otimes \llbracket C_0 \rrbracket + |\varphi_1\rangle_q \langle \varphi_1| \otimes \llbracket C_1 \rrbracket) U^\dagger \\
 &= \llbracket RHS \rrbracket.
 \end{aligned}$$

(2) Directly by definition:

$$\llbracket LHS \rrbracket = |\phi_0\rangle \langle \phi_0| \otimes \llbracket C_0 \rrbracket + |\phi_1\rangle \langle \phi_1| \otimes \llbracket C_1 \rrbracket = \llbracket RHS \rrbracket.$$

(3) It holds that  $\llbracket C \leftarrow q \rightarrow C \rrbracket = |0\rangle \langle 0| \otimes \llbracket C \rrbracket + |1\rangle \langle 1| \otimes \llbracket C \rrbracket = (|0\rangle \langle 0| + |1\rangle \langle 1|) \otimes \llbracket C \rrbracket = \llbracket C \rrbracket$ .

(4) By definition, we have:

$$\begin{aligned}
 \llbracket RHS \rrbracket &= |0\rangle_{q_2} \langle 0| \otimes \llbracket C \leftarrow q_1 \rightarrow C_0 \rrbracket + |1\rangle_{q_2} \langle 1| \otimes \llbracket C \leftarrow q_1 \rightarrow C_1 \rrbracket \\
 &= |0\rangle_{q_2} \langle 0| \otimes (|0\rangle_{q_1} \langle 0| \otimes \llbracket C \rrbracket + |1\rangle_{q_1} \langle 1| \otimes \llbracket C_0 \rrbracket) \\
 &\quad + |1\rangle_{q_2} \langle 1| \otimes (|0\rangle_{q_1} \langle 0| \otimes \llbracket C \rrbracket + |1\rangle_{q_1} \langle 1| \otimes \llbracket C_1 \rrbracket) \\
 &= |0\rangle_{q_1} \langle 0| \otimes (|0\rangle_{q_2} \langle 0| + |1\rangle_{q_2} \langle 1|) \otimes \llbracket C \rrbracket \\
 &\quad + |1\rangle_{q_1} \langle 1| \otimes (|0\rangle_{q_2} \langle 0| \otimes \llbracket C_0 \rrbracket + |1\rangle_{q_2} \langle 1| \otimes \llbracket C_1 \rrbracket) \\
 &= |0\rangle_{q_1} \langle 0| \otimes \llbracket C \rrbracket + |1\rangle_{q_1} \langle 1| \otimes (\llbracket C_0 \leftarrow q_2 \rightarrow C_1 \rrbracket) \\
 &= \llbracket LHS \rrbracket.
 \end{aligned}$$

(5) This follows by a straightforward expansion using the definition. □

PROOF OF PROPOSITION 4.2. (1–6) These follow immediately from the definition.

(7) We have:

$$\begin{aligned}
 \llbracket LHS \rrbracket &= (|0\rangle_q \langle 0| \otimes \llbracket D_0 \rrbracket + |1\rangle_q \langle 1| \otimes \llbracket D_1 \rrbracket) (|0\rangle_q \langle 0| \otimes \llbracket C_0 \rrbracket + |1\rangle_q \langle 1| \otimes \llbracket C_1 \rrbracket) \\
 &= |0\rangle_q \langle 0|_q \langle 0| \otimes \llbracket D_0 \rrbracket \llbracket C_0 \rrbracket + |0\rangle_q \langle 0|_q \langle 1| \otimes \llbracket D_0 \rrbracket \llbracket C_1 \rrbracket \\
 &\quad + |1\rangle_q \langle 1|_q \langle 0| \otimes \llbracket D_1 \rrbracket \llbracket C_0 \rrbracket + |1\rangle_q \langle 1|_q \langle 1| \otimes \llbracket D_1 \rrbracket \llbracket C_1 \rrbracket \\
 &= |0\rangle_q \langle 0| \otimes \llbracket C_0; D_0 \rrbracket + |1\rangle_q \langle 1| \otimes \llbracket C_1; D_1 \rrbracket \\
 &= \llbracket RHS \rrbracket.
 \end{aligned}$$

(8) This follows by a calculation similar to (7). □

PROOF OF PROPOSITION 4.3. (1) This can be shown by a routine calculation:

$$\begin{aligned}
 \llbracket LHS \rrbracket &= (|\psi_0\rangle_q \langle \psi_0| \otimes \llbracket C_0 \rrbracket + |\psi_1\rangle_q \langle \psi_1| \otimes \llbracket C_1 \rrbracket) U_q \\
 &= U_q |\phi_0\rangle \langle \phi_0| \otimes \llbracket C_0 \rrbracket + U_q |\phi_1\rangle_q \langle \phi_1| \otimes \llbracket C_1 \rrbracket \\
 &= \llbracket RHS \rrbracket.
 \end{aligned}$$

(2) This follows immediately from Proposition 4.2(7).

(3) We first note that  $q \notin qv(C)$  and  $qv(D) = \{q\}$  imply  $qv(C) \cap qv(D) = \emptyset$ . Then by Proposition 4.2(5) we obtain:

$$\begin{aligned} C; C_0 \leftarrow D[q] \rightarrow C_1 &\equiv C; D; C_0 \leftarrow q \rightarrow C_1 \\ &\equiv D; C; C_0 \leftarrow q \rightarrow C_1 \\ &\equiv D; (C; C_0) \leftarrow q \rightarrow (C; C_1) \\ &\equiv (C; C_0) \leftarrow D[q] \rightarrow (C; C_1). \end{aligned}$$

Other equivalences in this clause can be proved in a similar manner.  $\square$

## A.2 Proofs of the Laws in Section 5

PROOF OF PROPOSITION 5.1. (1) For any density operator  $\rho$ , by Proposition 3.1(2) we have:

$$\begin{aligned} \llbracket \bar{q} := |\varphi\rangle; \bar{q} := |\psi\rangle \rrbracket(\rho) &= \llbracket \bar{q} := |\psi\rangle \rrbracket \left( \sum_n |\varphi\rangle_{\bar{q}} \langle n|\rho|n\rangle_{\bar{q}} \langle \varphi| \right) \\ &= \sum_{n'} |\psi\rangle_{\bar{q}} \langle n'| \left( \sum_n |\varphi\rangle_{\bar{q}} \langle n|\rho|n\rangle_{\bar{q}} \langle \varphi| \right) |n'\rangle_{\bar{q}} \langle \psi| \\ &= \left( \sum_{n'} \langle n'|\varphi\rangle \langle \varphi|n'\rangle \right) \left( \sum_n |\psi\rangle_{\bar{q}} \langle n|\rho|n\rangle_{\bar{q}} \langle \psi| \right) \\ &= \sum_n |\psi\rangle_{\bar{q}} \langle n|\rho|n\rangle_{\bar{q}} \langle \psi| \\ &= \llbracket \bar{q} := |\psi\rangle \rrbracket(\rho), \end{aligned}$$

because

$$\sum_{n'} \langle n'|\varphi\rangle \langle \varphi|n'\rangle = \sum_{n'} |\langle \varphi|n'\rangle|^2 = \|\varphi\|^2 = 1.$$

When  $\bar{q}_1$  and  $\bar{q}_2$  are distinct, we observe:

$$\begin{aligned} \llbracket \bar{q}_1 := |\phi\rangle; \bar{q}_2 := |\psi\rangle \rrbracket(\rho) &= \sum_{n_2} |\psi\rangle_{\bar{q}_2} \langle n_2| \left( \sum_{n_1} |\phi\rangle_{\bar{q}_1} \langle n_1|\rho|n_1\rangle_{\bar{q}_1} \langle \phi| \right) |n_2\rangle_{\bar{q}_2} \langle \psi| \\ &= \sum_{(n_1, n_2)} (|\phi\rangle|\psi\rangle)_{\bar{q}_1, \bar{q}_2} (\langle n_1|\langle n_2|\rho(|n_1\rangle|n_2\rangle)_{\bar{q}_1, \bar{q}_2} (\langle \phi|\langle \psi|) \\ &= \llbracket \bar{q}_1, \bar{q}_2 := |\phi\rangle|\psi\rangle \rrbracket(\rho). \end{aligned}$$

(2) For any density operator  $\rho$ , by Proposition 3.1(2, 3) we have:

$$\begin{aligned} \llbracket \bar{q} := |\psi\rangle; C \rrbracket(\rho) &= \llbracket C \rrbracket \left( \sum_n |\psi\rangle_{\bar{q}} \langle n|\rho|n\rangle_{\bar{q}} \langle \psi| \right) \llbracket C \rrbracket^\dagger \\ &= \sum_n (\llbracket C \rrbracket |\psi\rangle)_{\bar{q}} \langle n|\rho|n\rangle_{\bar{q}} (\llbracket C \rrbracket |\psi\rangle)_{\bar{q}}^\dagger \\ &= \llbracket \bar{q} := \llbracket C \rrbracket |\psi\rangle \rrbracket(\rho). \end{aligned}$$

(3) Let  $A_i = |\varphi_i\rangle\langle\varphi_i| \otimes \llbracket C_i \rrbracket$  for  $i = 0, 1$ . Then  $|\psi\rangle \perp |\varphi_0\rangle$  implies: for any density operator  $\rho$ ,

$$A_0|\psi\rangle\langle n|\rho|n\rangle\langle\psi|A_0^\dagger = A_0|\psi\rangle\langle n|\rho|n\rangle\langle\psi|A_1^\dagger = A_1|\psi\rangle\langle n|\rho|n\rangle\langle\psi|A_0^\dagger = 0.$$

Additionally, we notice that, since  $q$  is a qubit and thus its state space is of dimension two,  $|\psi\rangle$  and  $|\varphi_1\rangle$  differ only in a phase coefficient, i.e.,  $|\psi\rangle = e^{i\theta}|\varphi_1\rangle$  with phase  $\theta \in \mathbb{R}$ . Therefore, the following equations hold:

$$(\langle\varphi_1|\psi\rangle|\varphi_1\rangle)(\langle\varphi_1|\psi\rangle|\varphi_1\rangle)^\dagger = |\varphi_1\rangle\langle\varphi_1| = |\psi\rangle\langle\psi|.$$

Consequently, with Proposition 3.1(2) and (3) we have:

$$\begin{aligned} \llbracket LHS \rrbracket(\rho) &= (A_0 + A_1) \left( \sum_n |\psi\rangle\langle n|\rho|n\rangle\langle\psi| \right) (A_0^\dagger + A_1^\dagger) \\ &= \sum_n A_1 |\psi\rangle\langle n|\rho|n\rangle\langle\psi| A_1^\dagger \\ &= \sum_n (|\varphi_1\rangle\langle\varphi_1|\psi\rangle \otimes \llbracket C_1 \rrbracket) \langle n|\rho|n\rangle (\langle\varphi_1|\langle\varphi_1|\psi\rangle)^\dagger \otimes \llbracket C_1 \rrbracket^\dagger \\ &= \llbracket C_1 \rrbracket \left( \sum_n (\langle\varphi_1|\psi\rangle|\varphi_1\rangle) \langle n|\rho|n\rangle (\langle\varphi_1|\psi\rangle|\varphi_1\rangle)^\dagger \right) \llbracket C_1 \rrbracket^\dagger \\ &= \llbracket C_1 \rrbracket \left( \sum_n |\varphi_1\rangle\langle n|\rho|n\rangle\langle\varphi_1| \right) \llbracket C_1 \rrbracket^\dagger \\ &= \llbracket C_1 \rrbracket (\llbracket q := |\varphi_1| \rrbracket(\rho)) \\ &= \llbracket C_1 \rrbracket \left( \sum_n |\psi\rangle\langle n|\rho|n\rangle\langle\psi| \right) \llbracket C_1 \rrbracket^\dagger \\ &= \llbracket C_1 \rrbracket (\llbracket q := |\psi| \rrbracket(\rho)). \end{aligned}$$

More generally, let  $|\varphi\rangle = \sum_i \lambda_i |\phi_i\rangle|\psi_i\rangle$ , notice that:

$$\left( \sum_{i=1}^d |\phi_i\rangle\langle\phi_i| \otimes U_i \right) |\varphi\rangle = \sum_i \sum_j \lambda_j |\phi_i\rangle\langle\phi_i|\phi_j\rangle (U_i|\psi_j\rangle) = \sum_i \lambda_i |\phi_i\rangle (U_i|\psi_i\rangle),$$

so we have

$$\begin{aligned} \llbracket \bar{q}, \bar{r} := \sum_i \lambda_i |\phi_i\rangle|\psi_i\rangle; \mathbf{qif} [\bar{q}] \left( \square_{i=1}^d |\phi_i\rangle \rightarrow C_i \right) \mathbf{fiq} \rrbracket(\rho) \\ &= \left( \sum_i |\phi_i\rangle_{\bar{q}}\langle\phi_i| \otimes (U_i)_{\bar{r}} \right) \left( \sum_k |\varphi\rangle_{\bar{q},\bar{r}}\langle k|\rho|k\rangle_{\bar{q},\bar{r}}\langle\varphi| \right) \left( \sum_i |\phi_i\rangle_{\bar{q}}\langle\phi_i| \otimes (U_i)_{\bar{r}} \right)^\dagger \\ &= \sum_k \left( \left( \sum_i |\phi_i\rangle\langle\phi_i| \otimes U_i \right) |\varphi\rangle \right)_{\bar{q},\bar{r}} \langle k|\rho|k\rangle_{\bar{q},\bar{r}} \left( \left( \sum_i |\phi_i\rangle\langle\phi_i| \otimes U_i \right) |\varphi\rangle \right)_{\bar{q},\bar{r}}^\dagger \\ &= \llbracket \bar{q}, \bar{r} := \sum_i \lambda_i |\phi_i\rangle (U_i|\psi_i\rangle) \rrbracket(\rho). \end{aligned}$$

(4) We only prove the second law, and the first can be shown in a similar way. First, from the assumption that  $M = \llbracket |\psi\rangle \rrbracket$  and  $|\varphi\rangle \perp |\psi\rangle$ , it follows that for any density operator  $\rho$ ,

$$M_0 |\varphi\rangle\langle n|\rho|n\rangle\langle\varphi| M_0^\dagger = (I - |\psi\rangle\langle\psi|) |\varphi\rangle\langle n|\rho|n\rangle\langle\varphi| (I - |\psi\rangle\langle\psi|) = |\varphi\rangle\langle n|\rho|n\rangle\langle\varphi|$$

and  $M_1|\varphi\rangle\langle n|\rho|n\rangle\langle\varphi|M_1^\dagger = 0$ . Then using Proposition 3.1(2) and (5) we have:

$$\begin{aligned}
\llbracket LHS \rrbracket(\rho) &= \llbracket P_0 \triangleleft M[q] \triangleright P_1 \rrbracket \left( \sum_n |\varphi\rangle\langle n|\rho|n\rangle\langle\varphi| \right) \\
&= \sum_n \llbracket P_0 \triangleleft M[q] \triangleright P_1 \rrbracket (|\varphi\rangle\langle n|\rho|n\rangle\langle\varphi|) \\
&= \sum_n \left[ \llbracket P_0 \rrbracket (M_0|\varphi\rangle\langle n|\rho|n\rangle\langle\varphi|M_0^\dagger) + \llbracket P_1 \rrbracket (M_1|\varphi\rangle\langle n|\rho|n\rangle\langle\varphi|M_1^\dagger) \right] \\
&= \sum_n \llbracket P_0 \rrbracket (|\varphi\rangle\langle n|\rho|n\rangle\langle\varphi|) \\
&= \llbracket P_0 \rrbracket \left( \sum_n |\varphi\rangle\langle n|\rho|n\rangle\langle\varphi| \right) \\
&= \llbracket RHS \rrbracket(\rho).
\end{aligned}$$

(5) We calculate as follows according to Proposition 3.1(5,2):

$$\begin{aligned}
\llbracket LHS \rrbracket(\rho) &= \sum_k |0\rangle_{\bar{r}}\langle k| \left( \sum_m (|m\rangle_{\bar{q}}\langle m| \otimes \llbracket P_m \rrbracket) \rho(|m\rangle_{\bar{q}}\langle m| \otimes \llbracket P_m \rrbracket) \right) |k\rangle_{\bar{r}}\langle 0| \\
&= \sum_k |0\rangle_{\bar{r}}\langle k| (|k\rangle_{\bar{q}}\langle k| \otimes \llbracket P_{k\downarrow\bar{q}} \rrbracket) \rho(|k\rangle_{\bar{q}}\langle k| \otimes \llbracket P_{k\downarrow\bar{q}} \rrbracket) |k\rangle_{\bar{r}}\langle 0| \\
&= \sum_k (|0\rangle_{\bar{r}}\langle k| \otimes \llbracket P_{k\downarrow\bar{q}} \rrbracket) \rho(|k\rangle_{\bar{r}}\langle 0| \otimes \llbracket P_{k\downarrow\bar{q}} \rrbracket) \\
&= \sum_k |0\rangle_{\bar{r}}\langle k| \left( \sum_{k'} (|k'\rangle_{\bar{r}}\langle k'| \otimes \llbracket P_{k\downarrow\bar{q}} \rrbracket) \rho(|k'\rangle_{\bar{r}}\langle k'| \otimes \llbracket P_{k\downarrow\bar{q}} \rrbracket) \right) |k\rangle_{\bar{r}}\langle 0| \\
&= \llbracket RHS \rrbracket(\rho).
\end{aligned}$$

□

To prove other results in Section 5, we need some technical lemmas:

LEMMA A.1. *Let  $A$  and  $B$  be two linear operators. If for every  $|\alpha\rangle$  there exists  $k_{|\alpha\rangle}$  such that*

$$B|\alpha\rangle = k_{|\alpha\rangle}A|\alpha\rangle$$

*then  $B = cA$  for some  $c \in \mathbb{C}$ , i.e.,  $B \bowtie_{|c|} A$ .*

PROOF. Let  $A = UDV^\dagger$  be the singular value decomposition of  $A$ , i.e.,  $U$  and  $V$  are unitary operators and  $D = \text{diag}(d_1, d_2, \dots, d_n)$  where  $d_i \geq 0$  for all  $i$ . Notice that:

$$\forall |\alpha\rangle, BVV^\dagger|\alpha\rangle = k_{|\alpha\rangle}UDV^\dagger|\alpha\rangle \implies \forall |\alpha\rangle, BV|\alpha\rangle = k_{V^\dagger|\alpha\rangle}UD|\alpha\rangle.$$

We write  $BV = (|v_1\rangle, |v_2\rangle, \dots, |v_n\rangle)$  and  $U = (|u_1\rangle, |u_2\rangle, \dots, |u_n\rangle)$  in the column vector form. Considering  $|\alpha\rangle = |i\rangle$ , we have:

$$\forall i, |v_i\rangle = c_i d_i |u_i\rangle$$

where  $c_i = k_{V^\dagger|i\rangle}$ . For any  $i \neq j$  such that  $d_i \neq 0$  and  $d_j \neq 0$ , select  $|\alpha\rangle = |i\rangle + |j\rangle$ , we have

$$\begin{aligned}
BV|\alpha\rangle &= BV(|i\rangle + |j\rangle) = |v_i\rangle + |v_j\rangle = c_i d_i |u_i\rangle + c_j d_j |u_j\rangle \\
&= k_{V^\dagger(|i\rangle+|j\rangle)} UD(|i\rangle + |j\rangle) = k_{V^\dagger(|i\rangle+|j\rangle)} (d_i |u_i\rangle + d_j |u_j\rangle),
\end{aligned}$$

which shows that  $c_i = c_j = k_{V^\dagger(|i\rangle+|j\rangle)}$ . This leads to  $c_i = c$  if  $d_i \neq 0$  for some constant  $c \in \mathbb{C}$ . Furthermore, if  $d_i = 0$ , then  $|v_i\rangle = 0 = cd_i|u_i\rangle$ . Therefore

$$BV = (cd_1|u_1\rangle, \dots, cd_n|u_n\rangle) = cUD,$$

and hence

$$B = BVV^\dagger = cUDV^\dagger = cA.$$

□

LEMMA A.2. For linear operators  $A, B$ , we have the following:

- (1)  $A \perp B$  iff for every state  $\rho$ ,  $AB\rho B^\dagger A^\dagger = 0$ .
- (2)  $A \times B$  iff for every state  $\rho$ ,  $AB\rho B^\dagger A^\dagger = B\rho B^\dagger$ .
- (3) The following two statements are equivalent:
  - (a)  $A_0 \bowtie_{c_0} B$  and  $A_1 \bowtie_{c_1} B$  for some  $c_0$  and  $c_1$  with  $c_0^2 + c_1^2 = 1$ ;
  - (b) for every state  $\rho$ ,  $A_0\rho A_0^\dagger + A_1\rho A_1^\dagger = B\rho B^\dagger$ .
 As a direct corollary, the following two statements are equivalent:
  - (c)  $B \bowtie_{c_0} A_0$  and  $B \bowtie_{c_1} A_1$  for some  $c_0$  and  $c_1$  with  $c_0^2 + c_1^2 = 1$ ;
  - (d) for every state  $\rho$ ,  $BA_0\rho A_0^\dagger B^\dagger + BA_1\rho A_1^\dagger B^\dagger = B\rho B^\dagger$ .
- (4) If  $\{A_0, A_1\}$  is a binary measurement, then  $A_0 \times B$  implies  $A_1 \perp B$ ; symmetrically,  $A_1 \times B$  implies  $A_0 \perp B$ .

PROOF. (1) “ $\Rightarrow$ ”: This is immediate, since  $A \perp B$  means  $AB = 0$ .

“ $\Leftarrow$ ”: Set  $\rho = |\alpha\rangle\langle\alpha|$  be arbitrary pure state. Then  $AB\rho B^\dagger A^\dagger = (AB|\alpha\rangle)(AB|\alpha\rangle)^\dagger = 0$  implies  $AB|\alpha\rangle = 0$ , so  $AB = 0$ , i.e.,  $A \perp B$ .

(2) “ $\Rightarrow$ ”:  $A \times B$ , so there exists  $c \in \mathbb{C}$ , such that  $|c| = 1$  and  $AB = cB$ . Thus, we have

$$AB\rho B^\dagger A^\dagger = cB\rho(cB)^\dagger = |c|^2 B\rho B^\dagger = B\rho B^\dagger.$$

“ $\Leftarrow$ ”: Let  $\rho = |\alpha\rangle\langle\alpha|$  be an arbitrary pure state. Then

$$AB\rho B^\dagger A^\dagger = (AB|\alpha\rangle)(AB|\alpha\rangle)^\dagger = (B|\alpha\rangle)(B|\alpha\rangle)^\dagger$$

which implies  $AB|\alpha\rangle = k_{|\alpha\rangle}B|\alpha\rangle$  where  $k_{|\alpha\rangle} \in \mathbb{C}$  such that  $|k_{|\alpha\rangle}| = 1$ . We apply Lemma A.1 and assert that there exists  $c \in \mathbb{C}$  such that  $AB = cB$ . Back to the assumption, we have:

$$AB\rho B^\dagger A^\dagger = cB\rho(cB)^\dagger = |c|^2 B\rho B^\dagger$$

So,  $|c| = 1$ , i.e.,  $A \times B$  if  $B \neq 0$ . If  $B = 0$ , then  $A \times B$  also holds trivially.

(3) “(a)  $\Rightarrow$  (b)”: By assumption, there exists  $d_0, d_1 \in \mathbb{C}$  such that  $|d_0| = c_0$ ,  $|d_1| = c_1$ ,  $A_0 = d_0B$  and  $A_1 = d_1B$ . So,

$$A_0\rho A_0^\dagger + A_1\rho A_1^\dagger = |d_0|^2 B\rho B^\dagger + |d_1|^2 B\rho B^\dagger = (c_0^2 + c_1^2)B\rho B^\dagger = B\rho B^\dagger.$$

“(b)  $\Leftarrow$  (a)”: Let  $\rho = |\alpha\rangle\langle\alpha|$  be an arbitrary pure state. Then

$$(A_0|\alpha\rangle)(A_0|\alpha\rangle)^\dagger + (A_1|\alpha\rangle)(A_1|\alpha\rangle)^\dagger = (B|\alpha\rangle)(B|\alpha\rangle)^\dagger.$$

Since the RHS is rank one, both LHS terms must be proportional to the same vector. Therefore, there exist  $k_{0,|\alpha\rangle}, k_{1,|\alpha\rangle} \in \mathbb{C}$  such that  $A_0|\alpha\rangle = k_{0,|\alpha\rangle}B|\alpha\rangle$  and  $A_1|\alpha\rangle = k_{1,|\alpha\rangle}B|\alpha\rangle$ . According to Lemma A.1, there exists  $d_0, d_1 \in \mathbb{C}$  such that  $A_0 = d_0B$  and  $A_1 = d_1B$ . Back to the assumption, we have

$$A_0\rho A_0^\dagger + A_1\rho A_1^\dagger = B\rho B^\dagger = (|d_0|^2 + |d_1|^2)B\rho B^\dagger,$$

so  $|d_0|^2 + |d_1|^2 = 1$  and  $A_0 \bowtie_{|d_0|} B$  and  $A_1 \bowtie_{|d_1|} B$  if  $B \neq 0$ . If  $B = 0$ , then it holds trivially.

(4) By the assumption, there exists  $c \in \mathbb{C}$  such that  $|c| = 1$  and  $A_0B = cB$ . Note that

$$\begin{aligned} (A_0B)^\dagger(A_0B) + (A_1B)^\dagger(A_1B) &= B^\dagger(A_0^\dagger A_0 + A_1^\dagger A_1)B = B^\dagger B \\ &= (cB)^\dagger(cB) + (A_1B)^\dagger(A_1B) = B^\dagger B + (A_1B)^\dagger(A_1B) \end{aligned}$$

which implies  $(A_1B)^\dagger(A_1B) = 0$ . So  $A_1B = 0$ , i.e.,  $A_1 \perp B$ . □

Now we are able to prove the remaining laws in Section 5.

PROOF OF PROPOSITION 5.2. The proofs of laws (1) and (3) are straightforward and thus omitted.

(2) For any density operator  $\rho$ , it follows from the linearity of  $\llbracket P \rrbracket$  and Proposition 3.1(5) that

$$\begin{aligned} \llbracket P \triangleleft M \triangleright P \rrbracket(\rho) &= \llbracket P \rrbracket(M_0\rho M_0^\dagger) + \llbracket P \rrbracket(M_1\rho M_1^\dagger) \\ &= \llbracket P \rrbracket(M_0\rho M_0^\dagger + M_1\rho M_1^\dagger) \\ &= \llbracket P \rrbracket(\llbracket [M] \rrbracket(\rho)) \\ &= \llbracket [M]; P \rrbracket(\rho). \end{aligned}$$

(4) Since  $M$  is a projective measurement, we have  $M_0M_1 = 0$  and  $M_1M_1 = M_1$ . Then for any  $\rho$ , using Proposition 3.1(5) we obtain:

$$\begin{aligned} \llbracket P \triangleleft M \triangleright (Q \triangleleft M \triangleright R) \rrbracket(\rho) &= \llbracket P \rrbracket(M_0\rho M_0^\dagger) + \llbracket Q \triangleleft M \triangleright R \rrbracket(M_1\rho M_1^\dagger) \\ &= \llbracket P \rrbracket(M_0\rho M_0^\dagger) + (\llbracket Q \rrbracket(M_0M_1\rho M_1^\dagger M_0^\dagger) + \llbracket R \rrbracket(M_1M_1\rho M_1^\dagger M_1^\dagger)) \\ &= \llbracket P \rrbracket(M_0\rho M_0^\dagger) + \llbracket R \rrbracket(M_1\rho M_1^\dagger) \\ &= \llbracket P \triangleleft M \triangleright R \rrbracket(\rho). \end{aligned}$$

Similarly, we can prove that  $\llbracket (P \triangleleft M \triangleright Q) \triangleleft M \triangleright R \rrbracket(\rho) = \llbracket P \triangleleft M \triangleright R \rrbracket(\rho)$ .

(5) For the first law, for any  $\rho$ , by Proposition 3.1(5) we have:

$$\begin{aligned} \llbracket LHS \rrbracket(\rho) &= \llbracket P \triangleleft N \triangleright Q \rrbracket(M_1\rho M_1^\dagger) \\ &= \llbracket P \rrbracket(N_0M_1\rho M_1^\dagger N_0^\dagger) + \llbracket Q \rrbracket(N_1M_1\rho M_1^\dagger N_1^\dagger) \\ &= \llbracket Q \rrbracket(N_1M_1\rho M_1^\dagger N_1^\dagger) \\ &= \llbracket Q \rrbracket(\llbracket [K] \rrbracket(\rho)) \\ &= \llbracket RHS \rrbracket(\rho). \end{aligned}$$

Here, the third equality comes from the assumption that  $M \blacktriangleright N$ , i.e.,  $N_0M_1 = 0$ , and the fourth equality comes from the assumption that  $K = M \& N$ , i.e.,  $K_1 = N_1M_1$ .

For the second law, since  $M \propto N$ , i.e.,  $M_1 \propto N_1$ , by Lemma A.2, we must have  $M_1N_1\rho N_1^\dagger M_1^\dagger = N_1\rho N_1^\dagger$  and  $M_0N_1\rho N_1^\dagger M_0^\dagger = 0$  for all  $\rho$ . Thus, the following hold:

$$\begin{aligned} \llbracket LHS \rrbracket(\rho) &= \llbracket P \rrbracket(N_0\rho N_0^\dagger) + (\llbracket Q \rrbracket(M_0N_1\rho N_1^\dagger M_0^\dagger) + \llbracket R \rrbracket(M_1N_1\rho N_1^\dagger M_1^\dagger)) \\ &= \llbracket P \rrbracket(N_0\rho N_0^\dagger) + \llbracket R \rrbracket(N_1\rho N_1^\dagger) \\ &= \llbracket P \triangleleft N \triangleright R \rrbracket(\rho). \end{aligned}$$

This completes the proof of the second law. □

Before presenting the proof of Proposition 5.3, we state a lemma that clarifies the conditions required in the laws of the proposition. Indeed, this lemma is also needed in the proof of these laws.

LEMMA A.3. *The following equations hold:*

- $[N] \equiv [K] \triangleleft M \triangleright [L]$  iff  $K_0 M_0 \bowtie_{c_0} N_0$  and  $L_0 M_1 \bowtie_{c_1} N_0$  for some  $c_0$  and  $c_1$  with  $c_0^2 + c_1^2 = 1$ ;
- $[N] \equiv [K] \triangleleft M \triangleright [L]$  iff  $K_1 M_0 \bowtie_{c_0} N_1$  and  $L_1 M_1 \bowtie_{c_1} N_1$  for some  $c_0$  and  $c_1$  with  $c_0^2 + c_1^2 = 1$ ;
- $[N] \equiv [M]; [N]$  iff  $N_0 \bowtie_{c_0} M_0$  and  $N_0 \bowtie_{c_1} M_1$  with  $c_0^2 + c_1^2 = 1$ ;
- $[N] \equiv [M]; [N]$  iff  $N_1 \bowtie_{c_0} M_0$  and  $N_1 \bowtie_{c_1} M_1$  with  $c_0^2 + c_1^2 = 1$ ;

PROOF. Direct calculation from the definition and Lemma A.2.  $\square$

Now we are in a position to present:

PROOF OF PROPOSITION 5.3. (1) For any density operator  $\rho$ , it follows from the assumptions  $[N] \equiv [K] \triangleleft M \triangleright [L]$  and  $[N] \equiv [K] \triangleleft M \triangleright [L]$  that

$$N_i \rho N_i^\dagger = K_i M_0 \rho M_0^\dagger K_0^\dagger + L_i M_1 \rho M_1^\dagger L_i^\dagger \quad (i = 0, 1).$$

Then, by Proposition 3.1(5) we obtain

$$\begin{aligned} \llbracket LHS \rrbracket(\rho) &= \llbracket P \triangleleft K \triangleright Q \rrbracket(M_0 \rho M_0^\dagger) + \llbracket P \triangleleft L \triangleright Q \rrbracket(M_1 \rho M_1^\dagger) \\ &= (\llbracket P \rrbracket(K_0 M_0 \rho M_0^\dagger K_0^\dagger) + \llbracket Q \rrbracket(K_1 M_0 \rho M_0^\dagger K_1^\dagger)) \\ &\quad + (\llbracket P \rrbracket(L_0 M_1 \rho M_1^\dagger L_0^\dagger) + \llbracket Q \rrbracket(L_1 M_1 \rho M_1^\dagger L_1^\dagger)) \\ &= \llbracket P \rrbracket(K_0 M_0 \rho M_0^\dagger K_0^\dagger + L_0 M_1 \rho M_1^\dagger L_0^\dagger) + \llbracket Q \rrbracket(K_1 M_0 \rho M_0^\dagger K_1^\dagger + L_1 M_1 \rho M_1^\dagger L_1^\dagger) \\ &= \llbracket P \rrbracket(N_0 \rho N_0^\dagger) + \llbracket Q \rrbracket(N_1 \rho N_1^\dagger) \\ &= \llbracket RHS \rrbracket(\rho). \end{aligned}$$

(2) For any  $\rho$ , by Proposition 3.1(5) we have:

$$\begin{aligned} \llbracket LHS \rrbracket(\rho) &= \llbracket R \rrbracket(N_0 \rho N_0^\dagger) + \llbracket P \triangleleft M \triangleright Q \rrbracket(N_1 \rho N_1^\dagger) \\ &= \llbracket R \rrbracket(N_0 \rho N_0^\dagger) + \llbracket P \rrbracket(M_0 N_1 \rho N_1^\dagger M_0^\dagger) + \llbracket Q \rrbracket(M_1 N_1 \rho N_1^\dagger M_1^\dagger) \\ &= \llbracket R \rrbracket(N_0 (M_0 \rho M_0^\dagger + M_1 \rho M_1^\dagger) N_0^\dagger) + \llbracket P \rrbracket(N_1 M_0 \rho M_0^\dagger N_1^\dagger) + \llbracket Q \rrbracket(N_1 M_1 \rho M_1^\dagger N_1^\dagger) \\ &= \left( \llbracket R \rrbracket(N_0 M_0 \rho M_0^\dagger N_0^\dagger) + \llbracket P \rrbracket(N_1 M_0 \rho M_0^\dagger N_1^\dagger) \right) \\ &\quad + \left( \llbracket R \rrbracket(N_0 M_1 \rho M_1^\dagger N_0^\dagger) + \llbracket Q \rrbracket(N_1 M_1 \rho M_1^\dagger N_1^\dagger) \right) \\ &= \llbracket R \triangleleft N \triangleright P \rrbracket(M_0 \rho M_0^\dagger) + \llbracket R \triangleleft N \triangleright Q \rrbracket(M_1 \rho M_1^\dagger) \\ &= \llbracket RHS \rrbracket(\rho). \end{aligned}$$

Here, the third equality follows from the assumption that  $[N] \equiv [M]; [N]$  and  $M \diamond_L N$ .

(3) can be proved in a way similar to (2).

(4) and (5) are special cases of (2) and (3), and are therefore omitted.  $\square$

PROOF OF PROPOSITION 5.4. (1) and (2) are trivial.

(3) By Proposition 3.1(4), it holds that for any density operator  $\rho$ ,

$$\begin{aligned} \llbracket (P; Q); R \rrbracket(\rho) &= \llbracket R \rrbracket(\llbracket Q; R \rrbracket(\rho)) \\ &= \llbracket R \rrbracket(\llbracket Q \rrbracket(\llbracket P \rrbracket(\rho))) \\ &= \llbracket Q; R \rrbracket(\llbracket P \rrbracket(\rho)) \\ &= \llbracket P; (Q; R) \rrbracket(\rho). \end{aligned}$$

(4) Since  $M$  is a projective measurement, so  $M_m^\dagger = M_m$  and  $M_m M_m = M_m$  for  $m = 0, 1$ ,  $M_0 M_1 = M_1 M_0 = 0$ , thus:

$$\begin{aligned} \llbracket LHS \rrbracket(\rho) &= \llbracket Q_0 \rrbracket(M_0(\llbracket P_0 \rrbracket(M_0 \rho M_0^\dagger) + \llbracket P_1 \rrbracket(M_1 \rho M_1^\dagger)))M_0^\dagger \\ &\quad + \llbracket Q_1 \rrbracket(M_1(\llbracket P_0 \rrbracket(M_0 \rho M_0^\dagger) + \llbracket P_1 \rrbracket(M_1 \rho M_1^\dagger)))M_1^\dagger \\ &= \llbracket Q_0 \rrbracket(\llbracket P_0 \rrbracket(M_0 \rho M_0^\dagger)) + \llbracket Q_1 \rrbracket(\llbracket P_1 \rrbracket(M_1 \rho M_1^\dagger)) \\ &= \llbracket RHS \rrbracket(\rho). \end{aligned}$$

(5) For any density operator  $\rho$ , by Proposition 3.1(4) and (5) we have:

$$\begin{aligned} \llbracket (P_0 \triangleleft M \triangleright P_1); P \rrbracket(\rho) &= \llbracket P \rrbracket(\llbracket (P_0 \triangleleft M \triangleright P_1) \rrbracket(\rho)) \\ &= \llbracket P \rrbracket(\llbracket P_0 \rrbracket(M_0 \rho M_0^\dagger) + \llbracket P_1 \rrbracket(M_1 \rho M_1^\dagger)) \\ &= \llbracket P \rrbracket(\llbracket P_0 \rrbracket(M_0 \rho M_0^\dagger)) + \llbracket P \rrbracket(\llbracket P_1 \rrbracket(M_1 \rho M_1^\dagger)) \\ &= \llbracket P_0; P \rrbracket(M_0 \rho M_0^\dagger) + \llbracket P_1; P \rrbracket(M_1 \rho M_1^\dagger) \\ &= \llbracket (P_0; P) \triangleleft M \triangleright (P_1; P) \rrbracket. \end{aligned}$$

(6) is a special case of (7).

(7) For any  $\rho$ , using Proposition 3.1(3–5) we obtain:

$$\begin{aligned} \llbracket P; (P_0 \triangleleft M \triangleright P_1) \rrbracket(\rho) &= \llbracket P_0 \triangleleft M \triangleright P_1 \rrbracket(\llbracket P \rrbracket \rho \llbracket P \rrbracket^\dagger) \\ &= \llbracket P_0 \rrbracket(M_0 \llbracket P \rrbracket(\rho) M_0^\dagger) + \llbracket P_1 \rrbracket(M_1 \llbracket P \rrbracket(\rho) M_1^\dagger) \\ &= \llbracket P_0 \rrbracket(\llbracket P' \rrbracket(N_0 \rho N_0^\dagger)) + \llbracket P_1 \rrbracket(\llbracket P' \rrbracket(N_1 \rho N_1^\dagger)) \\ &= \llbracket P'; P_0 \rrbracket(N_0 \rho N_0^\dagger) + \llbracket P'; P_1 \rrbracket(N_1 \rho N_1^\dagger) \\ &= \llbracket (P'; P_0) \triangleleft M \triangleright (P'; P_1) \rrbracket(\rho). \end{aligned}$$

Here, the third equality follows from the assumption that  $P; [M] \equiv [N]; P'$  and  $P; (M) \equiv (N); P'$ .  $\square$

PROOF OF PROPOSITION 5.5. Since  $M_i = |\psi_i\rangle\langle\psi_i|$ , we have:

$$\begin{aligned} \llbracket LHS \rrbracket(\rho) &= \sum_i \llbracket P_i \rrbracket \left( M_i \left( \sum_j |\psi_j\rangle\langle\psi_j| \otimes \llbracket C_j \rrbracket \right) \rho \left( \sum_j |\psi_j\rangle\langle\psi_j| \otimes \llbracket C_j \rrbracket \right)^\dagger M_i^\dagger \right) \\ &= \sum_i \llbracket P_i \rrbracket \left( \left( \sum_j |\psi_i\rangle\langle\psi_i| \cdot |\psi_j\rangle\langle\psi_j| \otimes \llbracket C_j \rrbracket \right) \rho \left( \sum_j |\psi_j\rangle\langle\psi_j| \cdot |\psi_i\rangle\langle\psi_i| \otimes \llbracket C_j \rrbracket \right)^\dagger \right) \\ &= \sum_i \llbracket P_i \rrbracket \left( (|\psi_i\rangle\langle\psi_i| \otimes \llbracket C_i \rrbracket) \rho (|\psi_i\rangle\langle\psi_i| \otimes \llbracket C_i \rrbracket)^\dagger \right) \\ &= \sum_i \llbracket P_i \rrbracket \left( \llbracket C_i \rrbracket (M_i \rho M_i^\dagger) \llbracket C_i \rrbracket^\dagger \right) \\ &= \llbracket RHS \rrbracket(\rho). \end{aligned}$$

$\square$

### A.3 Proofs of the Laws in Section 6

PROOF OF PROPOSITION 6.2. It was shown in [77] (see Proposition 3.3.6 therein) that for each quantum operation  $\mathcal{E} \in \mathcal{QO}(\mathcal{H}_{QV})$ , there exists a quantum program  $P_{\mathcal{E}}$  such that  $\llbracket P_{\mathcal{E}} \rrbracket = \mathcal{E}$ . Therefore, we can define a mapping  $\mathbb{F} : \mathcal{QO}(\mathcal{H}_{QV}) \rightarrow \mathcal{QO}(\mathcal{H}_{QV})$  by

$$\mathbb{F}(\mathcal{E}) = \llbracket F(P_{\mathcal{E}}) \rrbracket. \quad (22)$$

By induction on the structure of  $F$ , it is straightforward to show that (22) is well-defined in the sense that it does not depend on the choice of  $P_{\mathcal{E}}$ , and  $\mathbb{F}$  is continuous in the CPO  $(\mathcal{QO}(\mathcal{H}_{QV}), \sqsubseteq)$ .

(1) We compute:

$$\begin{aligned} \llbracket F(\mu X.F(X)) \rrbracket &= \mathbb{F} \left( \bigsqcup_{n=0}^{\infty} \llbracket F^{(n)} \rrbracket \right) = \bigsqcup_{n=0}^{\infty} \mathbb{F} \left( \llbracket F^{(n)} \rrbracket \right) \\ &= \bigsqcup_{n=0}^{\infty} \llbracket F^{(n+1)} \rrbracket = \bigsqcup_{n=1}^{\infty} \llbracket F^{(n)} \rrbracket \\ &= \bigsqcup_{n=0}^{\infty} \llbracket F^{(n)} \rrbracket = \llbracket \mu X.F(X) \rrbracket. \end{aligned}$$

(2) To prove  $\mu X.F(X) \sqsubseteq P$ , it suffices to show that  $\llbracket F^{(n)} \rrbracket \sqsubseteq \llbracket P \rrbracket$  for every  $n \geq 0$ . This follows easily by induction on  $n$ .  $\square$

PROOF OF PROPOSITION 6.3. (1)

$$\begin{aligned} \llbracket M \triangleright (P; (M * P)) \rrbracket(\rho) &= M_0 \rho M_0^\dagger + \llbracket P; (M * P) \rrbracket \left( M_1 \rho M_1^\dagger \right) \\ &= \mathcal{E}_0(\rho) + \sum_{n=0}^{\infty} (\mathcal{E}_0 \circ (\llbracket P \rrbracket \circ \mathcal{E}_1)^n) ((\llbracket P \rrbracket \circ \mathcal{E}_1)(\rho)) \\ &= \sum_{n=0}^{\infty} (\mathcal{E}_0 \circ (\llbracket P \rrbracket \circ \mathcal{E}_1)^n) (\rho) \\ &= \llbracket (M * P) \rrbracket. \end{aligned}$$

(2) For any density operator  $\rho$ , we first note that

$$\begin{aligned} \mathcal{E}_0(N_1 \rho N_1^\dagger) &= M_0 N_1 \rho N_1^\dagger M_0^\dagger = 0, \\ \mathcal{E}_1(N_1 \rho N_1^\dagger) &= M_1 N_1 \rho N_1^\dagger M_1^\dagger = N_1 \rho N_1^\dagger, \end{aligned}$$

from the assumption of  $M \propto N$  and Lemma A.2. Together with Proposition 3.1(4–6), this yields:

$$\begin{aligned} \llbracket N \triangleright (P; (M * P)) \rrbracket(\rho) &= N_0 \rho N_0^\dagger + \llbracket P; (M * P) \rrbracket \left( N_1 \rho N_1^\dagger \right) \\ &= N_0 \rho N_0^\dagger + \sum_{n=0}^{\infty} (\mathcal{E}_0 \circ (\llbracket P \rrbracket \circ \mathcal{E}_1)^n) \left( \llbracket P \rrbracket (N_1 \rho N_1^\dagger) \right) \\ &= N_0 \rho N_0^\dagger + \sum_{n=0}^{\infty} (\mathcal{E}_0 \circ (\llbracket P \rrbracket \circ \mathcal{E}_1)^n) \left( \llbracket P \rrbracket (\mathcal{E}_1(N_1 \rho N_1^\dagger)) \right) \\ &= N_0 \rho N_0^\dagger + \sum_{n=0}^{\infty} (\mathcal{E}_0 \circ (\llbracket P \rrbracket \circ \mathcal{E}_1)^{n+1}) \left( N_1 \rho N_1^\dagger \right) \end{aligned}$$

$$\begin{aligned}
&= N_0 \rho N_0^\dagger + \mathcal{E}_0 \left( N_1 \rho N_1^\dagger \right) + \sum_{n=1}^{\infty} (\mathcal{E}_0 \circ (\llbracket P \rrbracket \circ \mathcal{E}_1)^n) (N_1 \rho N_1^\dagger) \\
&= N_0 \rho N_0^\dagger + \sum_{n=0}^{\infty} (\mathcal{E}_0 \circ (\llbracket P \rrbracket \circ \mathcal{E}_1)^n) \left( N_1 \rho N_1^\dagger \right) \\
&= \llbracket N \triangleright (M * P) \rrbracket (\rho).
\end{aligned}$$

(3) For any density operator  $\rho$ , we first note that

$$\begin{aligned}
\mathcal{E}_0(N_1 \rho N_1^\dagger) &= M_0 N_1 \rho N_1^\dagger M_0^\dagger = (M^\perp)_0 N_1 \rho N_1^\dagger (M^\perp)_1^\dagger = N_1 \rho N_1^\dagger, \\
\mathcal{E}_1(N_1 \rho N_1^\dagger) &= M_1 N_1 \rho N_1^\dagger M_1^\dagger = (M^\perp)_0 N_1 \rho N_1^\dagger (M^\perp)_0^\dagger = 0,
\end{aligned}$$

from the assumption of  $M^\perp \propto N$  and Lemma A.2. Therefore, by Proposition 3.1(5, 6) we have:

$$\begin{aligned}
\llbracket N \triangleright (M * P) \rrbracket (\rho) &= N_0 \rho N_0^\dagger + \llbracket M * P \rrbracket \left( N_1 \rho N_1^\dagger \right) \\
&= N_0 \rho N_0^\dagger + \sum_{n=0}^{\infty} (\mathcal{E}_0 \circ (\llbracket P \rrbracket \circ \mathcal{E}_1)^n) \left( N_1 \rho N_1^\dagger \right) \\
&= N_0 \rho N_0^\dagger + \mathcal{E}_0 \left( N_1 \rho N_1^\dagger \right) + \sum_{n=1}^{\infty} (\mathcal{E}_0 \circ (\llbracket P \rrbracket \circ \mathcal{E}_1)^{n-1}) \left( \llbracket P \rrbracket \left( \mathcal{E}_1 \left( N_1 \rho N_1^\dagger \right) \right) \right) \\
&= N_0 \rho N_0^\dagger + N_1 \rho N_1^\dagger + \sum_{n=1}^{\infty} (\mathcal{E}_0 \circ (\llbracket P \rrbracket \circ \mathcal{E}_1)^{n-1}) (\llbracket P \rrbracket (0)) \\
&= N_0 \rho N_0^\dagger + N_1 \rho N_1^\dagger \\
&= \llbracket N \triangleright \text{skip} \rrbracket (\rho).
\end{aligned}$$

(4) For any density operator  $\rho$ , we first note that

$$\begin{aligned}
N_0 \mathcal{E}_0(\rho) N_0^\dagger &= N_0 M_0 \rho M_0^\dagger N_0^\dagger = M_0 \rho M_0^\dagger = \mathcal{E}_0(\rho), \\
N_1 \mathcal{E}_0(\rho) N_1^\dagger &= N_1 M_0 \rho M_0^\dagger N_1^\dagger = 0,
\end{aligned}$$

from the assumption of  $N \gg M$  and Lemma A.2. Thus,

$$\begin{aligned}
\llbracket RHS \rrbracket (\rho) &= N_0 \left( \sum_n (\mathcal{E}_0 \circ (\llbracket P \rrbracket \circ \mathcal{E}_1)^n) (\rho) \right) N_0^\dagger + N_1 \left( \sum_n (\mathcal{E}_0 \circ (\llbracket P \rrbracket \circ \mathcal{E}_1)^n) (\rho) \right) N_1^\dagger \\
&= \sum_n N_0 \mathcal{E}_0 \left( ((\llbracket P \rrbracket \circ \mathcal{E}_1)^n) (\rho) \right) N_0^\dagger + \sum_n N_1 \mathcal{E}_0 \left( ((\llbracket P \rrbracket \circ \mathcal{E}_1)^n) (\rho) \right) N_1^\dagger \\
&= \sum_n \mathcal{E}_0 \left( ((\llbracket P \rrbracket \circ \mathcal{E}_1)^n) (\rho) \right) \\
&= \llbracket LHS \rrbracket (\rho).
\end{aligned}$$

□

#### A.4 Proofs of the Laws in Section 7

PROOF OF PROPOSITION 7.1. The proofs of (1), (2), and (3) are straightforward.

(4) Let  $\mathcal{E}_0, \mathcal{E}_1$  be the quantum operations defined by the measurement  $M = \{M_0, M_1\}$ ; i.e.,  $\mathcal{E}_i(\rho) = M_i \rho M_i^\dagger$  for all density operators  $\rho$ . Using Proposition 3.1(5), we have:

$$\begin{aligned} \llbracket (P \sqcup Q) \triangleleft M \triangleright R \rrbracket &= \{\mathcal{E} \circ \mathcal{E}_0 + \mathcal{F} \circ \mathcal{E}_1 \mid \mathcal{E} \in \llbracket P \sqcup Q \rrbracket \text{ and } \mathcal{F} \in \llbracket R \rrbracket\} \\ &= \{\mathcal{E} \circ \mathcal{E}_0 + \mathcal{F} \circ \mathcal{E}_1 \mid \mathcal{E} \in \llbracket P \rrbracket \cup \llbracket Q \rrbracket \text{ and } \mathcal{F} \in \llbracket R \rrbracket\} \\ &= \{\mathcal{E} \circ \mathcal{E}_0 + \mathcal{F} \circ \mathcal{E}_1 \mid \mathcal{E} \in \llbracket P \rrbracket \text{ and } \mathcal{F} \in \llbracket R \rrbracket\} \\ &\quad \cup \{\mathcal{E} \circ \mathcal{E}_0 + \mathcal{F} \circ \mathcal{E}_1 \mid \mathcal{E} \in \llbracket Q \rrbracket \text{ and } \mathcal{F} \in \llbracket R \rrbracket\} \\ &= \llbracket P \triangleleft M \triangleright R \rrbracket \cup \llbracket Q \triangleleft M \triangleright R \rrbracket \\ &= \llbracket (P \triangleleft M \triangleright R) \sqcup (Q \triangleleft M \triangleright R) \rrbracket. \end{aligned}$$

This proves the first equivalence. The second equivalence can be proved in a similar way.

(5) We prove only the first equivalence; the second follows analogously. It follows from Proposition 3.1(4) that

$$\begin{aligned} \llbracket P; (Q \sqcup R) \rrbracket &= \{\mathcal{F} \circ \mathcal{E} \mid \mathcal{E} \in \llbracket P \rrbracket \text{ and } \mathcal{F} \in \llbracket Q \sqcup R \rrbracket\} \\ &= \{\mathcal{F} \circ \mathcal{E} \mid \mathcal{E} \in \llbracket P \rrbracket \text{ and } \mathcal{F} \in \llbracket Q \rrbracket \cup \llbracket R \rrbracket\} \\ &= \{\mathcal{F} \circ \mathcal{E} \mid \mathcal{E} \in \llbracket P \rrbracket \text{ and } \mathcal{F} \in \llbracket Q \rrbracket\} \cup \{\mathcal{F} \circ \mathcal{E} \mid \mathcal{E} \in \llbracket P \rrbracket \text{ and } \mathcal{F} \in \llbracket R \rrbracket\} \\ &= \llbracket P; Q \rrbracket \cup \llbracket P; R \rrbracket \\ &= \llbracket (P; Q) \sqcup (P; R) \rrbracket. \end{aligned}$$

□

## B Detailed Proof of Theorem 9.1

To provide a detailed proof of Theorem 9.1, we first introduce several lemmas. Let us first recall the following lemma from [48]:

**LEMMA B.1.** *Consider any quantum measurement  $M = \{M_i\}_{i \in I}$  on a Hilbert space  $\mathcal{H}$ , and an auxiliary system with a Hilbert space  $\mathcal{H}'$  of dimension  $n = |I|$ . Let  $\{|i\rangle\}_{i \in I}$  be an orthonormal basis of  $\mathcal{H}'$ . Then there exists a unitary operator  $U_M$  on  $\mathcal{H} \otimes \mathcal{H}'$  such that the measurement  $M$  can be equivalently implemented by:*

- (1) *The auxiliary system is initialised in  $|0\rangle$  (one of the basis states);*
- (2) *Perform the unitary  $U_M$  on the primary and auxiliary system;*
- (3) *Measure the auxiliary system in the basis  $\{|i\rangle\}_{i \in I}$ .*

*If the initial state is  $\rho$ , then with probability  $\text{tr}(M_i \rho M_i^\dagger)$ , we obtain outcome  $i$ , and the corresponding post-measurement state is  $|i\rangle\langle i| \otimes M_i \rho M_i^\dagger$ .*

In general, a POVM measurement can be implemented by the scheme of Lemma B.1. So,  $U_M$  can be understood as necessary information to implement the measurement  $M$ .

Applying the above lemma to **if**-statements, we obtain:

**LEMMA B.2 (EQUIVALENT IMPLEMENTATION OF IF STATEMENT).** *For an arbitrary if statement  $\text{if } M[\bar{q}] (\square m \rightarrow P_m) \mathbf{fi}$  where  $M = \{M_i\}_{i \in I}$ , let  $q_a$  be an arbitrary fresh auxiliary variable with Hilbert space of dimension  $|I|$  and initialised in basis state  $|0\rangle$ . Then:*

$$\text{if } M[\bar{q}] (\square m \rightarrow P_m) \mathbf{fi} \equiv U_M[\bar{q}, q_a]; \text{if } [q_a] (\square m \rightarrow P_m) \mathbf{fi}.$$

*More precisely, if the auxiliary system is taken into account, then:*

$$q_a := |0\rangle; \text{if } M[\bar{q}] (\square m \rightarrow P_m) \mathbf{fi} \equiv q_a := |0\rangle; U_M[\bar{q}, q_a]; \text{if } [q_a] (\square m \rightarrow P_m) \mathbf{fi}; q_a := |0\rangle.$$

**PROOF.** This is a direct calculation of the denotational semantics. □

The following proposition presents some generalised versions of laws (Sequentiality) and (Left-Distributivity) in Proposition 5.4, a special case of law (if-Expansion) in Proposition 5.1, as well as laws for SWAP. They are needed in the proof of Theorem 9.1.

PROPOSITION B.1. (1) (Sequentiality) Let  $M$  be a projective measurement. If for all  $m$ ,  $\bar{q} \cap qv(P_m) = \emptyset$ , then

$$\mathbf{if} M[\bar{q}] (\square m \rightarrow P_m; Q_m) \mathbf{fi} \equiv \mathbf{if} M[\bar{q}] (\square m \rightarrow P_m) \mathbf{fi}; \mathbf{if} M[\bar{q}] (\square m \rightarrow Q_m) \mathbf{fi}.$$

(2) (Left-Distributivity) If  $\bar{q} \cap qv(P) = \emptyset$ , then

$$P; \mathbf{if} M[\bar{q}] (\square m \rightarrow P_m) \mathbf{fi} \equiv \mathbf{if} M[\bar{q}] (\square m \rightarrow P; P_m) \mathbf{fi}.$$

(3) (If-Restrictive) If  $\bar{q} \cap qv(P) = \emptyset$ , then

$$\mathbf{if} M[\bar{q}] (\square m \rightarrow P) \mathbf{fi}; \bar{q} := |\phi\rangle \equiv \bar{q} := |\phi\rangle; P.$$

(4) (SWAP-Commutative)  $\text{SWAP}[\bar{q}_1, \bar{q}_2] \equiv \text{SWAP}[\bar{q}_2, \bar{q}_1]$ . Furthermore, if  $qv(P) \subseteq \bar{q}_1$ , then

$$\text{SWAP}[\bar{q}_1, \bar{q}_2]; P \equiv P[\bar{q}_2/\bar{q}_1]; \text{SWAP}[\bar{q}_1, \bar{q}_2].$$

(5) (SWAP-Cancellation)

$$\bar{q}_1 := |\phi_1\rangle; \bar{q}_2 := |\phi_2\rangle; \text{SWAP}[\bar{q}_1, \bar{q}_2] \equiv \bar{q}_1 := |\phi_2\rangle; \bar{q}_2 := |\phi_1\rangle.$$

PROOF. Similar to the proofs of Proposition 5.4. □

Now we are ready to present the detailed proof of Theorem 9.1.

PROOF OF THEOREM 9.1. We proceed by induction on the structure of  $P$ . The strategy is that we prove Eq. (20) and  $qv(C) \subseteq qv(P) \cup \bar{q}_a$  simultaneously.

**Case 1.**  $P = \text{skip}$  or  $\text{abort}$  or  $P = C$  (quantum circuit). We introduce an auxiliary qubit  $q_a$ . Then:

$$\begin{aligned} q_a &:= |0\rangle; \text{skip} \equiv q_a := |0\rangle; \text{skip}; \mathbf{if} [q_a] (\square m \rightarrow \text{skip}) \mathbf{fi}; q_a := |0\rangle, \\ q_a &:= |0\rangle; \text{abort} \equiv q_a := |0\rangle; \text{abort}; \mathbf{if} [q_a] (\square m \rightarrow \text{abort}) \mathbf{fi}; q_a := |0\rangle, \\ q_a &:= |0\rangle; C \equiv q_a := |0\rangle; C; \mathbf{if} [q_a] (\square m \rightarrow \text{skip}) \mathbf{fi}; q_a := |0\rangle. \end{aligned}$$

It is worth noting that, in order to minimise the number of auxiliary qubits, it is not necessary to introduce  $q_a$  for each **skip**, or **abort**, or  $C$  if they are embedded within a branch of **if**-statement or sequential composition.

**Case 2.**  $P \equiv q := |\phi\rangle$ . We introduce an auxiliary quantum variable  $q_a$  that has the same dimension as  $q$ . Let  $U_{|\phi\rangle}$  be the unitary that prepares  $|\phi\rangle$ , i.e.,  $U_{|\phi\rangle}|0\rangle = |\phi\rangle$ , noting that

$$q_a := |0\rangle; q := |\phi\rangle \equiv q_a := |0\rangle; (U_{|\phi\rangle}[q_a]; \text{SWAP}[q_a, q]); \mathbf{if} [q_a] (\square m \rightarrow \text{skip}) \mathbf{fi}; q_a := |0\rangle$$

by observing the following derivation:

$$\begin{aligned} \text{RHS} &\equiv q_a := |0\rangle; (U_{|\phi\rangle}[q_a]; \text{SWAP}[q_a, q]); \mathbf{if} [q_a] (\square m \rightarrow \text{skip}) \mathbf{fi}; q_a := |0\rangle \\ &\equiv q_a := |0\rangle; U_{|\phi\rangle}[q_a]; \text{SWAP}[q_a, q]; q_a := |0\rangle; \text{skip} \\ &\equiv q_a := |\phi\rangle; \text{SWAP}[q_a, q]; q_a := |0\rangle \\ &\equiv q_a := |\phi\rangle; q := |0\rangle; \text{SWAP}[q_a, q] \\ &\equiv q_a := |0\rangle; q := |\phi\rangle. \end{aligned}$$

**Case 3.**  $P = P_1; P_2$ . By the induction hypothesis, we assume that  $P_1, P_2$  have the following normal forms:

$$\begin{aligned} \bar{q}_{a1} &:= |0\rangle; P_1 \equiv \bar{q}_{a1} := |0\rangle; C_1; \mathbf{if} [\bar{q}_{a1}] (\square m_1 \rightarrow P_{m1}) \mathbf{fi}; \bar{q}_{a1} := |0\rangle \\ \bar{q}_{a2} &:= |0\rangle; P_2 \equiv \bar{q}_{a2} := |0\rangle; C_2; \mathbf{if} [\bar{q}_{a2}] (\square m_2 \rightarrow P_{m2}) \mathbf{fi}; \bar{q}_{a2} := |0\rangle. \end{aligned}$$

Since we have sufficient auxiliary variables and the construction does not depend on specific auxiliary variables, we can use disjoint auxiliary variables for each normal form, ensuring that  $\bar{q}_{a1} \cap \bar{q}_{a2} = \emptyset$ . Set  $\bar{q}_a \triangleq \bar{q}_{a1}, \bar{q}_{a2}$ . Then we claim:

$$\bar{q}_a := |0\rangle; (P1; P2) \equiv \bar{q}_a := |0\rangle; (C_1; C_2); \text{if } [\bar{q}_a] (\Box(m_1, m_2) \rightarrow P_{m_1 m_2}) \text{ fi}; \bar{q}_a := |0\rangle$$

where  $P_{m_1 m_2} = \text{skip}$  if both  $P_{m_1} = P_{m_2} = \text{skip}$ , and  $P_{m_1 m_2} = \text{abort}$  otherwise (i.e., if  $P_{m_1} = \text{abort}$  or  $P_{m_2} = \text{abort}$ ). This can be proved by observing that:

$$\begin{aligned} & \bar{q}_a := |0\rangle; (P1; P2) \\ \equiv & \bar{q}_a := |0\rangle; \bar{q}_{a1} := |0\rangle; \bar{q}_{a2} := |0\rangle; P1; P2 \\ \equiv & \bar{q}_a := |0\rangle; (\bar{q}_{a1} := |0\rangle; P1); (\bar{q}_{a2} := |0\rangle; P2) \\ \equiv & \bar{q}_a := |0\rangle; \bar{q}_{a1} := |0\rangle; C_1; \text{if } [\bar{q}_{a1}] (\Box m_1 \rightarrow P_{m_1}) \text{ fi}; \bar{q}_{a1} := |0\rangle; \\ & \bar{q}_{a2} := |0\rangle; C_2; \text{if } [\bar{q}_{a2}] (\Box m_2 \rightarrow P_{m_2}) \text{ fi}; \bar{q}_{a2} := |0\rangle \\ \equiv & \bar{q}_a := |0\rangle; C_1; \text{if } [\bar{q}_{a1}] (\Box m_1 \rightarrow P_{m_1}) \text{ fi}; \\ & C_2; \text{if } [\bar{q}_{a2}] (\Box m_2 \rightarrow P_{m_2}) \text{ fi}; \bar{q}_a := |0\rangle \\ \equiv & \bar{q}_a := |0\rangle; (C_1; C_2); \text{if } [\bar{q}_{a1}] (\Box m_1 \rightarrow P_{m_1}) \text{ fi}; \\ & \text{if } [\bar{q}_{a2}] (\Box m_2 \rightarrow P_{m_2}) \text{ fi}; \bar{q}_a := |0\rangle \\ \equiv & \bar{q}_a := |0\rangle; (C_1; C_2); \text{if } [\bar{q}_{a1}] (\Box m_1 \rightarrow \text{if } [\bar{q}_{a2}] (\Box m_2 \rightarrow P_{m_1}; P_{m_2}) \text{ fi}) \text{ fi}; \bar{q}_a := |0\rangle \\ \equiv & \bar{q}_a := |0\rangle; (C_1; C_2); \text{if } [\bar{q}_{a1}, \bar{q}_{a2}] (\Box(m_1, m_2) \rightarrow P_{m_1}; P_{m_2}) \text{ fi}; \bar{q}_a := |0\rangle \\ \equiv & \bar{q}_a := |0\rangle; (C_1; C_2); \text{if } [\bar{q}_a] (\Box(m_1, m_2) \rightarrow P_{m_1 m_2}) \text{ fi}; \bar{q}_a := |0\rangle \end{aligned}$$

For the variable sets, it suffices to note that

$$qv(C_1; C_2) = qv(C_1) \cup qv(C_2) \subseteq (qv(P_1) \cup \bar{q}_{a1}) \cup (qv(P_2) \cup \bar{q}_{a2}) = (qv(P_1; P_2) \cup \bar{q}_a).$$

**Case 4.**  $P = \text{if } M[\bar{q}] (\Box i \rightarrow P_i) \text{ fi}$ . This case was already proved in Section 9. What remains is to check the condition on variable sets. Indeed, we have:

$$\begin{aligned} & qv(U_M[\bar{q}, a]; \text{qif } [a] (\Box |i\rangle \rightarrow C_i) \text{ fi}) = \bar{q} \cup a \cup \bigcup_i qv(C_i) \\ \subseteq & q \cup \bigcup_i (qv(P_i) \cup \bar{r}_i) \cup a = \left( q \cup \bigcup_i qv(P_i) \right) \cup \left( a \cup \bigcup_i \bar{r}_i \right) = qv(P) \cup \bar{q}_a. \end{aligned}$$

□

## C A Discussion about Preservation of Refinement by Loop

In Section 8, we showed that refinement is preserved by sequential composition, **if**-statement and nondeterministic choice. Here, we further conjecture:

CONJECTURE C.1 (REFINEMENT RULE FOR LOOPS). *If  $P \sqsubseteq Q$ , then*

$$M * P \sqsubseteq M * Q.$$

At this point, however, we have not yet proved it. Let us examine where the difficulty arises. By definition, we have:

$$\llbracket M * P \rrbracket = \left\{ \sum_{i=0}^{\infty} \mathcal{E}_0 \circ (\mathcal{F}_i \circ \mathcal{E}_1) \circ \dots \circ (\mathcal{F}_1 \circ \mathcal{E}_1) : \mathcal{F}_1, \dots, \mathcal{F}_i, \dots \in \llbracket P \rrbracket \right\}.$$

Hence, we need to show that for any  $\mathcal{F}_1, \dots, \mathcal{F}_i, \dots \in \llbracket P \rrbracket$ , we have

$$\sum_{i=0}^{\infty} \mathcal{E}_0 \circ (\mathcal{F}_i \circ \mathcal{E}_1) \circ \dots \circ (\mathcal{F}_1 \circ \mathcal{E}_1) \in \overline{\text{Con}(\llbracket Q \rrbracket)}.$$

Indeed, since  $P \sqsubseteq Q$ , for each  $i$ , it follows from  $\mathcal{F}_i \in \llbracket P \rrbracket$  that  $\mathcal{F}_i \in \overline{\text{Con}(\llbracket Q \rrbracket)}$ , and

$$\mathcal{F}_i = \sum_{j_i} p_{ij_i} G_{ij_i}$$

for some (finite or countably infinite) probability distribution  $\{p_{ij_i}\}$  and  $G_{ij_i} \in \llbracket Q \rrbracket$ . Therefore, we have:

$$\sum_{i=0}^{\infty} \mathcal{E}_0 \circ (\mathcal{F}_i \circ \mathcal{E}_1) \circ \dots \circ (\mathcal{F}_1 \circ \mathcal{E}_1) = \sum_{i=0}^{\infty} \sum_{j_1, \dots, j_i} \left( \prod_{t=1}^i p_{tj_t} \right) \mathcal{E}_0 \circ (G_{ij_i} \circ \mathcal{E}_1) \circ \dots \circ (G_{1j_1} \circ \mathcal{E}_1) \quad (23)$$

However, it is not immediately clear how to interchange the order of big product and big sum; that is, it is not known whether the RHS of (23) is equivalent to  $\sum_{j_1, j_2, \dots} p_{j_1, j_2, \dots} \sum_{i=0}^{\infty} \mathcal{E}_0 \circ (G_{ij_i} \circ \mathcal{E}_1) \circ \dots \circ (G_{1j_1} \circ \mathcal{E}_1)$  for some probability distributions  $\{p_{j_1, j_2, \dots}\}$ .

#### D A ZX-Calculus Proof of Example 4.2

In this section, we demonstrate how Example 4.2 can be proved using the ZX-calculus, thus providing a concrete comparison with our approach. The rules we use are from the standard ZX-calculus, as shown in Figure 3, drawn from [36].

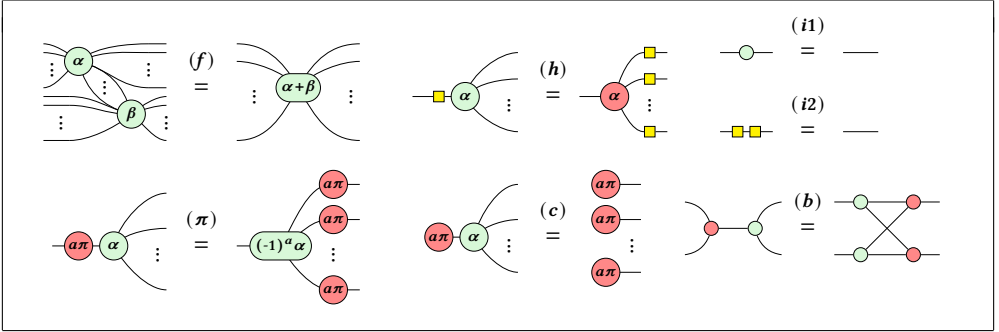
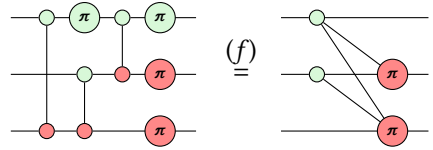


Fig. 3. A convenient presentation of the ZX-calculus rules from [36]. These rules hold for all  $\alpha, \beta \in [0, 2\pi)$ , and each rule also holds with the colours interchanged.

By rewriting the circuits as ZX-diagrams, the equivalence of the two circuits is translated into the equivalence of the following two ZX-diagrams:

$$\begin{array}{c} \text{---} \circ \text{---} \circ \text{---} \\ \text{---} \circ \text{---} \circ \text{---} \\ \text{---} \circ \text{---} \circ \text{---} \end{array} = \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \end{array} \quad (24)$$

Applying the fusion rule ( $f$ ) to merge all spiders of the same type, we immediately obtain the simplification of the LHS:



The simplification of the RHS proceeds in three steps. First, we apply the  $\pi$ -commutation rule ( $\pi$ ) to swap the intermediate  $X$  spider with the subsequent  $Z$  spider. Then, we use the strong complementarity rule ( $b$ ) to interchange the  $X$  and  $Z$  spiders. Finally, we apply the fusion rule ( $f$ ) to obtain the same ZX-diagram as above:

